EZR32HG Reference Manual

Preliminary

- 32-bit ARM Cortex-M0+ processor running at up to 25 MHz
- Up to 64 kB Flash and 8 kB RAM memory
- Energy efficient and autonomous peripherals
- Ultra low power Energy Modes with Sub-µA operation
- Fast wake-up time of only 2 µs

The EZR32HG microcontroller series revolutionizes the 8- to 32-bit market with a combination of unmatched performance and ultra low power consumption in both active- and sleep modes. EZR32HG devices consume as little as 114 µA/MHz in run mode.

EZR32HG's low energy consumption outperforms any other available 8-, 16-, and 32-bit solution. The EZR32HG includes autonomous and energy efficient peripherals, high overall chip- and analog integration, and the performance of the industry standard 32-bit ARM Cortex-M0+ processor.

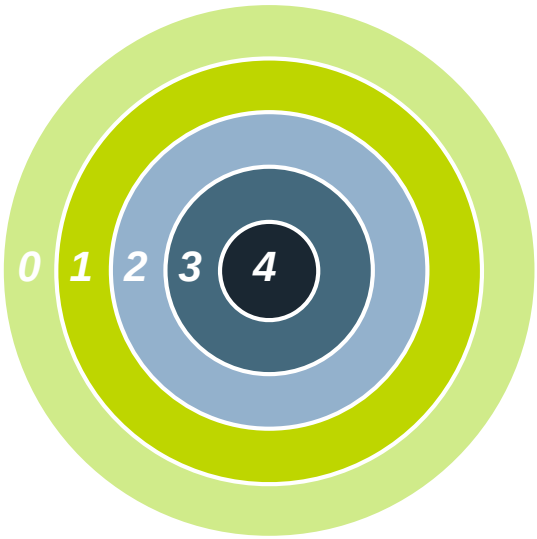
1 Energy Friendly Wireless Microcontrollers

1.1 Typical Applications

The EZR32HG wireless microcontroller is the ideal choice for demanding 8-, 16-, and 32-bit energy sensitive applications. These devices are developed to minimize the energy consumption by lowering both the power and the active time, over all phases of MCU operation. This unique combination of ultra low energy consumption and the performance of the 32-bit ARM Cortex-M0+ processor, help designers get more out of the available energy in a variety of applications.

Ultra low energy EZR32HG wireless microcontrollers are perfect for:

- Gas metering
- Energy metering
- Water metering
- Smart metering
- Alarm and security systems
- Health and fitness applications
- Industrial and home automation



1.2 EZR32HG Development

Because EZR32HG use the Cortex-M0+ CPU, embedded designers benefit from the largest development ecosystem in the industry, the ARM ecosystem. The development suite spans the whole design process and includes powerful debug tools, and some of the world's top brand compilers. Libraries with documentation and user examples shorten time from idea to market.

The range of EZR32HG devices ensure easy migration and feature upgrade possibilities.

2 About This Document

This document contains reference material for the EZR32HG series of wireless microcontrollers. All modules and peripherals in the EZR32HG series devices are described in general terms. Not all modules are present in all devices, and the feature set for each device might vary. Such differences, including pin-out, are covered in the device-specific datasheets.

2.1 Conventions

Register Names

Register names are given as a module name prefix followed by the short register name:

TIMERn_CTRL - Control Register

The "n" denotes the numeric instance for modules that might have more than one instance.

Some registers are grouped which leads to a group name following the module prefix:

GPIO_Px_DOUT - Port Data Out Register,

where *x* denotes the port instance (A,B,...).

Bit Fields

Registers contain one or more bit fields which can be 1 to 32 bits wide. Multi-bit fields are denoted with (x:y), where *x* is the start bit and *y* is the end bit.

Address

The address for each register can be found by adding the base address of the module (found in the Memory Map), and the offset address for the register (found in module Register Map).

Access Type

The register access types used in the register descriptions are explained in Table 2.1 (p. 3) .

Table 2.1. Register Access Types

Access Type	Description
R	Read only. Writes are ignored.
RW	Readable and writable.
RW1	Readable and writable. Only writes to 1 have effect.
RW1H	Readable, writable and updated by hardware. Only writes to 1 have effect.
W1	Read value undefined. Only writes to 1 have effect.
W	Write only. Read value undefined.
RWH	Readable, writable and updated by hardware.

Number format

0x prefix is used for hexadecimal numbers.

0b prefix is used for binary numbers.

Numbers without prefix are in decimal representation.

Reserved

Registers and bit fields marked with *reserved* are reserved for future use. These should be written to 0 unless otherwise stated in the Register Description. Reserved bits might be read as 1 in future devices.

Reset Value

The reset value denotes the value after reset.

Registers denoted with X have an unknown reset value and need to be initialized before use. Note that, before these registers are initialized, read-modify-write operations might result in undefined register values.

Pin Connections

Pin connections are given as a module prefix followed by a short pin name:

USn_TX (USARTn TX pin)

The pin locations referenced in this document are given in the device-specific datasheet.

2.2 Related Documentation

Further documentation on the EZR32HG family and the ARM Cortex-M0+ can be found at the Silicon Laboratories and ARM web pages:

www.silabs.com

www.arm.com

3 System Overview

3.1 Introduction

The EZR32HG Wireless MCUs are the latest in Silicon Labs family of wireless MCUs delivering a high performance, low energy wireless solution integrated into a small form factor package. By combining a high performance sub-1 GHz RF transceiver with an energy efficient 32-bit MCU, the EZR32HG family provides designers the ultimate in flexibility with a family of pin-compatible devices that scale with 64/128/256 kB of flash and support Silicon Labs EZRadio or EZRadioPRO transceivers. The ultra-low power operating modes and fast wake-up times of the Silicon Labs energy friendly 32-bit MCUs, combined with the low transmit and receive power consumption of the sub-1 GHz radio, result in a solution optimized for battery powered applications, see Figure 3.1 (p. 5) .

3.2 Block Diagram

Figure 3.1 (p. 5) shows the block diagram of EZR32HG. The color indicates peripheral availability in the different energy modes, described in Section 3.4 (p. 8) .

Figure 3.1. Block Diagram of EZR32HG

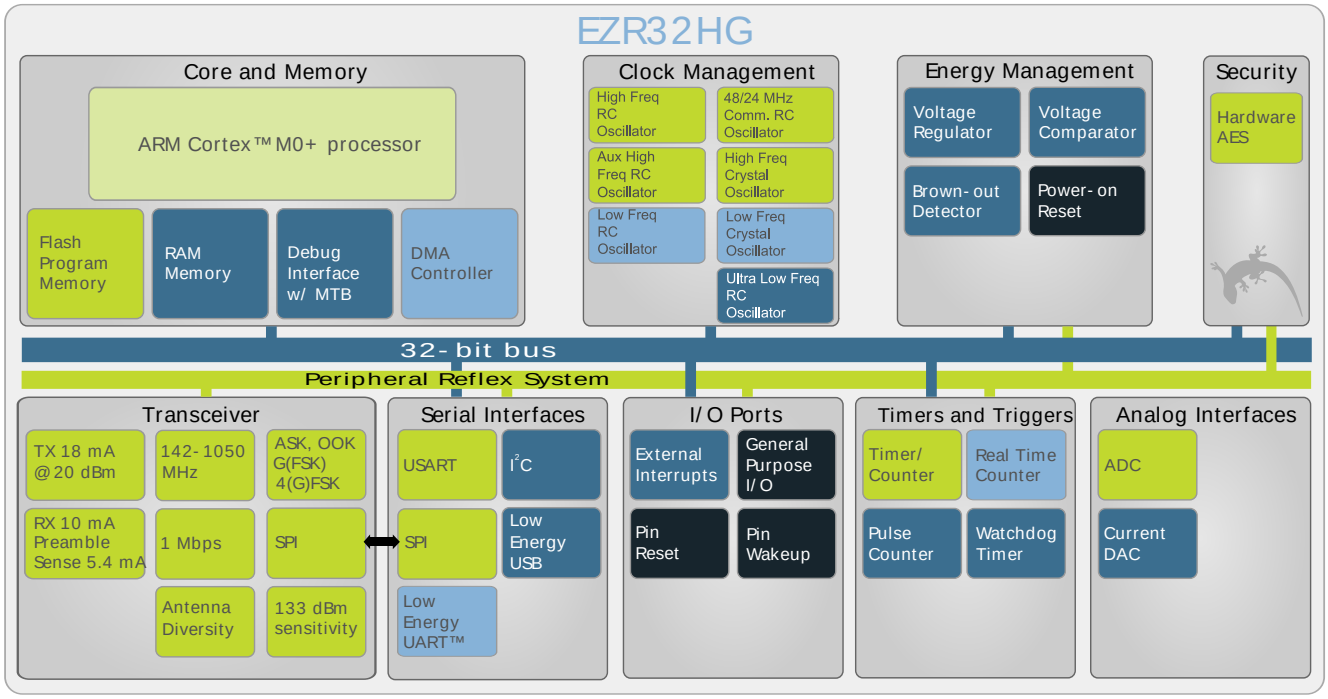
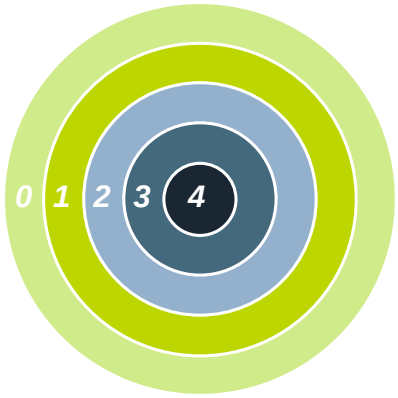


Figure 3.2. Energy Mode Indicator



Note

In the energy mode indicator, the numbers indicates Energy Mode, i.e EM0-EM4.

3.3 Features

3.3.1 MCU Features

- **ARM Cortex-M0+ CPU platform**
  - High Performance 32-bit processor @ up to 25 MHz
  - Wake-up Interrupt Controller
- **Flexible Energy Management System**
  - 20 nA @ 3 V Shutoff Mode
  - 0.5 µA @ 3 V Stop Mode, including Power-on Reset, Brown-out Detector, RAM and CPU retention
  - 0.9 µA @ 3 V Deep Sleep Mode, including RTC with 32768 Hz oscillator, Power-on Reset, Brown-out Detector, RAM and CPU retention
  - 46 µA/MHz @ 3 V Sleep Mode
  - 114 µA/MHz @ 3 V Run Mode, with code executed from flash
- **64/32 KB Flash**
- **8/4 KB RAM**
- **Up to 37 General Purpose I/O pins**
  - Configurable push-pull, open-drain, pull-up/down, input filter, drive strength
  - Configurable peripheral I/O locations
  - 16 asynchronous external interrupts
  - Output state retention and wake-up from Shutoff Mode
- **4 Channel DMA Controller**
  - Alternate/primary descriptors with scatter-gather/ping-pong operation
- **4 Channel Peripheral Reflex System**
  - Autonomous inter-peripheral signaling enables smart operation in low energy modes
- **Universal Serial Bus (USB)**
  - Fully USB 2.0 compliant
  - On-chip PHY and embedded 5V to 3.3V regulator
- **Hardware AES with 128-bit Keys in 54 cycles**
- **Communication interfaces**
  - 1× Universal Synchronous/Asynchronous Receiver/Transmitter
    - Triple buffered full/half-duplex operation
    - 4-16 data bits
  - 1× Low Energy UART
    - Autonomous operation with DMA in Deep Sleep Mode
  - 1× I^2C Interface with SMBus support
    - Address recognition in Stop Mode
- **Timers/Counters**
  - 3× 16-bit Timer/Counter
    - 3 Compare/Capture/PWM channels
    - Dead-Time Insertion on TIMER0
  - 24-bit Real-Time Counter
  - 1× 8/16-bit Pulse Counter
    - Asynchronous pulse counting/quadrature decoding
  - Watchdog Timer with dedicated RC oscillator @ 50 nA
- **Ultra low power precision analog peripherals**
  - 12-bit 1 Msamples/s Analog to Digital Converter
    - 8 input channels and on-chip temperature sensor
    - Single ended or differential operation
    - Conversion tailgating for predictable latency
  - Current Digital to Analog Converter

- Source or sink a configurable constant current
- Supply Voltage Comparator

3.3.2 RF Features

- **Frequency range = 142–1050 MHz**
- **Receive sensitivity = -133 dBm**
- **Modulation**
  - (G)FSK, 4(G)FSK, (G)MSK, OOK
- **Max output power**
  - +20 dBm
  - +16 dBm
  - +13 dBm
- **PA support for +27 or +30 dBm**
- **Low active power consumption**
  - 10/13 mA RX
  - 18 mA TX at +10 dBm
- **Preamble sense mode**
  - 6 mA average RX current at 1.2 kbps
  - 10 µA average RX current at 50 kbps and 1 sec sleep interval
- **Fast preamble detection**
  - 1 byte preamble detection
- **Data rate = 100 bps to 1 Mbps**
- **Fast wake and hop times**
- **Power supply = 1.8 to 3.8 V**
- **Excellent selectivity performance**
  - 69 dB adjacent channel
  - 79 dB blocking at 1 MHz
- **Antenna diversity and T/R switch control**
- **Highly configurable packet handler**
- **TX and RX 64 byte FIFOs**
  - 129 bytes dedicated Tx or Rx
- **Auto frequency control (AFC)**
- **Automatic gain control (AGC)**
- **Low battery detector**
- **Temperature sensor**
- **IEEE 802.15.4g and WMBus compliant**
- **Suitable for FCC Part 90 Mask D, FCC part 15.247, 15,231, 15,249, ARIB T-108, T-96, T-67, RCR STD-30, China regulatory**
- **ETSI Category I Operation EN300 220**

3.3.3 System Features

- **Ultra efficient Power-on Reset and Brown-Out Detector**
- **2-pin Serial Wire Debug interface**
- **Temperature range -40 - 85°C**
- **Single power supply 1.98 - 3.8 V**
- **Packages**
  - QFN48

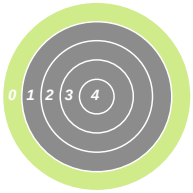
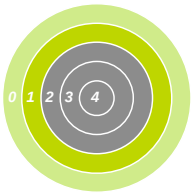
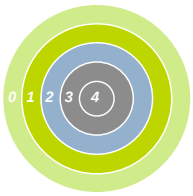
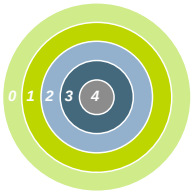
3.4 Energy Modes

There are five different Energy Modes (EM0-EM4) in the EZR32HG, see Table 3.1 (p. 8). The EZR32HG is designed to achieve a high degree of autonomous operation in low energy modes. The intelligent combination of peripherals, RAM with data retention, DMA, low-power oscillators, and short wake-up time, makes it attractive to remain in low energy modes for long periods and thus saving energy consumption.

Tip

Throughout this document, the first figure in every module description contains an Energy Mode Indicator showing which energy mode(s) the module can operate (see Table 3.1 (p. 8)).

Table 3.1. Energy Mode Description

Energy Mode	Name	Description
	EM0 – Energy Mode 0 (Run mode)	In EM0, the CPU is running and consuming as little as 114 µA/MHz, when running code from flash. All peripherals can be active.
	EM1 – Energy Mode 1 (Sleep Mode)	In EM1, the CPU is sleeping and the power consumption is only 46 µA/MHz. All peripherals, including DMA, PRS and memory system, are still available.
	EM2 – Energy Mode 2 (Deep Sleep Mode)	In EM2 the high frequency oscillator is turned off, but with the 32.768 kHz oscillator running, selected low energy peripherals (RTC, PCNT, LEUART, I^2C, USB, WDOG and ACMP) are still available. This gives a high degree of autonomous operation with a current consumption as low as 0.9 µA with RTC enabled. Power-on Reset, Brown-out Detection and full RAM and CPU retention is also included.
	EM3 - Energy Mode 3 (Stop Mode)	In EM3, the low-frequency oscillator is disabled, but there is still full CPU and RAM retention, as well as Power-on Reset, Pin reset, EM4 wake-up and Brown-out Detection, with a consumption of only 0.5 µA. The low-power ACMP, asynchronous external interrupt, PCNT, and I^2C can wake-up the device. Even in this mode, the wake-up time is a few microseconds.
	EM4 – Energy Mode 4 (Shutoff Mode)	In EM4, the current is down to 20 nA and all chip functionality is turned off except the pin reset, GPIO pin wake-up, GPIO pin retention and the Power-On Reset. All pins are put into their reset state.

3.5 Product Overview

Table 3.2 (p. 9) shows a device overview of the EZR32HG Wireless Microcontroller Series, including peripheral functionality. For more information, the reader is referred to the device specific datasheets.

Table 3.2. EZR32HG Wireless Microcontroller Series

EZR32HG Part #	Flash	RAM	GPIO(pins)	USB	LCD	USART+UART	LEUART	I²C	Timer(PWM)	LETIMER	RTC	PCNT	Watchdog	ADC(pins)	DAC(pins)	IDAC(pins)	ACMP(pins)	AES	EBI	LESENSE	Op-Amps	Package
230F32	32	8	22	-	-	2	1	1	3 (9)	-	1	1	1	1 (4)	-	1 (1)	-	Y	-	-	-	QFN48
230F64	64	8	22	-	-	2	1	1	3 (9)	-	1	1	1	1 (4)	-	1 (1)	-	Y	-	-	-	QFN48
330F32	32	8	22	Y	-	2	1	1	3 (9)	-	1	1	1	1 (4)	-	1 (1)	-	Y	-	-	-	QFN48
330F64	64	8	34	Y	-	2	1	1	3 (9)	-	1	1	1	1 (4)	-	1 (1)	-	Y	-	-	-	QFN48

3.6 Device Revision

The device revision number is read from the ROM Table. The major revision number and the chip family number is read from PID0 and PID1 registers. The minor revision number is extracted from the PID2 and PID3 registers, as illustrated in Figure 3.3 (p. 9). The Fam[5:2] and Fam[1:0] must be combined to complete the chip family number, while the Minor Rev[7:4] and Minor Rev[3:0] must be combined to form the complete revision number.

Figure 3.3. Revision Number Extraction

PID2 (**0xF00FFFE8**)		
31:8	7:4	3:0
	Minor Rev[7:4]	

PID3 (**0xF00FFFEC**)		
31:8	7:4	3:0
	Minor Rev[3:0]	

PID0 (**0xF00FFFE0**)		
31:8	7:6	5:0
	Fam[1:0]	Major Rev[5:0]

PID1 (**0xF00FFFE4**)	
31:4	3:0
	Fam[5:2]

For the latest revision of the Happy Gecko family, the chip family number is 0x05 and the major revision number is 0x01. The minor revision number is to be interpreted according to Table 3.3 (p. 9) .

Table 3.3. Minor Revision Number Interpretation

Minor Rev[7:0]	Revision
0x00	A

4 Radio Overview

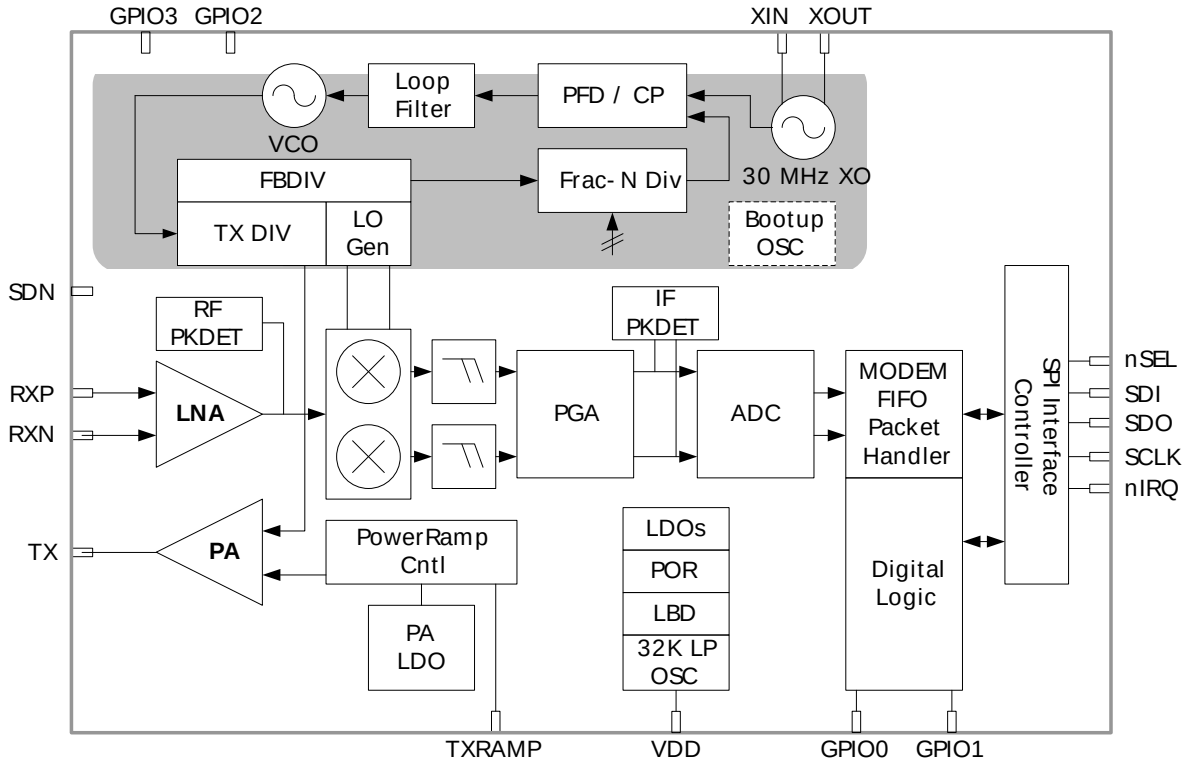
The EZR32HG family of devices is built using high-performance, low-current EZRadio and EZRadioPRO RF transceivers covering the sub-GHz frequency bands from 142 to 1050 MHz. These devices offer outstanding sensitivity of upto -133 dBm (using EZRadioPRO) while achieving extremely low active and standby current consumption. The EZR32HG devices using the EZRadioPRO transceiver offer frequency coverage in all major bands and include optimal phase noise, blocking, and selectivity performance for narrow band and licensed band applications, such as FCC Part 90 and 169 MHz wireless M-Bus. The 69 dB adjacent channel selectivity with 12.5 kHz channel spacing ensures robust receive operation in harsh RF conditions, which is particularly important for narrow band operation. The active mode TX current consumption of 18 mA at +10 dBm and RX current of 10 mA coupled with extremely low standby current and fast wake times is optimized for extended battery life in the most demanding applications. The EZR32HG devices can achieve up to +27 dBm output power with built-in ramping control of a low-cost external FET. The devices can meet worldwide regulatory standards: FCC, ETSI, and ARIB. All devices are designed to be compliant with 802.15.4g and Wireless M-Bus smart metering standards. The devices are highly flexible and can be programmed and configured via Simplicity Studio, available at www.silabs.com.

4.1 EZRadioPRO Overview

4.1.1 Introduction

Silicon Laboratories' EZRadioPRO devices are high-performance, low-current RF transceivers covering the sub-GHz frequency bands from 142 to 1050 MHz. All parts offer outstanding sensitivity of -129 dBm while achieving extremely low active and standby current consumption. EZRadioPRO devices offers frequency coverage in all major bands. The EZRadioPRO devices includes optimal phase noise, blocking, and selectivity performance for narrow band and wireless M-Bus licensed band applications, such as FCC Part90 and 169 MHz wireless M-Bus. The 69 dB adjacent channel selectivity with 12.5 kHz channel spacing ensures robust receive operation in harsh RF conditions, which is particularly important for narrow band operation. The EZRadioPRO devices offers exceptional output power of up to +20 dBm with outstanding TX efficiency. The high output power and sensitivity results in an industry-leading link budget of 146 dB allowing extended ranges and highly robust communication links. The EZRadioPRO active mode TX current consumption of 18 mA at +10 dBm and RX current of 10 mA coupled with extremely low standby current and fast wake times ensure extended battery life in the most demanding applications. The EZRadioPRO devices can achieve up to +27 dBm output power with built-in ramping control of a low-cost external FET. The devices can meet worldwide regulatory standards: FCC, ETSI, wireless M-Bus, and ARIB. All devices are designed to be compliant with 802.15.4g and Wireless M-Bus smart metering standards.

Figure 4.1. EZRadioPRO Block Diagram



4.1.2 Functional Description

The EZRadioPRO devices are high-performance, low-current, wireless ISM transceivers that cover the sub-GHz bands. The wide operating voltage range of 1.8-3.8 V and low current consumption make the EZRadioPRO an ideal solution for battery powered applications. The EZRadioPRO operates as a time division duplexing (TDD) transceiver where the device alternately transmits and receives data packets. The device uses a single-conversion mixer to downconvert the 2/4-level FSK/GFSK or OOK modulated receive signal to a low IF frequency. Following a programmable gain amplifier (PGA) the signal is converted to the digital domain by a high performance ΔΣ ADC allowing filtering, demodulation, slicing, and packet handling to be performed in the built-in DSP increasing the receiver's performance and flexibility versus analog based architectures. The demodulated signal is output to the system MCU through a programmable GPIO or via the standard SPI bus by reading the 64-byte RX FIFO.

A single high precision local oscillator (LO) is used for both transmit and receive modes since the transmitter and receiver do not operate at the same time. The LO is generated by an integrated VCO and ΔΣ Fractional-N PLL synthesizer. The synthesizer is designed to support configurable data rates from 100 bps to 1 Mbps. The EZRadioPRO devices operate in the frequency bands of 142-175, 283-350, 350-525, and 850-1050 MHz with a maximum frequency accuracy step size of 28.6 Hz. The transmit FSK data is modulated directly into the ΔΣ data stream and can be shaped by a Gaussian low-pass filter to reduce unwanted spectral content.

The EZRadioPRO contains a power amplifier (PA) that supports output power up to +20 dBm with very high efficiency, consuming only 70 mA at 169 MHz and 85 mA at 915 MHz. The integrated +20 dBm power amplifier can also be used to compensate for the reduced performance of a lower cost, lower performance antenna or antenna with size constraints due to a small form-factor. Competing solutions require large and expensive external PAs to achieve comparable performance. EZRadioPRO is designed to support single coin cell operation with current consumption below 18 mA for +10 dBm output power. Two match topologies are available for the EZRadioPRO, class-E and switched-current. Class-

E matching provides optimal current consumption, while switched-current matching demonstrates the best performance over varying battery voltage and temperature with slightly higher current consumption. The PA is single-ended to allow for easy antenna matching and low BOM cost. The PA incorporates automatic ramp-up and ramp-down control to reduce unwanted spectral spreading. The EZRadioPRO family supports frequency hopping, TX/RX switch control, and antenna diversity switch control to extend the link range and improve performance. Built-in antenna diversity and support for frequency hopping can be used to further extend range and enhance performance. Antenna diversity is completely integrated into the EZRadioPRO and can improve the system link budget by 8-10 dB, resulting in substantial range increases under adverse environmental conditions. A highly configurable packet handler allows for autonomous encoding/decoding of nearly any packet structure. Additional system features, such as an automatic wake-up timer, low battery detector, 64 byte TX/RX FIFOs, and preamble detection, reduce overall current consumption and allows for the use of lower-cost system MCUs. An integrated temperature sensor, power-on-reset (POR), and GPIOs further reduce overall system cost and size.

4.1.3 Controller Interface

4.1.3.1 Serial Peripheral Interface (SPI)

The EZRadioPRO communicates with the EFM32HG MCU over a standard 4-wire serial peripheral interface (SPI): SCLK, SDI, SDO, and nSEL. The SPI interface is designed to operate at a maximum of 10 MHz. The SPI timing parameters are demonstrated in Table 4.1 (p. 12). The host MCU writes data over the SDI pin and can read data from the device on the SDO output pin. Table 4.1 (p. 12) demonstrates an SPI write command. The nSEL pin should go low to initiate the SPI command. The first byte of SDI data will be one of the firmware commands followed by n bytes of parameter data which will be variable depending on the specific command. The rising edges of SCLK should be aligned with the center of the SDI data. For details regarding pin setup, see datasheet for the specific part.

Table 4.1. Serial Interface Timing Parameters

Symbol	Parameter	Min (ns)	Max (ns)
tCH	Clock high time	40	
tCL	Clock low time	40	
tDS	Data setup time	20	
tDH	Data hold time	20	
tDD	Output data delay time		43
tDE	Output disable time		45
tSS	Select setup time	20	
tSH	Select hold time	50	
tSW	Select high period	80	

Note

CL = 10 pF; VDD = 1.8 V; SDO Drive strength setting = 10.

Figure 4.2. Serial Interface Timing

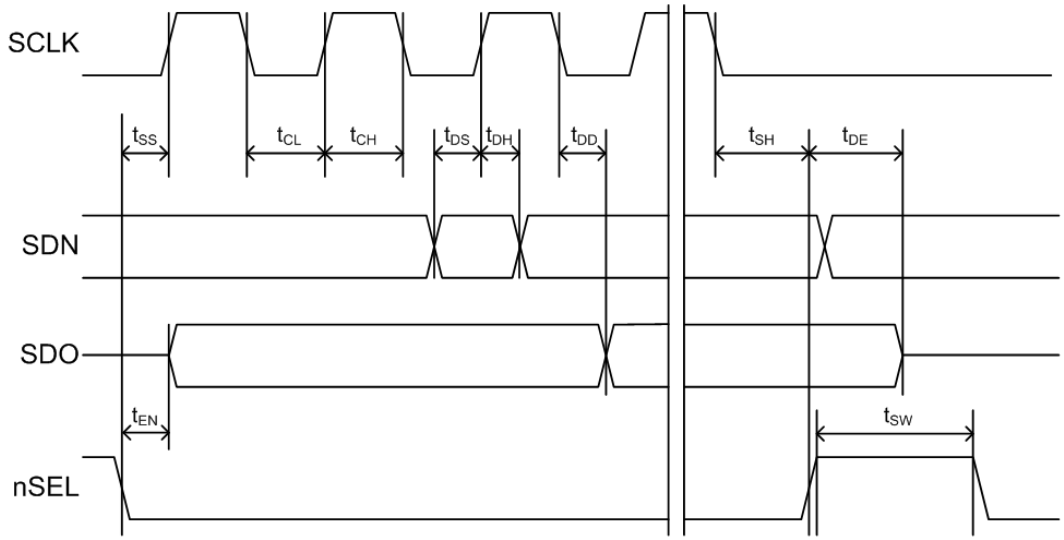
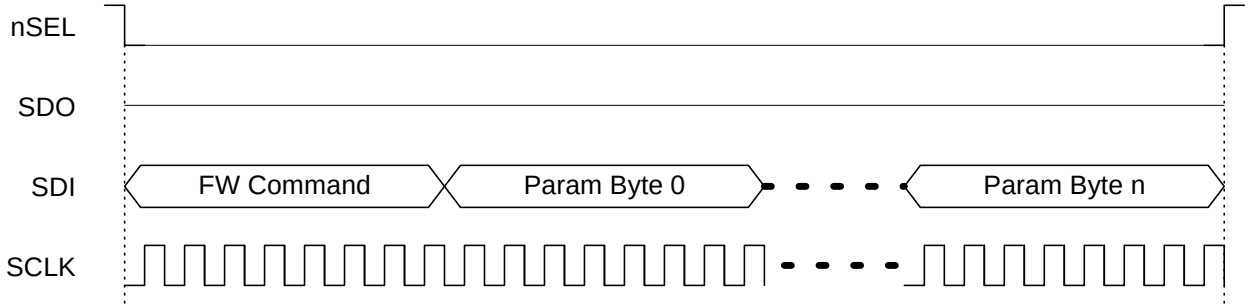


Figure 4.3. SPI Write Command



The EZRadioPRO contains an internal MCU which controls all the internal functions of the radio. For SPI read commands a typical MCU flow of checking clear-to-send (CTS) is used to make sure the internal MCU has executed the command and prepared the data to be output over the SDO pin. Figure 4.4 (p. 14) demonstrates the general flow of an SPI read command. Once the CTS value reads FFh then the read data is ready to be clocked out to the host MCU. The typical time for a valid FFh CTS reading is 20 µs. Figure 4.5 (p. 14) demonstrates the remaining read cycle after CTS is set to FFh. The internal MCU will clock out the SDO data on the negative edge so the host MCU should process the SDO data on the rising edge of SCLK.

Figure 4.4. SPI Read Command - Check CTS Value

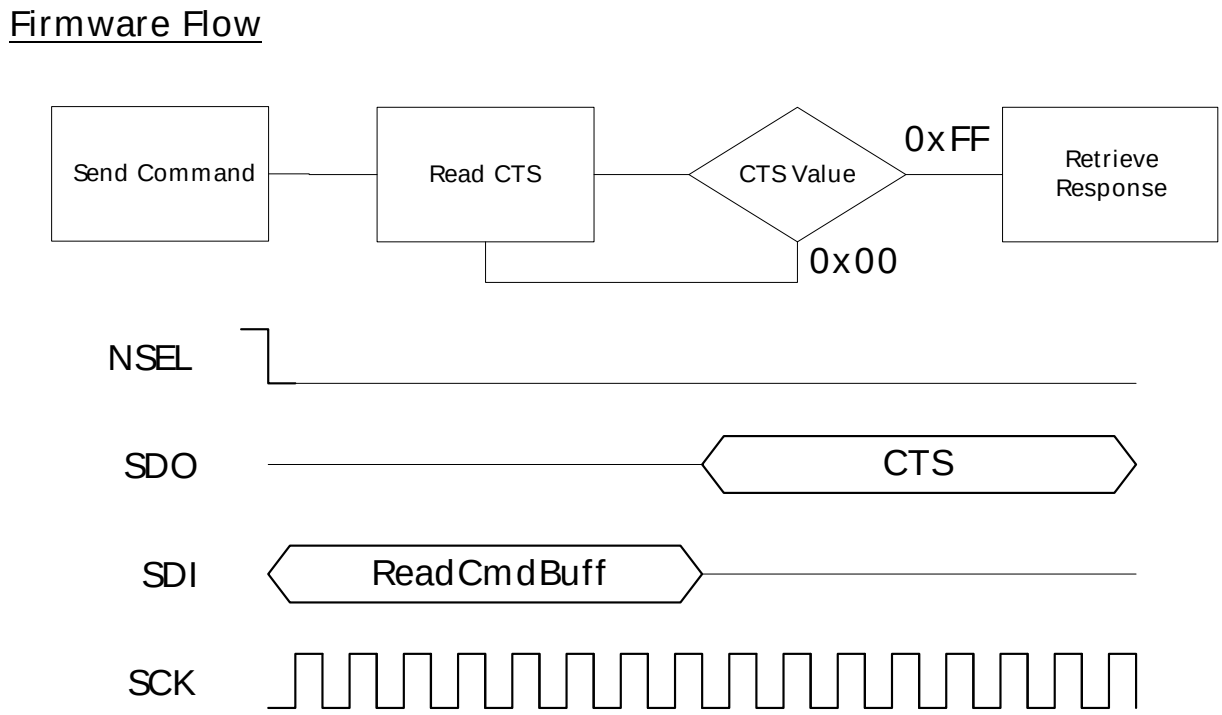
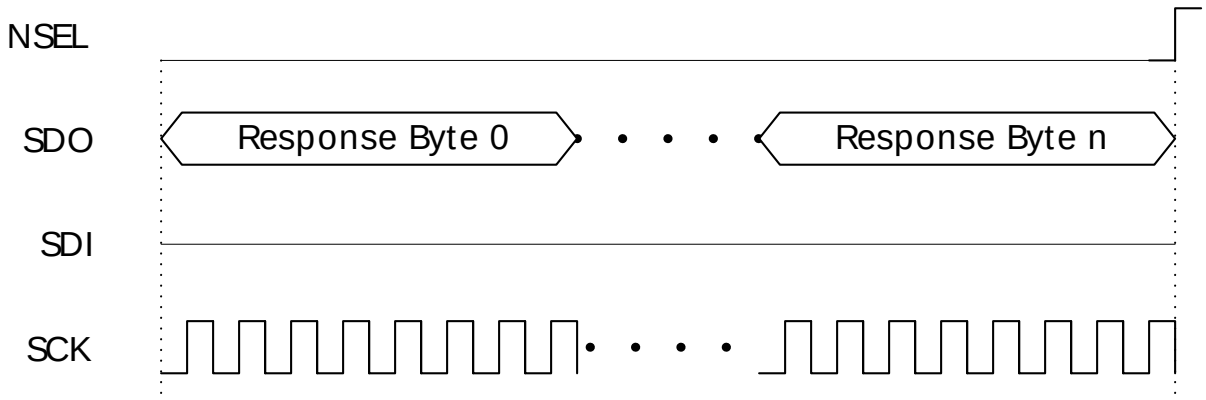


Figure 4.5. SPI Read Command - Clock Out Read Data



The fast response registers are registers that can be read immediately without the requirement to monitor and check CTS. There are four fast response registers that can be programmed for a specific function. The fast response registers can be read through API commands, 0x50 for Fast Response A, 0x51 for Fast Response B, 0x53 for Fast Response C, and 0x57 for Fast Response D. The fast response registers can be configured by the “FRR_CTL_X_MODE” properties.

4.1.3.2 Fast Response Registers

The fast response registers are registers that can be read immediately without the requirement to monitor and check CTS. There are four fast response registers that can be programmed for a specific function. The fast response registers can be read through API commands, 0x50 for Fast Response A, 0x51 for Fast Response B, 0x53 for Fast Response C, and 0x57 for Fast Response D. The fast response registers can be configured by the “FRR_CTL_X_MODE” properties.

4.1.3.3 Operating Modes and Timing

The primary states of the EZRadioPRO are shown in Figure 4.6 (p. 15) . The shutdown state completely shuts down the radio to minimize current consumption. Standby/Sleep, SPI Active, Ready, TX Tune, and RX tune are available to optimize the current consumption and response time to RX/TX for a given application. API commands START_RX, START_TX, and CHANGE_STATE control the operating state with the exception of shutdown which is controlled by SDN, pin 1. Table 4.2 (p. 16) shows each of the operating modes with the time required to reach either RX or TX mode as well as the current consumption of each mode. The times in Table 4.1 (p. 12) are measured from the rising edge of nSEL until the chip is in the desired state. Note that these times are indicative of state transition timing but are not guaranteed and should only be used as a reference data point. An automatic sequencer will put the chip into RX or TX from any state. It is not necessary to manually step through the states. To simplify the diagram it is not shown but any of the lower power states can be returned to automatically after RX or TX.

Figure 4.6. State Machine Diagram

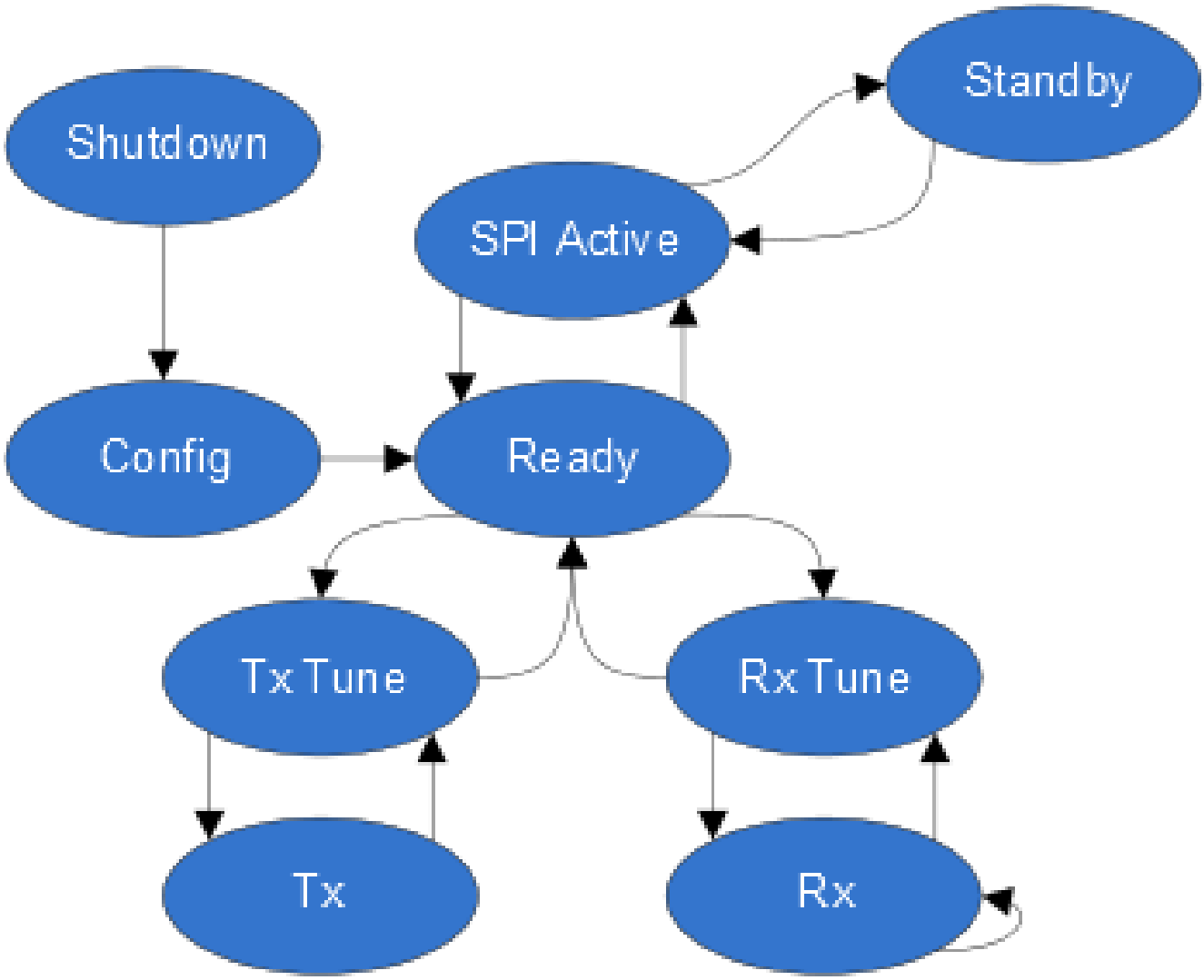


Table 4.2. Operating State Response Time and Current Consumption

State/Mode	Response Time to		Current in State /Mode
	TX	RX	
Shutdown State	15 ms	15 ms	30 nA
Standby State	440 µs	440 µs	40 nA
Sleep State	440 µs	440 µs	740 nA
SPI Active State	340 µs	340 µs	1.35 mA
Ready State	100 µs	100 µs	1.8 mA
TX Tune State	58 µs		7.8 mA
RX Tune State		60 µs	7.6 mA
TX State		100 µs	18 mA @ +10 dBm
RX State	100 µs	75 µs	10.9 or 13.7 mA

Note

TX→RX and RX→TX state transition timing can be reduced to 70 µs if using Zero-IF mode.

Figure 4.7 (p. 16) shows the POR timing and voltage requirements. The power consumption (battery life) depends on the duty cycle of the application or how often the part is in either Rx or Tx state. In most applications the utilization of the standby state will be most advantageous for battery life but for very low duty cycle applications shutdown will have an advantage. For the fastest timing the next state can be selected in the START_RX or START_TX API commands to minimize SPI transactions and internal MCU processing.

4.1.3.3.1 Power on Reset (POR)

A Power On Reset (POR) sequence is used to boot the device up from a fully off or shutdown state. To execute this process, VDD must ramp within 1ms and must remain applied to the device for at least 10 ms. If VDD is removed, then it must stay below 0.15 V for at least 10 ms before being applied again. See Figure 4.7 (p. 16) and Table 4.3 (p. 17) for details.

Figure 4.7. POR Timing Diagram

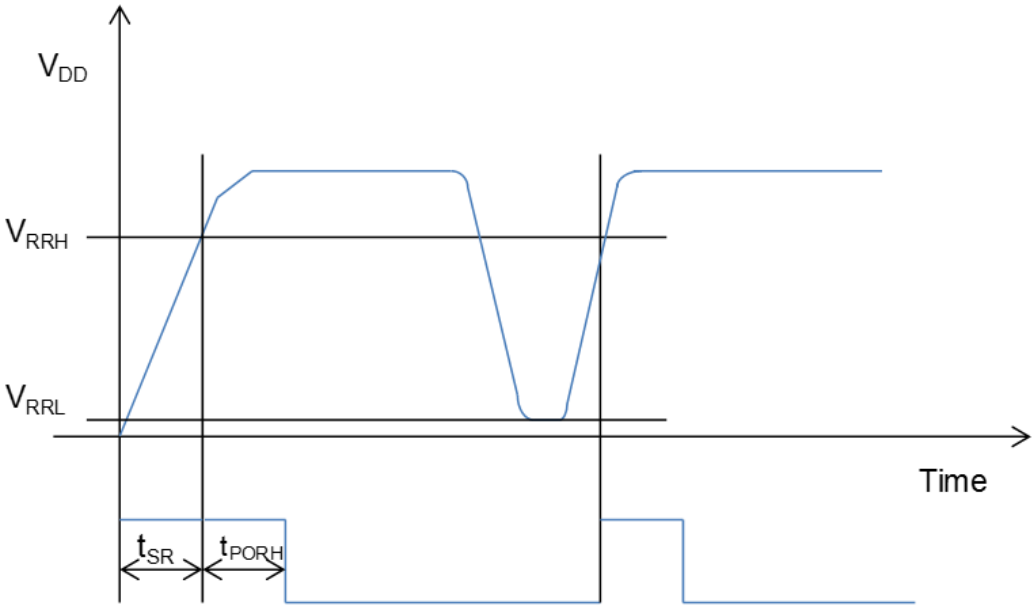


Table 4.3. POR Timing

Variable	Description	Min	Typ	Max	Units
tPORH	High time for VDD to fully settle POR circuit	10			ms
tPORL	Low time for VDD to enable POR	10			ms
VRRH	Voltage for successful POR	90% x Vdd			V
VRRL	Starting Voltage for successful POR	0		150	mV
tSR	Slew rate of VDD for successful POR			1	ms

4.1.3.3.2 Shutdown State

The shutdown state is the lowest current consumption state of the device with nominally less than 30 nA of current consumption. The shutdown state may be entered by driving the SDN pin (Pin 1) high. The SDN pin should be held low in all states except the shutdown state. In the shutdown state, the contents of the registers are lost and there is no SPI access. When coming out of the shutdown state a power on reset (POR) will be initiated along with the internal calibrations. After the POR the POWER_UP command is required to initialize the radio. The SDN pin needs to be held high for at least 10us before driving low again so that internal capacitors can discharge. Not holding the SDN high for this period of time may cause the POR to be missed and the device to boot up incorrectly. If POR timing and voltage requirements cannot be met, it is highly recommended that SDN be controlled using the host processor rather than tying it to GND on the board.

4.1.3.3.3 Standby State

Standby state has the lowest current consumption with the exception of shutdown but has much faster response time to RX or TX mode. In most cases standby should be used as the low power state. In this state the register values are maintained with all other blocks disabled. The SPI is accessible during this mode but any SPI event, including FIFO R/W, will enable an internal boot oscillator and automatically move the part to SPI active state. After an SPI event the host will need to re-command the device back to standby through the “Change State” API command to achieve the 40 nA current consumption. If an interrupt has occurred (i.e., the nIRQ pin = 0) the interrupt registers must be read to achieve the minimum current consumption of this mode.

4.1.3.3.4 Sleep State

Sleep state is the same as standby state but the wake-up-timer and a 32 kHz clock source are enabled. The source of the 32 kHz clock can either be an internal 32 kHz RC oscillator which is periodically calibrated or a 32 kHz oscillator using an external XTAL.The SPI is accessible during this mode but an SPI event will enable an internal boot oscillator and automatically move the part to SPI active mode. After an SPI event the host will need to re-command the device back to sleep. If an interrupt has occurred (i.e., the nIRQ pin = 0) the interrupt registers must be read to achieve the minimum current consumption of this mode.

4.1.3.3.5 SPI Active State

In SPI active state the SPI and a boot up oscillator are enabled. After SPI transactions during either standby or sleep the device will not automatically return to these states. A “Change State” API command will be required to return to either the standby or sleep modes.

4.1.3.3.6 Ready State

Ready state is designed to give a fast transition time to TX or RX state with reasonable current consumption. In this mode the Crystal oscillator remains enabled reducing the time required to switch to TX or RX mode by eliminating the crystal start-up time.

4.1.3.3.7 TX State

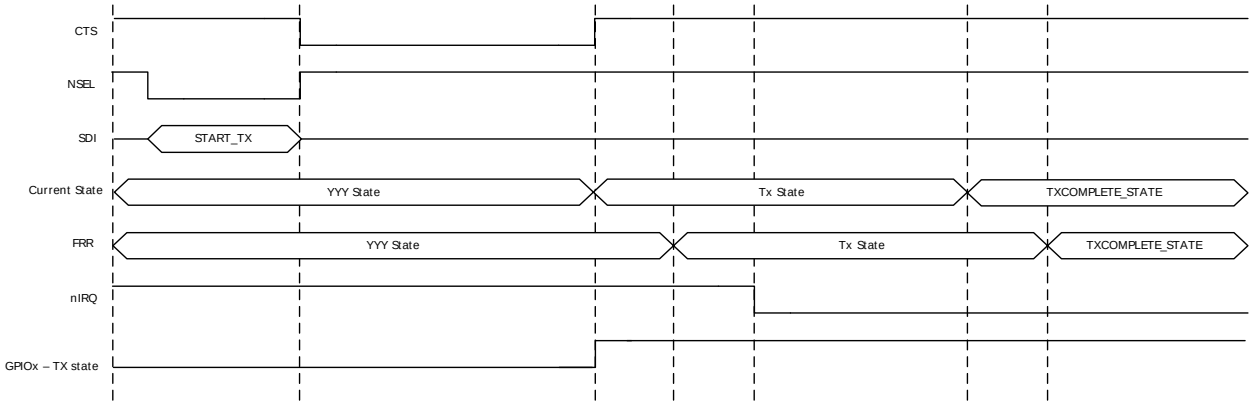
The TX state may be entered from any of the state with the “Start TX” or “Change State” API commands. A built-in sequencer takes care of all the actions required to transition between states from enabling the crystal oscillator to ramping up the PA. The following sequence of events will occur automatically when going from standby to TX state.

- Enable internal LDOs.
- Start up crystal oscillator and wait until ready (controlled by an internal timer).
- Enable PLL.
- Calibrate VCO/PLL.
- Wait until PLL settles to required transmit frequency (controlled by an internal timer).
- Activate power amplifier and wait until power ramping is completed (controlled by an internal timer).
- Transmit packet.

Steps in this sequence may be eliminated depending on which state the chip is configured to prior to commanding to TX. By default, the VCO and PLL are calibrated every time the PLL is enabled. When the START_TX API command is utilized the next state may be defined to ensure optimal timing and turnaround.

Figure 4.8 (p. 18) shows an example of the commands and timing for the START_TX command. CTS will go high as soon as the sequencer puts the part into TX state. As the sequencer is stepping through the events listed above, CTS will be low and no new commands or property changes are allowed. If the Fast Response (FRR) or nIRQ is used to monitor the current state there will be slight delay caused by the internal hardware from when the event actually occurs to when the transition occurs on the FRR or nIRQ. The time from entering TX state to when the FRR will update is 5 µs and the time to when the nIRQ will transition is 13 µs. If a GPIO is programmed for TX state or used as control for a transmit/ receive switch (TR switch) there is no delay.

Figure 4.8. Start_TX Commands and Timing



4.1.3.3.8 RX State

The RX state may be entered from any of the other states by using the “Start RX” or “Change State” API command. A built-in sequencer takes care of all the actions required to transition between states. The following sequence of events will occur automatically to get the chip into RX mode when going from standby to RX state:

- Enable the digital LDO and the analog LDOs.
- Start up crystal oscillator and wait until ready (controlled by an internal timer).
- Enable PLL.

- Calibrate VCO.
- Wait until PLL settles to required receive frequency (controlled by an internal timer).
- Enable receiver circuits: LNA, mixers, and ADC.
- Enable receive mode in the digital modem.

Depending on the configuration of the radio, all or some of the following functions will be performed automatically by the digital modem: AGC, AFC (optional), update status registers, bit synchronization, packet handling (optional) including sync word, header check, and CRC. Similar to the TX state, the next state after RX may be defined in the “Start RX” API command. The START_RX commands and timing will be equivalent to the timing shown in Figure 4.8 (p. 18) .

4.1.3.4 Application Programming Interface (API)

An application programming interface (API), which the host MCU will communicate with, is embedded inside the device. The API is divided into two sections, commands and properties. The commands are used to control the chip and retrieve its status. The properties are general configurations which will change infrequently. The API descriptions can be found on the Silicon Labs website.

4.1.3.5 Interrupts

The EZRadioPRO is capable of generating an interrupt signal when certain events occur. The chip notifies the microcontroller that an interrupt event has occurred by setting the nIRQ output pin LOW = 0. This interrupt signal will be generated when any one (or more) of the interrupt events (corresponding to the Interrupt Status bits) occur. The nIRQ pin will remain low until the microcontroller reads the Interrupt Status Registers. The nIRQ output signal will then be reset until the next change in status is detected.

The interrupts sources are grouped into three groups: packet handler, chip status, and modem. The individual interrupts in these groups can be enabled/disabled in the interrupt property registers. An interrupt must be enabled for it to trigger an event on the nIRQ pin. The interrupt group must be enabled as well as the individual interrupts in API properties described in the API documentation.

Once an interrupt event occurs and the nIRQ pin is low there are two ways to read and clear the interrupts. All of the interrupts may be read and cleared in the “GET_INT_STATUS” API command. By default all interrupts will be cleared once read. If only specific interrupts want to be read in the fastest possible method the individual interrupt groups (Packet Handler, Chip Status, Modem) may be read and cleared by the “GET_MODEM_STATUS”, “GET_PH_STATUS” (packet handler), and “GET_CHIP_STATUS” API commands.

The instantaneous status of a specific function maybe read if the specific interrupt is enabled or disabled. The status results are provided after the interrupts and can be read with the same commands as the interrupts. The status bits will give the current state of the function whether the interrupt is enabled or not.

The fast response registers can also give information about the interrupt groups but reading the fast response registers will not clear the interrupt and reset the nIRQ pin.

4.1.3.6 GPIO

Four general purpose IO pins are available to utilize in the application. The GPIO are configured by the GPIO_PIN_CFG command in address 13h. For a complete list of the GPIO options please see the API guide. GPIO pins 0 and 1 should be used for active signals such as data or clock. GPIO pins 2 and 3 have more susceptibility to generating spurious in the synthesizer than pins 0 and 1. The drive strength of the GPIOs can be adjusted with the GEN_CONFIG parameter in the GPIO_PIN_CFG command. By default the drive strength is set to minimum. The default configuration for the GPIOs and the state during SDN is shown below in Table 4.4 (p. 20) .The state of the IO during shutdown is also shown in Table 4.4 (p. 20) .

Table 4.4. Energy Mode Description

Pin	SDN State	POR Default
GPIO0	0	POR
GPIO1	0	CTS
GPIO2	0	POR
GPIO3	0	POR
nIRQ	resistive VDD pull-up	nIRQ
SDO	resistive VDD pull-up	SDO
SDI	High Z	SDI
SCLK	High Z	SCLK
NSEL	High Z	NSEL

4.1.4 Modulation and Hardware Configuration Options

The EZRadioPRO supports different modulation options and can be used in various configurations to tailor the device to any specific application or legacy system for drop in replacement. The modulation and configuration options are set in API property, MODEM_MOD_TYPE. Refer to the API documentation for details on modem related properties.

4.1.4.1 Modulation Types

The EZRadioPRO supports five different modulation options: Gaussian frequency shift keying (GFSK), frequency-shift keying (FSK), four-level GFSK (4GFSK), four-level FSK (4FSK), and on-off keying (OOK). Minimum shift keying (MSK) can also be created by using GFSK with the appropriate modulation index (h = 0.5). GFSK is the recommended modulation type as it provides the best performance and cleanest modulation spectrum. The modulation type is set by the “MOD_TYPE[2:0]” field in the “MODEM_MOD_TYPE” API property. A continuous-wave (CW) carrier may also be selected for RF evaluation purposes. The modulation source may also be selected to be a pseudo-random source for evaluation purposes.

4.1.4.2 Hardware Configuration Options

There are different receive demodulator options to optimize the performance and mutually-exclusive options for how the RX/TX data is transferred from the host MCU to the RF device.

4.1.4.2.1 Receive Demodulator Options

There are multiple demodulators integrated into the device to optimize the performance for different applications, modulation formats, and packet structures. The calculator built into Simplicity Studio will choose the optimal demodulator based on the input criteria.

4.1.4.2.2 Synchronous Demodulator

The synchronous demodulator's internal frequency error estimator acquires the frequency error based on a 101010 preamble structure. The bit clock recovery circuit locks to the incoming data stream within four transactions of a “10” or “01” bit stream. The synchronous demodulator gives optimal performance for 2- or 4-level (G)FSK modulation that has a modulation index less than 2.

4.1.4.2.3 Asynchronous Demodulator

The asynchronous demodulator should be used for OOK modulation and for (G)FSK modulation under one or more of the following conditions:

- Modulation index ≥ 2
- Non-standard preamble (not 1010101... pattern)

When the modulation index exceeds 2, the asynchronous demodulator has better sensitivity compared to the synchronous demodulator. An internal deglitch circuit provides a glitch-free data output and a data clock signal to simplify the interface to the host. There is no requirement to perform deglitching in the host MCU. The asynchronous demodulator will typically be utilized for legacy systems and will have many performance benefits over devices used in legacy designs. There is no requirement to perform deglitching on the data in the host MCU. Glitch-free data is output from EZRadioPRO devices, and a sample clock for the asynchronous data can also be supplied to the host MCU; so, oversampling or bit clock recovery is not required by the host MCU. There are multiple detector options in the asynchronous demodulator block, which will be selected based upon the options entered into the SS calculator. The asynchronous demodulator's internal frequency error estimator is able to acquire the frequency error based on any preamble structure.

4.1.4.2.4 RX/TX Data Interface With MCU

There are two different options for transferring the data from the RF device to the host MCU. FIFO mode uses the SPI interface to transfer the data, while direct mode transfers the data in real time over a GPIO pin.

4.1.4.2.4.1 FIFO Mode

In FIFO mode, the transmit and receive data is stored in integrated FIFO register memory. The TX FIFO is accessed by writing command 66h followed directly by the data/clk that the host wants to write into the TX FIFO. The RX FIFO is accessed by writing command 77h followed by the number of clock cycles of data the host would like to read out of the RX FIFO. The RX data will be clocked out onto the SDO pin.

In TX FIFO mode, the data bytes stored in FIFO memory are “packaged” together with other fields and bytes of information to construct the final transmit packet structure. These other potential fields include the Preamble, Sync word, and CRC checksum. In TX mode, the packet structure may be highly customized by enabling or disabling individual fields; for example, it is possible to disable both the Preamble and Sync Word fields and to load the entire packet structure into FIFO memory. For further information on the configuration of the FIFOs for a specific application or packet size, see Section 4.1.6 (p. 33) . In RX mode, the Packet Handler must be enabled to allow storage of received data bytes into RX FIFO memory. The Packet Handler is required to detect the Sync Word, and proper detection of the Sync Word is required to determine the start of the Payload. All bytes after the Sync Word are stored in RX FIFO memory except the CRC checksum and (optionally) the variable packet length byte(s). When the FIFO is being used in RX mode, all of the received data may still be observed directly (in realtime) by properly programming a GPIO pin as the RXDATA output pin; this can be quite useful during application development. When in FIFO mode, the chip will automatically exit the TX or RX State when either the PACKET_SENT or PACKET_RX interrupt occurs. The chip will return to the state programmed in the argument of the “START TX” or “START RX” API command, TXCOMPLETE_STATE[3:0] or RXVALID_STATE[3:0]. For example, the chip may be placed into READY mode after a TX packet by sending the “START TX” command and by writing 30h to the TXCOMPLETE_STATE[3:0] argument. The chip will transmit all of the contents of the FIFO, and the PACKET_SENT interrupt will occur. When this event occurs, the chip will return to the READY state as defined by TXCOMPLETE_STATE[3:0] = 30h.

4.1.4.2.4.2 FIFO Direct Mode (Infinite Receive)

In some applications, there is a need to receive extremely long packets (greater than 40 kB) while relying on preamble and sync word detection from the on-chip packet handler. In these cases, the packet length is unknown, and the device will load the bits after the sync word into the RX FIFO forever. Other features, such as Data Whitening, CRC, Manchester, etc., are supported in this mode, but CRC calculation is not because the end of packet is unknown to the device. The RX data and clock are also available on GPIO

pins. The host MCU will need to reset the packet handler by issuing a START_RX to begin searching for a new packet.

4.1.4.2.4.3 Automatic TX Packet Repeat

In TX mode, there is an option to send the FIFO contents repeatedly with a user-defined number of times to repeat. This is limited to the FIFO size, and the entire contents of the packet including preamble and sync word need to be loaded into the TX FIFO. This is selectable via the START_TX API, and packets will be sent without any gaps between them.

4.1.4.2.4.4 Direct Mode

For legacy systems that perform packet handling within the host MCU or other baseband chip, it may not be desirable to use the FIFO. For this scenario, a Direct mode is provided, which bypasses the FIFOs entirely. In TX Direct mode, the TX modulation data is applied to an input pin of the chip and processed in “real time” (i.e., not stored in a register for transmission at a later time). Any of the GPIOs may be configured for use as the TX Data input function. Furthermore, an additional pin may be required for a TX Clock output function if GFSK modulation is desired (only the TX Data input pin is required for FSK or OOK). To achieve direct mode, the desired GPIO pin must be configured as a digital input by setting the GPIO_PIN_CFG API command = enumeration 0x04 in addition to setting the MODEM_MOD_TYPE API property to source the TXDATA stream from that same GPIO pin. For GFSK, “TX_DIRECT_MODE_TYPE” must be set to synchronous. For 2FSK or OOK, the type can be set to asynchronous or synchronous. The MOD_SOURCE[1:0] field within the MODEM_MOD_TYPE property should be set = 0x01h for all Direct mode configurations. In RX Direct mode, the RX Data and RX Clock can be programmed for direct (real-time) output to GPIO pins. The microcontroller may then process the RX data without using the FIFO or packet handler functions of the RFIC.

4.1.4.3 Preamble Length

4.1.4.3.1 Digital Signal Arrival Detector (DSA)

Traditional preamble detection requires 20 bits to detect preamble. This device introduces a new approach to signal detection that can detect a preamble pattern in as little as one byte. If AFC is enabled, a preamble length of two bytes is sufficient to reliably detect signal arrival and settle a one-shot AFC. The impact of this is significant for low-power solutions as it reduces the amount of time the receiver has to stay active to detect the preamble. This feature is used with Preamble Sense Mode (Section 4.1.8.6 (p. 36)) and the latest Wireless M-Bus N modes as well as with features, such as frequency hopping, which may use signal arrival as a condition to hop. The traditional preamble detector is also available to maintain backward compatibility. Note that the DSA is using the RSSI jump detector. When used for collision detection, the RSSI jump detector may need to be reconfigured after preamble detection. Refer to the API documentation for details on how to configure the device to use the signal arrival detector.

4.1.4.3.2 Traditional Preamble Detection

Optimal performance of the chip is obtained by qualifying reception of a valid Preamble pattern prior to continuing with reception of the remainder of the packet (e.g., Sync Word and Payload). Reception of the Preamble is considered valid when a minimum number of consecutive bits of 101010... pattern have been received; the required threshold for preamble detection is specified by the RX_THRESH[6:0] field in the PREAMBLE_CONFIG_STD_1 property. The appropriate value of the detection threshold depends upon the system application and typically trades off speed of acquisition against the probability of false detection. If the detection threshold is set too low, the chip may readily detect the short pattern within noise; the chip then proceeds to attempt to detect the remainder of the non-existent packet, with the result that the arrival of an actual valid packet may be missed. If the detection threshold is set too high, the required number of transmitted Preamble bits must be increased accordingly, leading to longer packet lengths and shorter battery life. A preamble detection threshold value of 20 bits is suitable for most applications. The total length of the transmitted Preamble field must be at least equal to the receive preamble detection threshold, plus an additional number of bits to allow for acquisition of bit timing and

settling of the AFC algorithm. The recommended preamble detection thresholds and preamble lengths for a variety of operational modes are listed in Table 4.5 (p. 23) .

Configuration of the preamble detection threshold in the RX_THRESH[6:0] field is only required for reception of a standard Preamble pattern (i.e., 101010... pattern). Reception of a repetitive but non-standard Preamble pattern is also supported in the chip but is configured through the PREAMBLE_CONFIG_NSTD and PREAMBLE_PATTERN properties.

Table 4.5. Recommended Preamble Length

Mode	AFC	Antenna Diversity	Preamble Type	Recommended Preamble Length	Recommended Preamble Detection Threshold
(G)FSK	Disabled	Disabled	Standard	4 Bytes	20 bits
(G)FSK	Enabled	Disabled	Standard	5 Bytes	20 bits
(G)FSK	Disabled	Disabled	Non-standard	2 Bytes	0 bits
(G)FSK	Enabled		Non-standard	Not Supported	
(G)FSK	Disabled	Enabled	Standard	7 Bytes	24 bits
(G)FSK	Enabled	Enabled	Standard	8 Bytes	24 bits
4(G)FSK	Disabled	Disabled	Standard	40 symbols	16 symbols
4(G)FSK	Enabled	Disabled	Standard	48 symbols	16 symbols
4(G)FSK			Non-standard	Not Supported	
OOK	Disabled	Disabled	Standard	4 Bytes	20 bits
OOK	Disabled	Disabled	Non-standard	2 Bytes	0 bits
OOK	Enabled			Not Supported	

Note

- The recommended preamble length and preamble detection thresholds listed above are to achieve 0% PER. They may be shortened when occasional packet errors are tolerable.
- All recommended preamble lengths and detection thresholds include AGC and BCR settling times.
- “Standard” preamble type should be set for an alternating data sequence at the max data rate (…10101010…).
- “Non-standard” preamble type can be set for any preamble type including …10101010...
- When preamble detection threshold = 0, sync word needs to be 3 Bytes to avoid false syncs. When only a 2 Byte sync word is available the sync word detection can be extended by including the last preamble Byte into the RX sync word setting.

4.1.5 Internal Functional Blocks

The following sections provide an overview to the key internal blocks and features.

4.1.5.1 RX Chain

The internal low-noise amplifier (LNA) is designed to be a wide-band LNA that can be matched with three or four external discrete components to cover any common range of frequencies in the sub-GHz band. The LNA has extremely low noise to suppress the noise of the following stages and achieve optimal sensitivity; so, no external gain or front-end modules are necessary. The LNA has gain control, which is controlled by the internal automatic gain control (AGC) algorithm. The LNA is followed by an I-Q mixer, filter, programmable gain amplifier (PGA), and ADC. The I-Q mixers downconvert the signal to an intermediate frequency. The PGA then boosts the gain to be within dynamic range of the ADC. The ADC

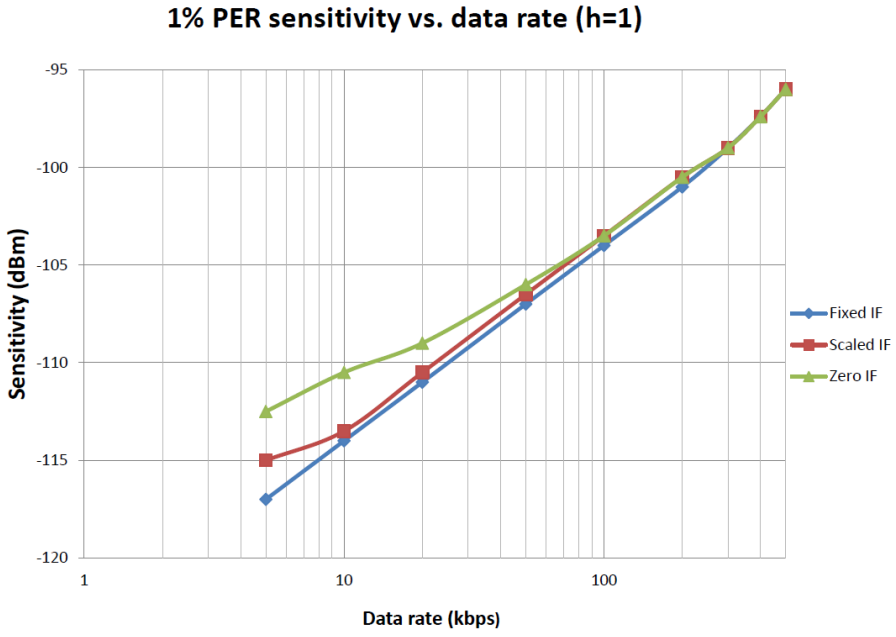
rejects out-of-band blockers and converts the signal to the digital domain where filtering, demodulation, and processing is performed. Peak detectors are integrated at the output of the LNA and PGA for use in the AGC algorithm.

The RX and TX pins may be directly tied externally for output powers less than +17 dBm in the higher-frequency bands and can support +20 dBm in the lower bands, such as 169MHz. This reduces BOM cost by saving the expense of a switch for single antenna solutions. See the direct-tie reference designs on the Silicon Labs web site for more details.

4.1.5.1.1 RX Chain Architecture

It is possible to operate the RX chain in different architecture configurations: fixed-IF, zero-IF, and scaled-IF. There are trade-offs between the architectures in terms of sensitivity, selectivity, and image rejection. Fixed-IF is the default configuration and is recommended for most applications. With 35 dB native image rejection and autonomous image calibration to achieve 55 dB, the fixed-IF solution gives the best performance for most applications. Fixed-IF obtains the best sensitivity, but it has the effect of degraded selectivity at the image frequency. An autonomous image rejection calibration is included in EZRadioPRO devices and described in more detail in Section 4.1.5.2.3 (p. 25). For scaled-IF and zero-IF, the sensitivity is degraded for data rates less than 100 kbps or bandwidths less than 200 kHz. The reduction in sensitivity is caused by increased flicker noise as dc is approached. The benefit of zero-IF is that there is no image frequency; so, there is no degradation in the selectivity curve, but it has the worst sensitivity. Scaled-IF is a trade-off between fixed-IF and zero-IF. In the scaled-IF architecture, the image frequency is placed or hidden in the adjacent channel where it only slightly degrades the typical adjacent channel selectivity. The scaled-IF approach has better sensitivity than zero-IF but still some degradation in selectivity due to the image. In scaled-IF mode, the image frequency is directly proportional to the channel bandwidth selected. Figure 4.8 (p. 18) demonstrates the trade-off in sensitivity between the different architecture options.

Figure 4.9. RX Architecture vs. Data Rate



4.1.5.2 RX Modem

Using high-performance ADCs allows channel filtering, image rejection, and demodulation to be performed in the digital domain, which allows for flexibility in optimizing the device for particular applications. The digital modem performs the following functions:

- Channel selection filter
- TX modulation

- RX demodulation
- Automatic Gain Control (AGC)
- Preamble detection
- Invalid preamble detection
- Radio signal strength indicator (RSSI)
- Automatic frequency compensation (AFC)
- Image Rejection Calibration
- Packet handling
- Cyclic redundancy check (CRC)

The digital channel filter and demodulator are optimized for ultra-low-power consumption and are highly configurable. Supported modulation types are GFSK, FSK, 4GFSK, 4FSK, GMSK, and OOK. The channel filter can be configured to support bandwidths ranging from 850 kHz down to 1.1 kHz. A large variety of data rates are supported ranging from 100 bps up to 1 Mbps. The configurable preamble detector is used with the synchronous demodulator to improve the reliability of the sync-word detection. Preamble detection can be skipped using only sync detection, which is a valuable feature in some applications. The received signal strength indicator (RSSI) provides a measure of the signal strength received on the tuned channel. The resolution of the RSSI is 0.5 dB. This high-resolution RSSI enables accurate channel power measurements for clear channel assessment (CCA), carrier sense (CS), and listen before talk (LBT) functionality. A comprehensive programmable packet handler is integrated to create a variety of communication topologies ranging from peer-to-peer networks to mesh networks. The extensive programmability of the packet header allows for advanced packet filtering, which, in turn enables a mix of broadcast, group, and point-to-point communication. A wireless communication channel can be corrupted by noise and interference, so it is important to know if the received data is free of errors. A cyclic redundancy check (CRC) is used to detect the presence of erroneous bits in each packet. A CRC is computed and appended at the end of each transmitted packet and verified by the receiver to confirm that no errors have occurred. The packet handler and CRC can significantly reduce the load on the system microcontroller allowing for a simpler and cheaper microcontroller. The digital modem includes the TX modulator, which converts the TX data bits into the corresponding stream of digital modulation values to be summed with the fractional input to the sigma-delta modulator. This modulation approach results in highly accurate resolution of the frequency deviation. A Gaussian filter is implemented to support GFSK and 4GFSK, considerably reducing the energy in adjacent channels. The default bandwidth-time product (BT) is 0.5 for all programmed data rates, but it may be adjusted to other values.

4.1.5.2.1 Automatic Gain Control (AGC)

The AGC algorithm is implemented digitally using an advanced control loop optimized for fast response time. The AGC occurs within a single bit or in less than 2 µs. Peak detectors at the output of the LNA and PGA allow for optimal adjustment of the LNA gain and PGA gain to optimize IM3, selectivity, and sensitivity performance.

4.1.5.2.2 Auto Frequency Correction (AFC)

Frequency mistuning caused by crystal inaccuracies can be compensated for by enabling the digital automatic frequency control (AFC) in receive mode. There are two types of integrated frequency compensation: modem frequency compensation and AFC by adjusting the PLL frequency. With AFC disabled, the modem compensation can correct for frequency offsets up to ±0.25 times the IF bandwidth. When the AFC is enabled, the received signal is centered in the passband of the IF filter, providing optimal sensitivity and selectivity over a wider range of frequency offsets up to ±0.35 times the IF bandwidth. When AFC is enabled, the preamble length needs to be long enough to settle the AFC. As shown in Table 4.5 (p. 23) , an additional byte of preamble is typically required to settle the AFC.

4.1.5.2.3 Image Rejection and Calibration

Since the receiver utilizes a low-IF architecture, the selectivity will be affected by the image frequency. The IF frequency is 468.75 kHz (Fxtal/64), and the image frequency will be at 937.5 kHz (2 x Fxtal/64)

below the RF frequency. The native image rejection of the EZRadioPRO family is 40 dB. Image rejection calibration is available in the EZRadioPRO to improve the image rejection to more than 55 dB. The calibration is initiated with the IRCAL API command. The calibration uses an internal signal source, so no external signal generator is required. The initial calibration takes 250 ms, and periodic re-calibration takes 100 ms. Recalibration should be initiated when the temperature has changed more than 30 °C.

4.1.5.2.4 Received Signal Strength Indicator

The received signal strength indicator (RSSI) is an estimate of the signal strength in the channel to which the receiver is tuned. The RSSI measurement is done after the channel filter, so it is only a measurement of the in-band signal power (desired or undesired). There are two methods for reading the RSSI value and several different options for configuring the returned RSSI value. The fastest method for reading the RSSI is to configure one of the four fast response registers (FRR) to return a latched RSSI value. The latched RSSI value is measured once per packet and is latched at a configurable amount of time after RX mode is entered. The fast response registers can be read in 16 SPI clock cycles with no requirement to wait for CTS. The RSSI value may also be read out of the GET_MODEM_STATUS command. In this command, both the current RSSI and the latched RSSI are available. The current RSSI value represents the signal strength at the instant in time the GET_MODEM_STATUS command is processed and may be read multiple times per packet. Reading the RSSI in the GET_MODEM_STATUS command takes longer than reading the RSSI out of the fast response register. After the initial command, it takes 33 µs for CTS to be set and then the four or five bytes of SPI clock cycles to read out the respective current or latched RSSI values.

The RSSI configuration options are set in the MODEM_RSSI_CONTROL API property. The latched RSSI value may be latched and stored based on the following events: preamble detection, sync detection, or a configurable number of bit times measured after the start of RX mode (minimum of 4 bit times). The requirement for a minimum of four bit times is determined by the processing delay and settling through the modem and digital channel filter. In MODEM_RSSI_CONTROL, the RSSI may be defined to update every bit period or to be averaged and updated every four bit periods. If RSSI averaging over four bits is enabled, the latched RSSI value will be delayed to a minimum of seven bits after the start of RX mode to allow for the averaging. The latched RSSI values are cleared when entering RX mode so they may be read after the packet is received or after dropping back to standby mode. If the RSSI value has been cleared by the start of RX but not yet latched, a value of 0 will be returned if it is attempted to be read.

The RSSI value read by the API may be translated into dBm by the following linear equation: RF_Input_Level_dBm = (RSSI_value / 2) - MODEM_RSSI_COMP - 70

The MODEM_RSSI_COMP property provides for fine adjustment of the relationship between the actual RF input level (in dBm) and the returned RSSI value. That is, adjustment of this property allows the user to shift the RSSI vs RF Input Power curve up and down. This may be desirable to compensate for differences in front-end insertion loss between multiple designs (e.g., due to the presence of a SAW preselection filter, or an RF switch). A value of MODEM_RSSI_COMP = 0x40 = 64d is appropriate for most applications.

Clear channel assessment (CCA) or RSSI threshold detection is also available. An RSSI threshold may be set in the MODEM_RSSI_THRESH API property. If the Current RSSI value is above this threshold, an interrupt or GPIO may notify the host. Both the latched version and asynchronous version of this threshold are available on any of the GPIOs. Automatic fast hopping based on RSSI is available. See Section 4.1.5.3.1.2 (p. 28) .

4.1.5.2.5 RSSI Jump Indicator (Collision Detection)

The chip is capable of detecting a jump in RSSI in either direction (i.e., either a signal increase or a signal decrease). Both polarities of jump detection may be enabled simultaneously, resulting in detection of a Jump-Up or Jump-Down event. This may be used to detect whether a secondary interfering signal (desired or undesired) has “collided” with reception of the current packet. An interrupt flag or GPIO pin

may be configured to notify the host MCU of the Jump event. The change in RSSI level required to trigger the Jump event is programmable through the MODEM_RSSI_JUMP_THRESH API property.

The chip may be configured to reset the RX state machine upon detection of an RSSI Jump, and thus to automatically begin reacquisition of the packet. The chip may also be configured to generate an interrupt.

This functionality is intended to detect an abrupt change in RSSI level and to not respond to a slow, gradual change in RSSI level. This is accomplished by comparing the difference in RSSI level over a programmable time period. In this fashion, the chip effectively evaluates the slope of the change in RSSI level.

The arrival of a desired packet (i.e., the transition from receiving noise to receiving a valid signal) will likely be detected as an RSSI Jump event. For this reason, it is recommended to enable this feature in mid-packet (i.e., after signal qualification, such as PREAMBLE_VALID.) Refer to the API documentation for configuration options.

4.1.5.3 Synthesizer

An integrated Sigma Delta (ΣΔ) Fractional-N PLL synthesizer capable of operating over the bands from 142-175, 283-350, 350-525, and 850-1050 MHz. Using a ΣΔ synthesizer has many advantages; it provides flexibility in choosing data rate, deviation, channel frequency, and channel spacing. The transmit modulation is applied directly to the loop in the digital domain through the fractional divider, which results in very precise accuracy and control over the transmit deviation. The frequency resolution in the 850-1050 MHz band is 28.6 Hz with finer resolution in the other bands. The nominal reference frequency to the PLL is 30 MHz, but any XTAL frequency from 25 to 32 MHz may be used. The modem configuration calculator in SS will automatically account for the XTAL frequency being used. The PLL utilizes a differential LC VCO with integrated on-chip inductors. The output of the VCO is followed by a configurable divider, which will divide the signal down to the desired output frequency band.

4.1.5.3.1 Synthesizer Frequency Control

The frequency is set by changing the integer and fractional settings to the synthesizer. The SS calculator will automatically provide these settings, but the synthesizer equation is shown below for convenience. The APIs for setting the frequency are FREQ_CONTROL_INTE, FREQ_CONTROL_FRAC2, FREQ_CONTROL_FRAC1, and FREQ_CONTROL_FRAC0.

RF Channel Equation

$$(fc\_inte+fc\_frac/2^{19})*(2*freq\_xo/outdiv) \quad (4.1)$$

Note

The $fc\_frac/2^{19}$ value in the above formula has to be a number between 1 and 2.

Table 4.6. Output Divider (Outdiv) Values for the EZRadioPRO

Outdiv	Lower (MHz)	Upper (MHz)
24	142	175
12	284	350
10	350	420
8	420	525
4	850	1050

4.1.5.3.1.1 EZ Frequency Programming

In applications that utilize multiple frequencies or channels, it may not be desirable to write four API registers each time a frequency change is required. EZ frequency programming is provided so that

only a single register write (channel number) is required to change frequency. A base frequency is first set by first programming the integer and fractional components of the synthesizer. This base frequency will correspond to channel 0. Next, a channel step size is programmed into the FREQ_CONTROL_CHANNEL_STEP_SIZE_1 and FREQ_CONTROL_CHANNEL_STEP_SIZE_0 API registers. The resulting frequency will be:

RF Frequency Equation

$$\text{RF Frequency} = \text{Base Frequency} + (\text{Channel} * \text{Stepsize}) \quad (4.2)$$

The second argument of the START_RX or START_TX is CHANNEL, which sets the channel number for EZ frequency programming. For example, if the channel step size is set to 1 MHz, the base frequency is set to 900 MHz with the FREQ_CONTROL_INTE and FREQ_CONTROL_FRAC API properties, and a CHANNEL number of 5 is programmed during the START_TX command, the resulting frequency will be 905 MHz. If no CHANNEL argument is written as part of the START_RX/TX command, it will default to the previously-programmed value. The initial value of CHANNEL is 0; so, if no CHANNEL value is written, it will result in the programmed base frequency.

4.1.5.3.1.2 Automatic RX Hopping and Hop Table

The transceiver supports an automatic RX hopping feature that can be fully configured through the API. This functionality is useful in applications where it is desired to look for packets but to hop to the next channel if a packet is not found. The sequence of channel numbers that are visited are specified by entries in a hop table. If this feature is enabled, the device will automatically start hopping through the channels listed in the hop table as soon as the chip enters RX mode.

The transceiver supports an automatic RX hopping feature that can be fully configured through the API. This functionality is useful in applications where it is desired to look for packets but to hop to the next channel if a packet is not found. The sequence of channel numbers that are visited are specified by entries in a hop table. If this feature is enabled, the device will automatically start hopping through the channels listed in the hop table as soon as the chip enters RX mode.

The hop table can hold up to 64 entries and is maintained in firmware inside the RFIC. Each entry is a channel number, allowing construction of a frequency plan of up to 64 channels. The number of entries in the table is set by RX HOP TABLE_SIZE API. The specified channels correspond to the EZ frequency programming method for programming the frequency. The receiver starts at the base channel and hops in sequence from the top of the hop table to the bottom. The table will wrap around to the base channel once it reaches the end of the table. An entry of 0xFF in the table indicates that the entry should be skipped. The device will hop to the next entry in the table that contains a non-0xFF value.

There are three conditions that can be used to determine whether to continue hopping or to stay on a particular channel. These conditions are as follows:

- RSSI threshold
- Preamble timeout (invalid preamble pattern)
- Sync word timeout (invalid or no sync word detected after preamble)

These conditions can be used individually, or they can be enabled all together by configuring the RX_HOP_CONTROL API. However, the firmware will make a decision on whether or not to hop based on the first condition that is met.

The RSSI that is monitored is the current RSSI value. This is compared to the threshold value set in the MODEM_RSSI_THRESH API property, and, if it is above the threshold value, it will stay on the channel. If the RSSI is below the threshold, it will continue hopping. There is no averaging of RSSI done during the automatic hopping from channel to channel. Since the preamble timeout and the sync word timeout are features that require packet handling, the RSSI threshold is the only condition that can be used if the user is in “direct” or “RAW” mode where packet handling features are not used.

The RSSI threshold value may be converted to an approximate equivalent RF input power level through the equation shown in Section 4.1.5.2.4 (p. 26). However, performance should be verified on the bench to optimize the threshold setting for a given application.

The time spent in receive mode will be determined by the configuration of the hop conditions. Manual RX hopping will have the fastest turn-around time but will require more overhead and management by the host MCU.

The following are example steps for using Auto Hop:

- Set the base frequency (inte + frac) and channel step size.
- Define the number of entries in the hop table (RX_HOP_TABLE_SIZE).
- Write the channels to the hop table (RX_HOP_TABLE_ENTRY_n)
- Configure the hop condition and enable auto hopping- RSSI, preamble, or sync (RX_HOP_CONTROL).
- Set preamble and sync parameters if enabled.
- Program the RSSI threshold property in the modem using “MODEM_RSSI_THRESH”.
- Set the preamble threshold using “PREAMBLE_CONFIG_STD_1”.
- Program the preamble timeout property using “PREAMBLE_CONFIG_STD_2”.
- Set the sync detection parameters if enabled.
- If needed, use “GPIO_PIN_CFG” to configure a GPIO to toggle on hop and hop table wrap.
- Use the “START_RX” API with channel number set to the first valid entry in the hop table (i.e., the first non 0xFF entry).
- Device should now be in auto hop mode.

4.1.5.3.1.3 Manual RX Hopping

The RX_HOP command provides the fastest method for hopping from RX to RX but it requires more overhead and management by the host MCU. The timing is faster with this method than Start_RX or RX hopping because one of the calculations required for the synthesizer calibrations is offloaded to the host and must be calculated/stored by the host, VCO_CNT0. For VCO_CNT values, download the EZRadioPRO RX_HOP PLL calculator spreadsheet from the EZRadioPRO product website.

4.1.5.4 Transmitter (TX)

The EZRadioPRO contains an integrated +20 dBm transmitter or power amplifier that is capable of transmitting from -20 to +20 dBm. The resolution of the programmable steps in output power is less than 0.25 dB when operated within 6 dB of the maximum power setting; the resolution of the steps in output power becomes coarser and more non-linear as the output power is reduced towards the minimum end of its control range. The EZRadioPRO PA is designed to provide the highest efficiency and lowest current consumption possible. The EZRadioPRO is designed to supply +13 dBm output power for less than 20 mA for applications that require operation from a single coin cell battery. The EZRadioPRO can operate with Class-E matching and output up to +13 dBm Tx power at a supply voltage of VDD = 3.3 V. All PA options are single-ended to allow for easy antenna matching and low BOM cost. Automatic ramp-up and ramp-down is automatically performed to reduce unwanted spectral spreading. Refer to “AN627: Si4460/61 Low-Power PA Matching” and “AN648: PA Matching” for details on TX matching options.

The chip’s TXRAMP pin is disabled by default to save current in cases where the on-chip PA provides sufficient output power to drive the antenna. In cases where on-chip PA will drive the external PA, and the external PA needs a ramping signal, TXRAMP is the signal to use. To enable TXRAMP, set the API Property PA_MODE[7] = 1. TXRAMP will start to ramp up, and ramp down at the SAME time as the internal on-chip PA ramps up/down.

However, the time constant of the TXRAMP signal for the external PA is programmed independently of the ramp time constant for the on-chip PA. The ramp time constant for TXRAMP is programmed by

the TC[3:0] field in the PA_RAMP_EX API property and provides the following approximate ramp times as a function of TC[3:0] value.

Table 4.7. Ramp Times as a Function of TC[3:0] Value

TC	Ramp Time (µs)
0	1.25
1	1.33
2	1.43
3	1.54
4	1.67
5	1.82
6	2.00
7	2.22
8	2.50
9	2.86
10	3.33
11	4.00
12	5.00
13	6.67
14	10.00
15	20.00

The ramping profile is close to a linear ramping profile with smoothed out corner when approaching Vhi and Vlo. The TXRAMP pin can source up to 1 mA without voltage drooping. The TXRAMP pin's sinking capability is equivalent to a 10 kOhm pull-down resistor.

Vhi = 3 V when Vdd > 3.3 V. When Vdd < 3.3 V, the Vhi will be closely following the Vdd, and ramping time will be smaller also.

Vlo = 0 V when NO current needed to be sunk into TXRAMP pin. If 10uA need to be sunk into the chip, Vlo will be 10 µA x 10k = 100 mV.

Table 4.8. Command

Number	Command	Summary
0x2200	PA_MODE	Sets PA type.
0x2201	PA_PWR_LVL	Adjust TX power in fine steps.
0x2202	PA_BIAS_CLKDUTY	Adjust TX power in coarse steps and optimizes for different match configurations.
0x2203	PA_TC	Changes the ramp up/down time of the PA.

4.1.5.4.1 EZRadioPRO: +20 dBm PA

The +20 dBm configuration utilizes a class-E matching configuration for all frequency bands except 169 MHz where it uses a Square Wave match. Typical performance for the 915 MHz band for output power steps, voltage, and temperature are shown in Figure 4.10 (p. 31). The output power is changed in 128 steps through PA_PWR_LVL API. For detailed matching values, BOM, and performance at other frequencies, refer to "AN648: PA Matching".

Figure 4.10. +20 dBm TX Power vs. PA_PWR_LVL

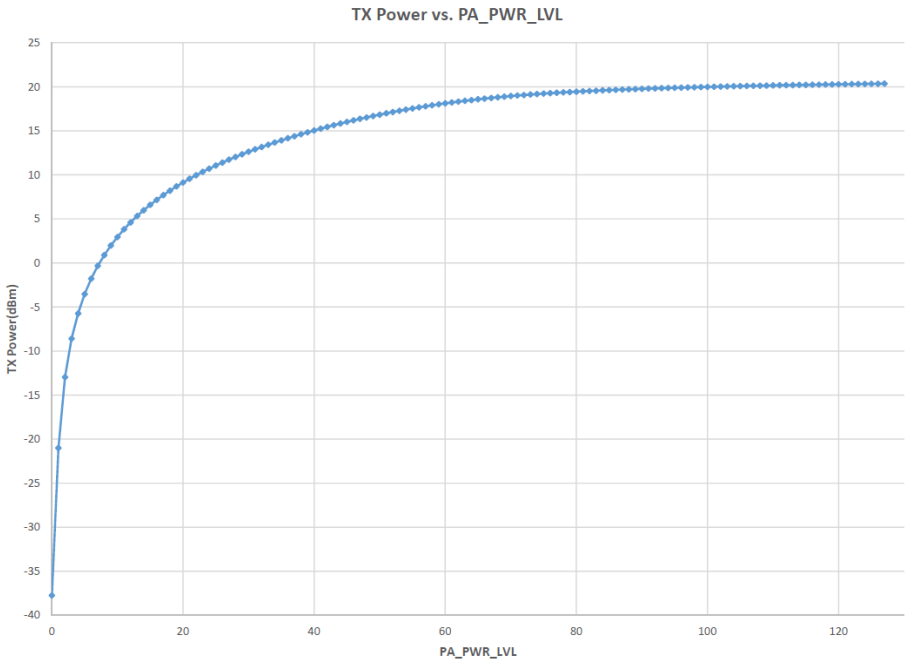


Figure 4.11. +20 dBm TX Power vs. VDD

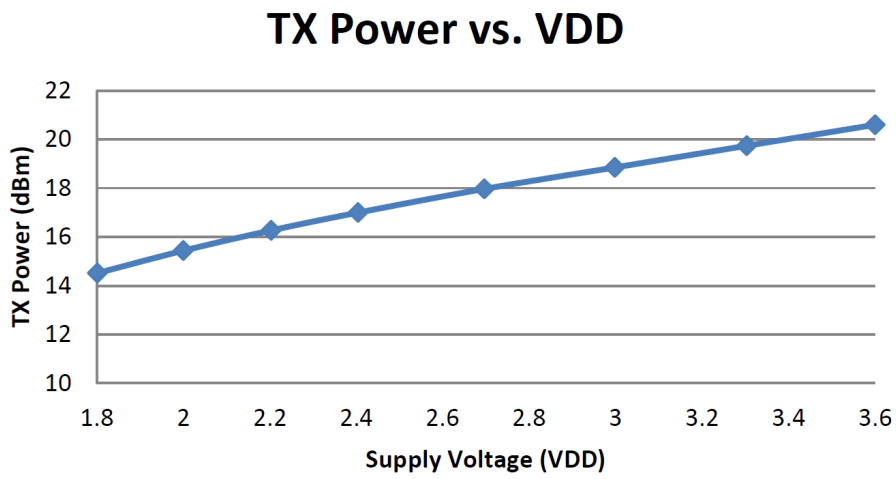
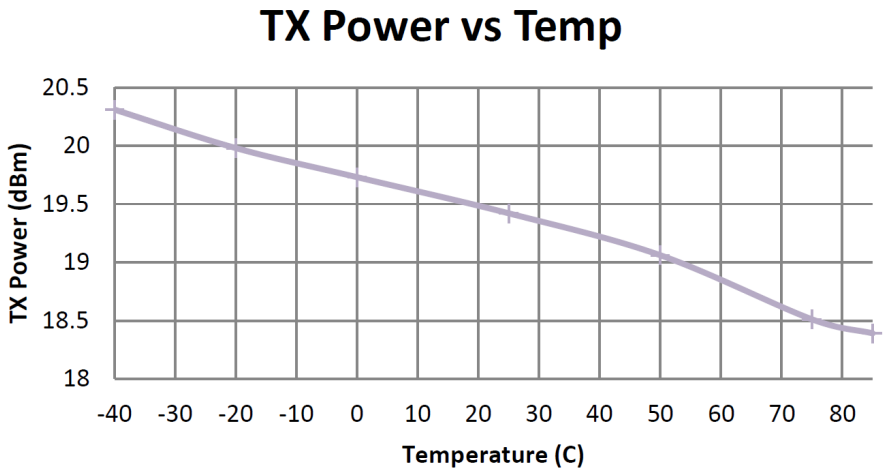


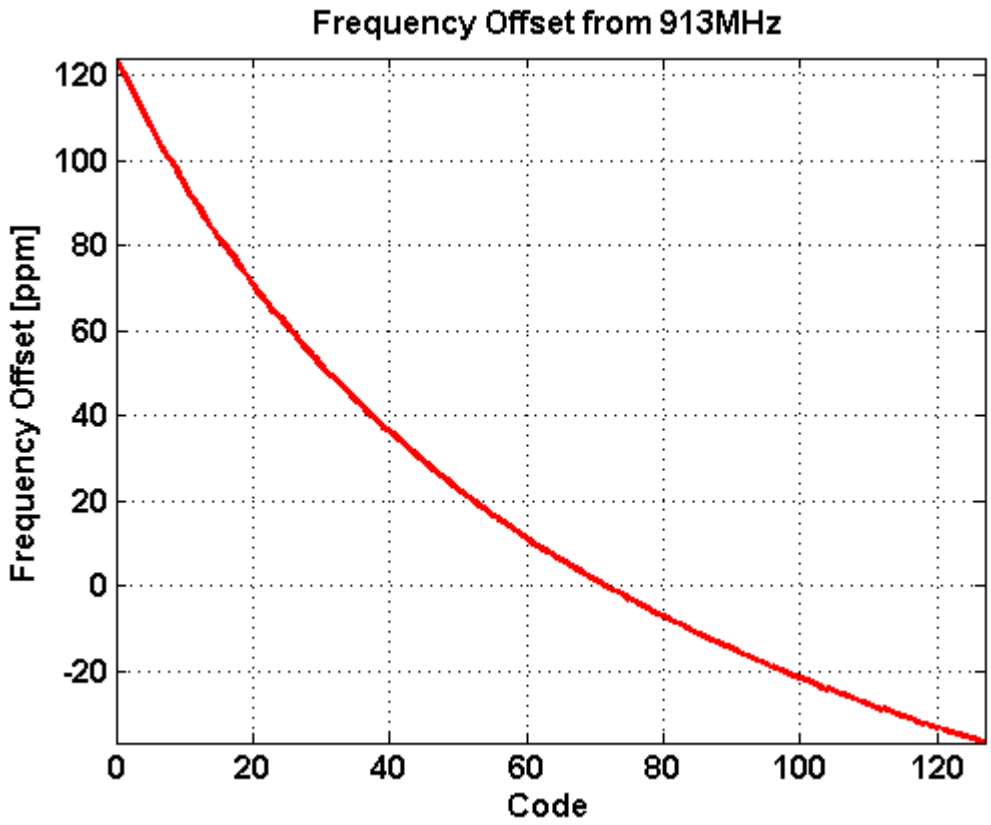
Figure 4.12. +20 dBm TX Power vs. Temp



4.1.5.5 Crystal Oscillator

The EZRadioPRO includes an integrated crystal oscillator with a fast start-up time of less than 250 µs. The design is differential with the required crystal load capacitance integrated on-chip to minimize the number of external components. By default, all that is required off-chip is the crystal. The default crystal is 30 MHz, but the circuit is designed to handle any XTAL from 25 to 32 MHz. If a crystal different than 30 MHz is used, the POWER_UP API boot command must be modified. The SS calculator crystal frequency field must also be changed to reflect the frequency being used. The crystal load capacitance can be digitally programmed to accommodate crystals with various load capacitance requirements and to adjust the frequency of the crystal oscillator. The tuning of the crystal load capacitance is programmed through the GLOBAL_XO_TUNE API property. The total internal capacitance is 11 pF and is adjustable in 127 steps (70 fF/step). The crystal frequency adjustment can be used to compensate for crystal production tolerances. The frequency offset characteristics of the capacitor bank are demonstrated in Figure 4.13 (p. 32) .

Figure 4.13. Capacitor Bank Frequency Offset Characteristics



Utilizing the on-chip temperature sensor and suitable control software, the temperature dependency of the crystal can be canceled.

A TCXO or external signal source can easily be used in place of a conventional XTAL and should be connected to the XIN pin. The incoming clock signal is recommended to have a peak-to-peak swing in the range of 600 mV to 1.4 V and ac-coupled to the XIN pin. If the peak-to-peak swing of the TCXO exceeds 1.4 V peak-to-peak, then dc coupling to the XIN pin should be used. The maximum allowed swing on XIN is 1.8 V peak-to-peak.

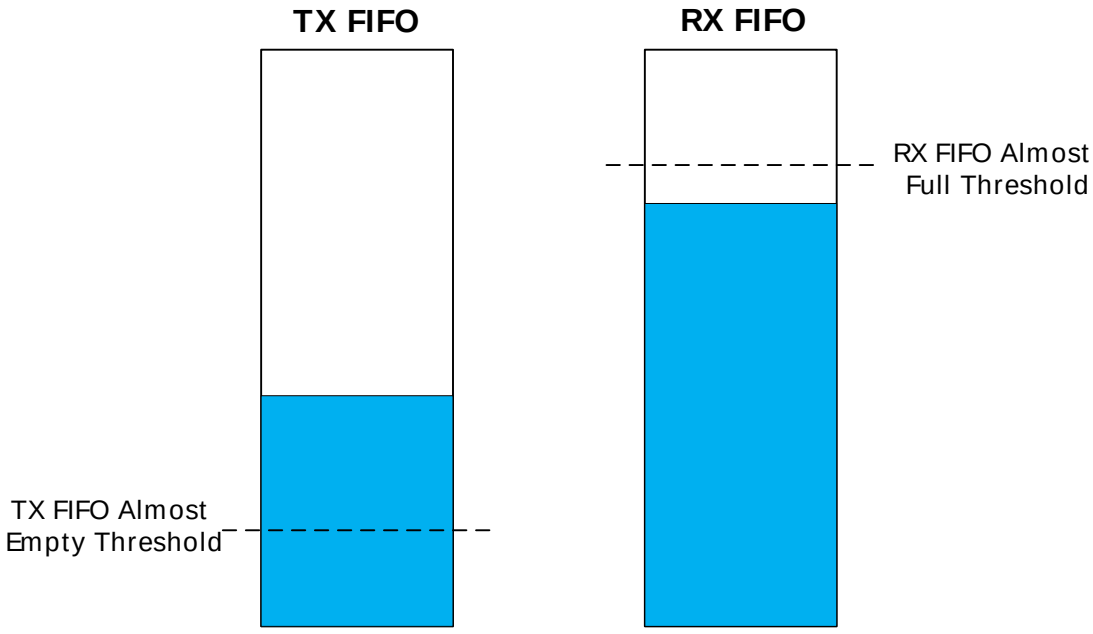
The XO capacitor bank should be set to 0 whenever an external drive is used on the XIN pin. In addition, the POWER_UP command should be invoked with the TCXO option whenever external drive is used.

4.1.6 Data Handling and Packet Handler

4.1.6.1 RX and TX FIFOs

Two 64-byte FIFOs are integrated into the chip, one for RX and one for TX, as shown in Figure 4.14 (p. 33) . For dedicated TX or RX, the FIFO size is up to 129 bytes. Writing to command Register 66h loads data into the TX FIFO, and reading from command Register 77h reads data from the RX FIFO. The TX FIFO has a threshold for when the FIFO is almost empty, which is set by the “TX_FIFO_EMPTY” property. An interrupt event occurs when the data in the TX FIFO reaches the almost empty threshold. If more data is not loaded into the FIFO, the chip automatically exits the TX state after the PACKET_SENT interrupt occurs. The RX FIFO has one programmable threshold, which is programmed by setting the “RX_FIFO_FULL” property. When the incoming RX data crosses the Almost Full Threshold, an interrupt will be generated to the microcontroller via the nIRQ pin. The microcontroller will then need to read the data from the RX FIFO. The RX Almost Full Threshold indication implies that the host can read at least the threshold number of bytes from the RX FIFO at that time. Both the TX and RX FIFOs may be cleared or reset with the “FIFO_RESET” command.

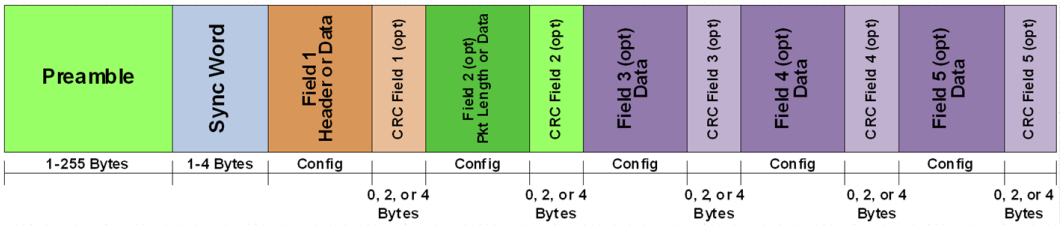
Figure 4.14. TX and RX FIFOs



4.1.6.2 Packet Handler

When using the FIFOs, automatic packet handling may be enabled for TX mode, RX mode, or both. The usual fields for network communication, such as preamble, synchronization word, headers, packet length, and CRC, can be configured to be automatically added to the data payload. The fields needed for packet generation normally change infrequently and can therefore be stored in registers. Automatically adding these fields to the data payload in TX mode and automatically checking them in RX mode greatly reduces the amount of communication between the microcontroller and EZRadioPRO. It also greatly reduces the required computational power of the microcontroller. The general packet structure is shown in Figure 4.15 (p. 34) . Any or all of the fields can be enabled and checked by the internal packet handler.

Figure 4.15. Packet Handler Structure



The fields are highly programmable and can be used to check any kind of pattern in a packet structure. The general functions of the packet handler include the following:

- Detection/validation of Preamble quality in RX mode (PREAMBLE_VALID signal)
- Detection of Sync word in RX mode (SYNC_OK signal)
- Detection of valid packets in RX mode (PKT_VALID signal)
- Detection of CRC errors in RX mode (CRC_ERR signal)
- Data de-whitening and/or Manchester decoding (if enabled) in RX mode
- Match/Header checking in RX mode
- Storage of Data Field bytes into FIFO memory in RX mode
- Construction of Preamble field in TX mode
- Construction of Sync field in TX mode
- Construction of Data Field from FIFO memory in TX mode
- Construction of CRC field (if enabled) in TX mode
- Data whitening and/or Manchester encoding (if enabled) in TX mode

For details on how to configure the packet handler, see “AN626: Packet Handler Operation for Si446x RFICs”.

4.1.7 RX Modem Configuration

The EZRadioPRO can easily be configured for different data rate, deviation, frequency, etc. by using the Radio Configuration Application (RCA) GUI which is part of Simplicity Studio (SS).

4.1.8 Auxiliary Blocks

4.1.8.1 Wake-up Timer and 32 kHz Clock Source

The chip contains an integrated wake-up timer that can be used to periodically wake the chip from sleep mode. The wake-up timer runs from either the internal 32 kHz RC Oscillator, or from an external 32 kHz XTAL.

The wake-up timer can be configured to run when in sleep mode. If WUT_EN = 1 in the GLOBAL_WUT_CONFIG property, prior to entering sleep mode, the wake-up timer will count for a time specified defined by the GLOBAL_WUT_R and GLOBAL_WUT_M properties. At the expiration of this period, an interrupt will be generated on the nIRQ pin if this interrupt is enabled in the INT_CTL_CHIP_ENABLE property. The microcontroller will then need to verify the interrupt by reading the chip interrupt status either via GET_INT_STATUS or a fast response register. The formula for calculating the Wake-Up Period is as follows:

Wake-up Time Equation

$$WUT = WUT\_M * (4 * 2^{WUT\_R} / 32768 \quad (4.3)$$

The RC oscillator frequency will change with temperature; so, a periodic recalibration is required. The RC oscillator is automatically calibrated during the POWER_UP command and exits from the Shutdown state. To enable the recalibration feature, CAL_EN must be set in the GLOBAL_WUT_CONFIG property, and the desired calibration period should be selected via WUT_CAL_PERIOD[2:0] in the same API

property. During the calibration, the 32 kHz RC oscillator frequency is compared to the 30 MHz XTAL and then adjusted accordingly. The calibration needs to start the 30 MHz XTAL, which increases the average current consumption; so, a longer CAL_PERIOD results in a lower average current consumption. The 32 kHz XTAL accuracy is comprised of both the XTAL parameters and the internal circuit. The XTAL accuracy can be defined as the XTAL initial error + XTAL aging + XTAL temperature drift + detuning from the internal oscillator circuit. The error caused by the internal circuit is typically less than 10 ppm. Refer to API documentation for details on WUT related commands and properties.

4.1.8.2 Low Duty Cycle Mode (Auto RX Wake-Up)

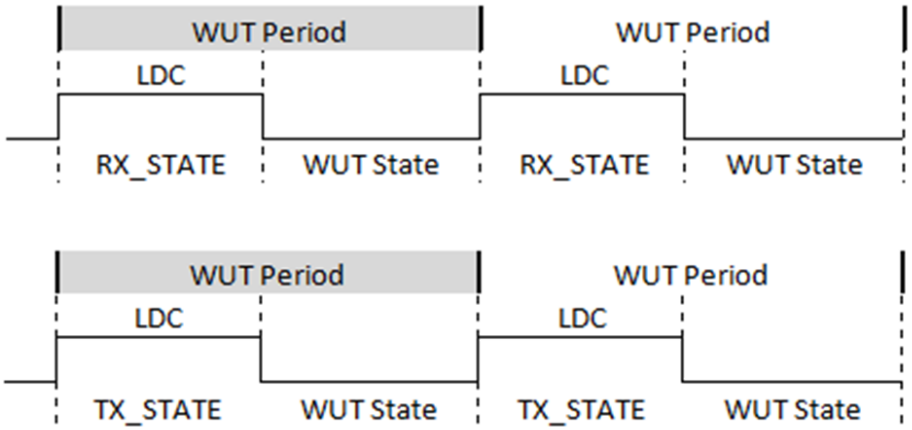
The low duty cycle (LDC) mode is implemented to automatically wake-up the receiver to check if a valid signal is available or to enable the transmitter to send a packet. It allows low average current polling operation by the EZRadioPRO for which the wake-up timer (WUT) is used. RX and TX LDC operation must be set via the GLOBAL_WUT_CONFIG property when setting up the WUT. The LDC wake-up period is determined by the following formula:

LDC Wake-up Time Equation

$$LDC = WUT\_LDC * (4 * 2^{WUT\_R} / 32768 \quad (4.4)$$

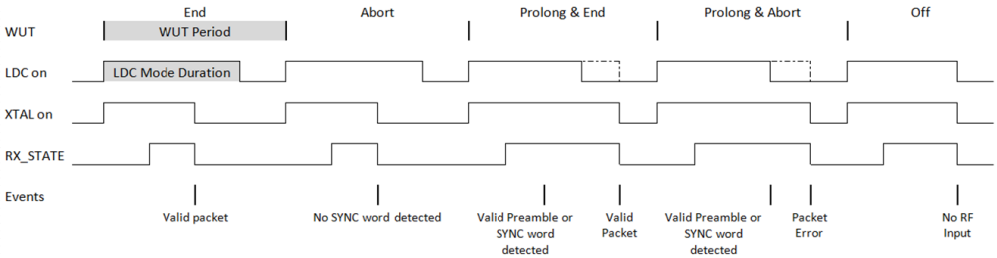
where the WUT_LDC parameter can be set by the GLOBAL_WUT_LDC property. The WUT period must be set in conjunction with the LDC mode duration; for the relevant API properties, see the wake-up timer (WUT) section.

Figure 4.16. RX and TX LDC Sequences



The basic operation of RX LDC mode is shown in Figure 4.17 (p. 35). The receiver periodically wakes itself up to work on RX_STATE during LDC mode duration. If a valid preamble is not detected, a receive error is detected, or an entire packet is not received, the receiver returns to the WUT state (i.e., ready or sleep) at the end of LDC mode duration and remains in that mode until the beginning of the next wake-up period. If a valid preamble or sync word is detected, the receiver delays the LDC mode duration to receive the entire packet. If a packet is not received during two LDC mode durations, the receiver returns to the WUT state at the last LDC mode duration until the beginning of the next wake-up period.

Figure 4.17. Low Duty Cycle Mode for RX



In TX LDC mode, the transmitter periodically wakes itself up to transmit a packet that is in the data buffer. If a packet has been transmitted, nIRQ goes low if the option is set in the INT_CTL_ENABLE property. After transmitting, the transmitter immediately returns to the WUT state and stays there until the next wake-up time expires.

4.1.8.3 Temperature, Battery Voltage, and Auxiliary ADC

The EZRadioPRO family contains an integrated auxiliary ADC for measuring internal battery voltage, an internal temperature sensor, or an external component over a GPIO. The ADC utilizes a SAR architecture and achieves 11-bit resolution. The Effective Number of Bits (ENOB) is 9 bits. When measuring external components, the input voltage range is 1 V, and the conversion rate is between 300 Hz to 2.44 kHz. The ADC value is read by first sending the GET_ADC_READING command and enabling the inputs that are desired to be read: GPIO, battery, or temp. The temperature sensor accuracy at 25 °C is typically ±2 °C. Refer to API documentation for details on the command and reply stream.

4.1.8.4 Low Battery Detector

The low battery detector (LBD) is enabled and utilized as part of the wake-up-timer (WUT). The LBD function is not available unless the WUT is enabled, but the host MCU can manually check the battery voltage anytime with the auxiliary ADC. The LBD function is enabled in the GLOBAL_WUT_CONFIG API property. The battery voltage will be compared against the threshold each time the WUT expires. The threshold for the LBD function is set in GLOBAL_LOW_BATT_THRESH. The threshold steps are in increments of 50 mV, ranging from a minimum of 1.5 V up to 3.05 V. The accuracy of the LBD is ±3%. The LBD notification can be configured as an interrupt on the nIRQ pin or enabled as a direct function on one of the GPIOs.

4.1.8.5 Antenna Diversity

To mitigate the problem of frequency-selective fading due to multipath propagation, some transceiver systems use a scheme known as antenna diversity. In this scheme, two antennas are used. Each time the transceiver enters RX mode the receive signal strength from each antenna is evaluated. This evaluation process takes place during the preamble portion of the packet. The antenna with the strongest received signal is then used for the remainder of that RX packet. The same antenna will also be used for the next corresponding TX packet. This chip fully supports antenna diversity with an integrated antenna diversity control algorithm. The required signals needed to control an external SPDT RF switch (such as a PIN diode or GaAs switch) are available on the GPIOx pins. The operation of these GPIO signals is programmable to allow for different antenna diversity architectures and configurations. The antdiv[2:0] bits are found in the MODEM_ANT_DIV_CONTROL API property descriptions and enable the antenna diversity mode. The GPIO pins are capable of sourcing up to 5 mA of current; so, it may be used directly to forward-bias a PIN diode if desired. The antenna diversity algorithm will automatically toggle back and forth between the antennas until the packet starts to arrive. The recommended preamble length for optimal antenna selection is 8 bytes.

4.1.8.6 Preamble Sense Mode

This mode of operation is suitable for extremely low power applications where power consumption is important. The preamble sense mode (PSM) takes advantage of the Digital Signal Arrival detector (DSA), which can detect a preamble within eight bit times with no sensitivity degradation. This fast detection of an incoming signal can be combined with duty cycling of the receiver during the time the device is searching or sniffing for packets over the air. The average receive current is lowered significantly when using this mode. In applications where the timing of the incoming signal is unknown, the amount of power savings is primarily dependent on the data rate and preamble length as the Rx inactive time is determined by these factors. In applications where the sleep time is fixed and the timing of the incoming signal is known, the average current also depends on the sleep time. The PSM mode is similar to the low duty cycle mode but has the benefit of faster signal detection and autonomous duty cycling of the receiver to achieve even lower average receive currents. This mode can be used with the low power mode (LP) which has an active RX current of 10 mA or with the high-performance (HP) mode which has an active RX current of 13 mA.

Figure 4.18. Preamble Sense Mode

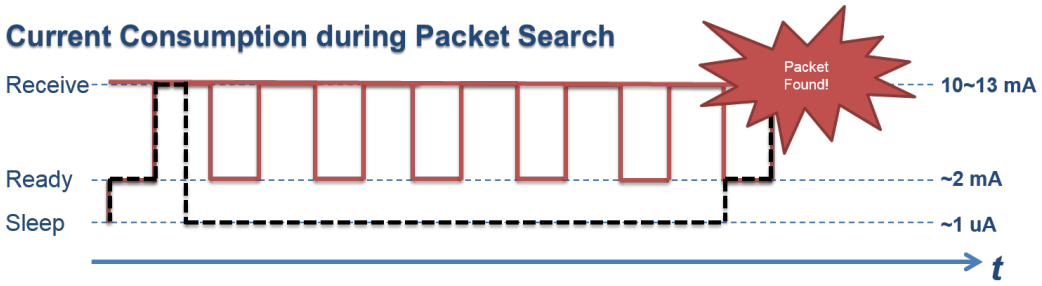


Table 4.9. Data Rates

	Data Rate				
	1.2 kbps	9.6 kbps	50 kbps	100 kbps	
PM length = 4 bytes	6.48	6.84	8.44	10.43	mA
PM length = 8 bytes	3.83	3.96	4.57	5.33	mA

Note

Typical values. Active RX current is 13 mA.

4.1.9 Wireless MBUS support

Wireless M-Bus is a widely accepted standard for smart meter communication in Europe. The radio supports all Wireless M-Bus modes per the latest draft specification of the EN13757-4 standard. This includes a much wider deviation error tolerance of ±30% and frequency error tolerance of ±4 kHz, short preamble support (16-bit preamble for 2 and 4 level FSK modes), 3-of-6 encoding and decoding and 169 MHz N modes including N2g.

In addition, Silicon Labs has a production ready Wireless M-Bus stack available at no additional cost which supports all modes and runs on the EZR32 (32-bit ARM) family of energy friendly microcontrollers. This stack and complete documentation including PHY configuration and test results are available for download from the EZRadioPRO page on the Silicon Labs website.

4.1.10 ETSI EN300 220 Category 1

The radio is capable of supporting ETSI Category 1 applications (social alarms, healthcare applications, etc.) in the 169 MHz and 868 MHz bands. Blocking performance is improved at the 2 MHz and 10 MHz offsets allowing for additional margin from the regulatory limits. The radio complies with ACS limits at the 25 kHz offset in both, 169 MHz and 868 MHz bands. In the 169 MHz band, there is no need for an external SAW filter for 2 MHz and 10 MHz blocking resulting in a lower system cost. In the 868 MHz band, an external SAW filter is still required to meet the Cat 1 blocking limits. An RF Pico board is available for evaluation specifically for ETSI Cat 1 applications.

Test conditions for ETSI Cat 1 specifications are different from the typical conditions and are stated below.

- Data Rate: 3 kbps
- Deviation: 2 kHz
- Modulation: 2 GFSK
- IF mode: Fixed and/or Scaled IF
- RX bandwidth: 13 kHz
- BER target: 0.1%
- Blocker signal: CW

Table 4.10. Energy Mode Description

	ETSI Cat 1 limits	169 MHz band(no SAW)	868 MHz band(no SAW)
±25 kHz ACS	54 dB	62 dB	58 dB
±2 MHz blocking	84 dB	88 dB	76 dB
±10 MHz blocking	84 dB	90 dB	82 dB
RX sensitivity	-107 dB	-108 dB	-108 dB
Spurious response	35 dB	40 dB	40 dB

For further details on configuring the radio for ETSI Cat 1 applications, refer to the application notes available on the Silicon Labs website

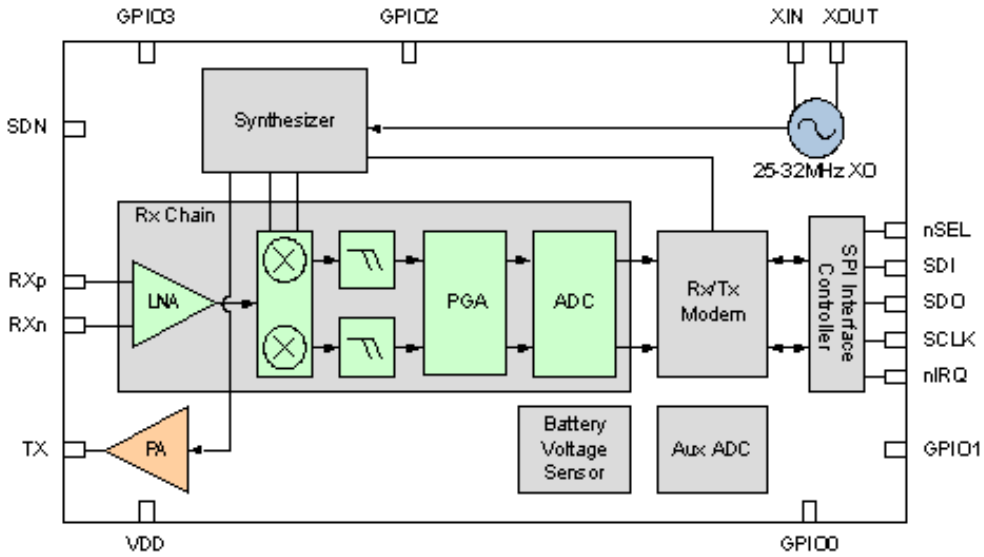
4.2 EZRadio Overview

4.2.1 Introduction

The EZRadio with its +13 dBm output power and excellent sensitivity of -116 dBm allows for a longer operating range, while the low current consumption of 18 mA TX (at 10 dBm), 10 mA RX, and 40 nA standby, provides for superior battery life. By fully integrating all components from the antenna to the GPIO or SPI interface to the MCU, the EZRadio makes it easy to realize this performance in an application. The EZRadio is capable of supporting major worldwide regulatory standards, such as FCC, ETSI, ARIB, and China regulatory standards.

4.2.2 Functional Description

Figure 4.19. EZRadio Block Diagram



The EZRadio is an easy-to-use, size efficient, low current wireless ISM device that covers the sub-GHz bands. The wide operating voltage range of 1.8-3.6 V and low current consumption make the EZRadio an ideal solution for battery powered applications. The EZRadio operates as a time division duplexing (TDD) transceiver where the device alternately transmits and receives data packets. The device uses a single-conversion mixer to downconvert the FSK/GFSK or OOK modulated receive signal to a low IF frequency. Following a programmable gain amplifier (PGA), the signal is converted to the digital domain by a high performance ΔΣ ADC allowing filtering, demodulation, slicing, and packet handling to be performed in the built-in digital modem, increasing the receiver's performance and flexibility versus analog based architectures. The demodulated signal is output to the system MCU through a programmable GPIO or via the standard SPI bus by reading the 64-byte Rx FIFO.

A single high-precision local oscillator (LO) is used for both transmit and receive modes since the transmitter and receiver do not operate at the same time. The LO signal is generated by an integrated VCO and ΔΣ Fractional-N PLL synthesizer. The synthesizer is designed to support configurable data rates up to 500 kbps. The EZRadio operates in the frequency bands of 283-350, 350-525, and 850-960 MHz. The transmit FSK data is modulated directly into the ΔΣ data stream and can be shaped by a Gaussian low-pass filter to reduce unwanted spectral content.

The device contains a power amplifier (PA) that supports output powers up to +13 dBm and is designed to support single coin cell operation with current consumption of 18 mA for +10 dBm output power. The PA is single-ended to allow for easy antenna matching and low BOM cost. The PA incorporates automatic ramp-up and ramp-down control to reduce unwanted spectral spreading. Additional system features, such as 64-byte TX/RX FIFOs, preamble detection, sync word detector, and CRC, reduce overall current consumption and allow for the use of lower-cost system MCUs. Power-on-reset (POR) and GPIOs further reduce overall system cost and size. The EZRadio is designed to work with an MCU, crystal, and a few passives to create a very compact and low-cost system.

4.2.2.1 Receiver Chain

The internal low-noise amplifier (LNA) is designed to be a wideband LNA that can be matched with three external discrete components to cover any common range of frequencies in the sub-GHz band. The LNA has extremely low noise to suppress the noise of the following stages and achieve optimal sensitivity; therefore, no external gain or front-end modules are necessary. The LNA has gain control, which is controlled by the internal automatic gain control (AGC) algorithm. The LNA is followed by an I-Q mixer, filter, programmable gain amplifier (PGA), and ADC. The I-Q mixers downconvert the signal to an intermediate frequency. The PGA then boosts the gain to be within dynamic range of the ADC. The ADC rejects out-of-band blockers and converts the signal to the digital domain where filtering, demodulation, and processing is performed. Peak detectors are integrated at the output of the LNA and PGA for use in the AGC algorithm.

The RX and TX pins can be directly tied externally on the EZRadio transceiver.

4.2.2.2 Receiver Modem

Using high-performance ADCs allows channel filtering, image rejection, and demodulation to be performed in the digital domain, which allows for flexibility in optimizing the device for particular applications. The digital modem performs the following functions:

- Channel selection filter
- Preamble detection
- Invalid preamble detection
- TX modulation
- RX demodulation
- Automatic Gain Control (AGC)
- Automatic frequency compensation (AFC)
- Radio signal strength indicator (RSSI)
- Cyclic redundancy check (CRC)

The digital channel filter and demodulator are optimized for ultra-low-power consumption and are highly configurable. Supported modulation types are GFSK, FSK, and OOK. The channel filter can be configured to support bandwidths ranging from 850 kHz down to 40 kHz. A large variety of data rates are supported ranging from 0.5 kbps up to 500 kbps. The configurable preamble detector is used with the synchronous demodulator to improve the reliability of the sync-word detection. Preamble detection can be skipped using only sync detection, which is a valuable feature of the asynchronous demodulator when very short preambles are used. The received signal strength indicator (RSSI) provides a measure of the signal strength received on the tuned channel. The resolution of the RSSI is 0.5 dB. This high-resolution RSSI enables accurate channel power measurements for clear channel assessment (CCA), carrier sense (CS), and listen before talk (LBT) functionality. A wireless communication channel can be

corrupted by noise and interference, so it is important to know if the received data is free of errors. A cyclic redundancy check (CRC) is used to detect the presence of erroneous bits in each packet. A CRC is computed and appended at the end of each transmitted packet and verified by the receiver to confirm that no errors have occurred. The packet handler and CRC can significantly reduce the load on the system microcontroller, allowing for a simpler and cheaper microcontroller. The digital modem includes the TX modulator, which converts the TX data bits into the corresponding stream of digital modulation values to be summed with the fractional input to the sigma-delta modulator. This modulation approach results in highly accurate resolution of the frequency deviation. A Gaussian filter is implemented to support GFSK, considerably reducing the energy in adjacent channels. The default bandwidth-time (BT) product is 0.5 for all programmed data rates.

4.2.2.3 Received Signal Strength Indicator

The received signal strength indicator (RSSI) is an estimate of the signal strength in the channel to which the receiver is tuned. The RSSI measurement is done after the channel filter, so it is only a measurement of the desired or undesired in-band signal power. The EZRadio uses a fast response register to read RSSI and so can complete the read in 16 SPI clock cycles with no requirement to wait for CTS. The RSSI value reported by this API command can be converted to dBm using the following equation:

RSSI Equation

$$RSSI_{dBm} = (RSSI\_value\backslash 2) - 130 \quad (4.5)$$

The value of 130 in the above formula is based on bench characterization of the EZRadio RF Pico boards (evaluation boards). The RSSI value is latched at sync word detection and can be read via the fast response register. The latched value of RSSI is available until the device re-enters Rx mode. In addition, the current value of RSSI can be read out using the GET_MODEM_STATUS command. This can be used to implement CCA (clear channel assessment) functionality. The user can set up an RSSI threshold value using the SS Radio Configuration Application GUI.

4.2.2.4 Synthesizer

The EZRadio includes an integrated Sigma Delta (ΣΔ) Fractional-N PLL synthesizer capable of operating over the bands from 283-350, 350-525, and 850-960 MHz. The synthesizer has many advantages; it provides flexibility in choosing data rate, deviation, channel frequency, and channel spacing. The transmit modulation is applied directly to the loop in the digital domain through the fractional divider, which results in very precise accuracy and control over the transmit deviation. The frequency resolution is (2/3)Freq_xo/(219) for 283-350 MHz, Freq_xo/(219) for 350-525 MHz, and Freq_xo/(218) for 850-960 MHz. The nominal reference frequency to the PLL is 30 MHz, but any XTAL frequency from 25 to 32 MHz may be used. The modem configuration calculator in SS will automatically account for the XTAL frequency being used. The PLL utilizes a differential LC VCO with integrated on-chip inductors. The output of the VCO is followed by a configurable divider, which will divide the signal down to the desired output frequency band.

4.2.2.4.1 Synthesizer Frequency Control

The frequency is set by changing the integer and fractional settings to the synthesizer. The SS calculator will automatically provide these settings, but the synthesizer equation is shown below for convenience. Initial frequency settings are configured in the EZConfig setup and can also be modified using the API commands: FREQ_CONTROL_INTE, FREQ_CONTROL_FRAC2, FREQ_CONTROL_FRAC1, and FREQ_CONTROL_FRAC0.

RF Frequency Equation

$$RF\ Frequency = (fc\_inte + fc\_frac\backslash 2^{19}) * (4 * freq\_xo\backslash outdiv) \quad (4.6)$$

Note

The $fc\_frac/2^{19}$ value in the above formula must be a number between 1 and 2. The LSB of fc_frac must be "1".

Table 4.11. Output Divider (Outdiv) Values

Outdiv	Lower (MHz)	Upper (MHz)
12	284	350
10	350	420
8	420	525
4	850	960

4.2.2.4.1.1 EZ Frequency Programming

EZ frequency programming allows for easily changing radio frequency using a single API command. The base frequency is first set using the EZConfig setup. This base frequency will correspond to channel 0. Next, a channel step size is also programmed within the EZConfig setup. The resulting frequency will be:

RF Frequency Equation

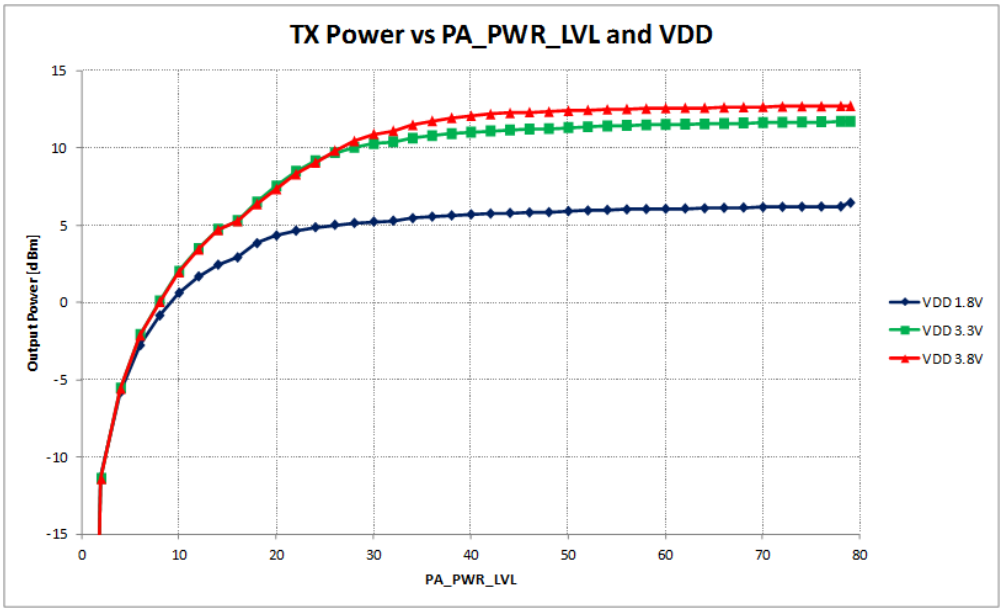
$$\text{RR Frequency} = \text{Base Frequency} + \text{Channel} * \text{Step Size} \quad (4.7)$$

The second argument of the START_RX or START_TX is CHANNEL, which sets the channel number for EZ frequency programming. For example, if the channel step size is set to 1 MHz, the base frequency is set to 900 MHz, and a CHANNEL number of 5 is programmed during the START_TX command, the resulting frequency will be 905 MHz. If no CHANNEL argument is written as part of the START_RX/TX command, it will default to the previous value. The initial value of CHANNEL is 0 and so will be set to the base frequency if this argument is never used.

4.2.2.5 Transmitter

The device contains a +13 dBm power amplifier that is capable of transmitting from -40 to +13 dBm. The output power set size is dependent on the power level and can be seen in Figure 4.20 (p. 41). The PA power level is set using the API command: PA_PWR_LVL. The power amplifier is single-ended to allow for easy antenna matching and low BOM cost. For detailed matching values, BOM, and performance expectations, refer to "AN686: Antennas for the Si4455/4355 RF ICs". Power ramp-up and ramp-down is automatically performed to reduce unwanted spectral spreading.

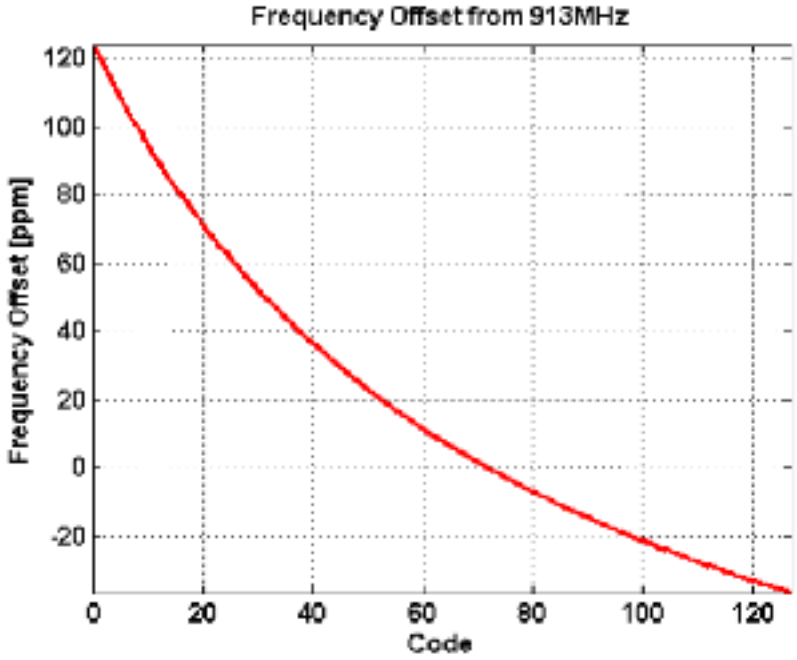
Figure 4.20. Tx Power vs PA_PWR_LVL and VDD



4.2.2.6 Crystal Oscillator

The EZRadio includes an integrated crystal oscillator with a fast start-up time of less than 250 µs. The design is differential with the required crystal load capacitance integrated on-chip to minimize the number of external components. By default, all that is required off-chip is the crystal. The default crystal is 30 MHz, but the circuit is designed to handle any XTAL from 25 to 32 MHz, set in the EZConfig setup. The crystal load capacitance can be digitally programmed to accommodate crystals with various load capacitance and to adjust the frequency of the crystal oscillator. The tuning of the crystal load capacitance is programmed through the GLOBAL_XO_TUNE API property. The total internal capacitance is 11 pF and is adjustable in 127 steps (70 fF/step). The crystal frequency adjustment can be used to compensate for crystal production tolerances. The frequency offset characteristics of the capacitor bank are demonstrated in Figure 4.21 (p. 42) .

Figure 4.21. Capacitor Bank Frequency Offset Characteristics



An external signal source can easily be used in lieu of a conventional XTAL and should be connected to the XIN pin. The incoming clock signal is recommended to be ac-coupled to the XIN pin since the dc bias is controlled by the internal crystal oscillator buffering circuitry. The input swing range should be between 600 mV-1.8 V peak-to-peak. If external drive is desired, the incoming signal amplitude should not go below 0 V or exceed 1.8 V. The best dc bias should be approximately 0.7 V. However, if the signal swing exceeds 1.4 Vpp, the dc bias can be set to 1/2 the peak-to-peak voltage swing. The XO capacitor bank should be set to 0 whenever an external drive is used on the XIN pin. In addition, the POWER_UP command should be invoked with the TCXO option whenever external drive is used.

4.2.2.7 Battery Voltage and Auxiliary ADC

The EZRadio contains an integrated auxiliary 11-bit ADC used for the internal battery voltage detector or an external component via GPIO. The Effective Number of Bits (ENOB) is 9 bits. When measuring external components, the input voltage range is 1 V, and the conversion rate is between 300 Hz to 2.44 kHz. The ADC value is read by first sending the GET_ADC_READING command and enabling the desired inputs. When the conversion is finished and all the data is ready, CTS will go high, and the data can be read out. For details on this command and the formulas needed to interpret the results, refer to the EZRadio API documentation zip file available from www.silabs.com.

4.2.3 Configuration Options and User Interface

4.2.3.1 Radio Configuration Application (RCA) GUI

The Radio Configuration Application (RCA) GUI is part of the Simplicity Studio (SS) program. This setup interface provides an easy path to quickly selecting and loading the desired configuration for the device. The RCA allows for two different methods for device setup. One option is the configuration table, which provides a list of preloaded, common configurations. A second option allows for custom configurations to be loaded. After the desired configuration is selected, the RCA automatically creates the EZConfig configuration array that will be passed to the chip for setup. The program then gives the option to load a sample project with the selected configuration onto the evaluation board or launch IDE with the new configuration array preloaded into the user program. For more information on EZConfig usage, refer to application note, “AN692: Si4355/Si4455 Programming Guide”.

4.2.3.1.1 Radio Configuration Application

The Radio Configuration Application provides an intuitive interface for directly modifying the device configuration. Using this control panel, the device parameters such as modulation type, data rate, frequency deviation, and any packet related settings can be set. The program then takes these parameters and automatically determines the appropriate device settings. This method allows the user to have complete flexibility in determining the configuration of the device without the need to translate the system requirements into device specific properties. The resulting configuration array is automatically generated and available for use in the user's program. The resulting configuration array is obfuscated; therefore, its content changes every time a new array is generated, even if the input parameters are the same.

4.2.3.2 Configuration Options

4.2.3.2.1 Frequency Band

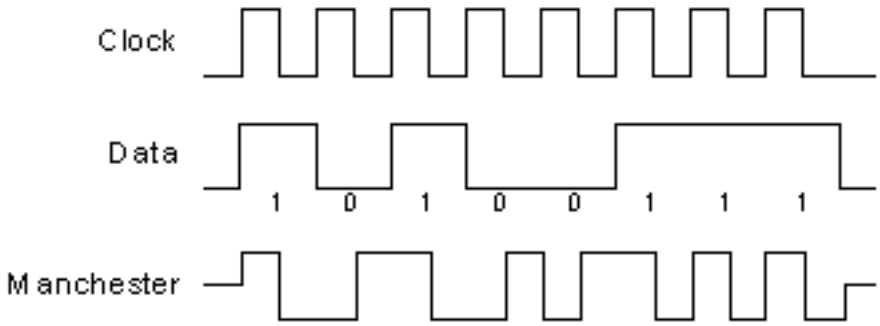
The EZRadio can operate in the 283-350 MHz, 350-525 MHz, or 850-960 MHz bands. One of these three bands will be selected during the configuration setup and then the specific transmission frequency that will be used within this band can be selected.

4.2.3.2.2 Modulation Type

The EZRadio can operate using On/Off Keying (OOK), Frequency Shift Keying (FSK), or Gaussian Frequency Shift Keying (GFSK). OOK modulation is the most basic modulation type available. It is the most power-efficient method and does not require as high oscillator accuracy as FSK. FSK provides the best sensitivity and range performance, but generally requires more precision from the oscillator used. GFSK is a version of FSK where the signal is passed through a Gaussian filter, limiting its spectral width. As a result, the out-of-band components of the signal are reduced.

The EZRadio also has an option for Manchester coding. This method provides a state transition at each bit and so allows for more reliable clock recovery. Manchester code is available only when using the packet handler option and, if selected, will be applied to the entire packet (the preamble pattern is set to continuous “1” if the Manchester mode is enabled; therefore, the chip rate of the resulting preamble pattern is the same as for the rest of the packet). The polarity can be configured to a “10” or “01”.

Figure 4.22. Manchester Code Example



4.2.3.2.3 Frequency Deviation

If FSK or GFSK modulation is selected, then a frequency deviation will also need to be selected. The frequency deviation is the maximum instantaneous difference between the FM modulated frequency and the nominal carrier frequency. The EZRadio can operate across a wide range of data rates and frequency deviations. If a frequency deviation needs to be selected, the following guideline might be helpful to build a robust link. A proper frequency deviation is linked to the frequency error between transmitter and receiver. The frequency error can be calculated using the crystal tolerance parameters and the RF operating frequency: (ppm_tx+ppm_rx)*Frf/1E-6. For frequency errors below 50 kHz, the deviation can be about the same as the frequency error. For frequency errors exceeding 50 kHz, the frequency deviation can be set to about 0.75 times the frequency error. It is advised to position the modulation index (= 2*freq_dev/data_rate) into a range between 1 and 100 for Packet Handling mode and 2 to 100 for direct mode (non-standard preamble). For example, when in Packet Handling mode and the frequency error is smaller than data_rate/2, the frequency deviation is set to about data_rate/2. When the frequency error exceeds 100xdata_rate/2, the frequency deviation is preferred to be set to 100xdata_rate/2.

4.2.3.2.4 Channel Bandwidth

The channel bandwidth sets the bandwidth for the receiver. Since the receiver bandwidth is directly proportional to the noise allowed in the system, this will normally be set as low as possible. The specific channel bandwidth used will usually be determined based upon the precision of the oscillator and the frequency deviation of the transmitted signal.

4.2.3.2.5 Preamble Length

A preamble is a defined simple bit sequence used to notify the receiver that a data transmission is imminent. The length of this preamble will normally be set as short as possible to minimize power while insuring that it will be reliably detected given the receiver characteristics, such as duty cycling and packet error rate performance. The EZRadio allows the preamble length to be set between 0 to 255 bytes in length with a default length of 4 bytes. The preamble pattern for the EZRadio will always be 55h with a first bit of “0” if the packet handler capability is used.

4.2.3.2.6 Sync Word Length and Pattern

The sync word follows the preamble in the packet structure and is used to identify the start of the payload data and to synchronize the receiver to the transmitted bit stream. The EZRadio allows for sync word lengths of 1 to 4 bytes, where the default is a 2 byte length 2d d4 pattern.

4.2.3.2.7 Cyclic Redundancy Check

Cyclic Redundancy Check (CRC) is used to verify that no errors have occurred during transmission and the received packet has exactly the same data as it did when transmitted. If this function is enabled in the EZRadio, the last byte of transmitted data must include the CRC generated by the transmitter. The

EZRadio then performs a CRC calculation on the received packet and compares that to the transmitted CRC. If these two values are the same, the EZRadio will set an interrupt indicating a valid packet has been received and is waiting in the Rx FIFO. If these two CRC values differ, the EZRadio will flag an interrupt indicating that a packet error occurred. The EZRadio uses CRC(16)-IBM: x16+x15+x2+1 with a seed of 0xFFFF as well as a 16-bit ITU-T CRC as specified in the IEEE 802.15.4g standard.

4.2.3.2.8 Preamble Sense Mode

This mode of operation is suitable for extremely low power applications where power consumption is important. The preamble sense mode (PSM) takes advantage of the Digital Signal Arrival detector (DSA), which can detect a preamble within eight bit times with no sensitivity degradation. This fast detection of an incoming signal can be combined with duty cycling of the receiver during the time the device is searching or sniffing for packets over the air. The average receive current is lowered significantly when using this mode. In applications where the timing of the incoming signal is unknown, the amount of power savings is primarily dependent on the data rate and preamble length as the Rx inactive time is determined by these factors. In applications where the sleep time is fixed and the timing of the incoming signal is known, the average current also depends on the sleep time. The PSM mode is similar to the low duty cycle mode but has the benefit of faster signal detection and autonomous duty cycling of the receiver to achieve even lower average receive currents.

Figure 4.23. Preamble Sense Mode

Table 4.12. Data Rates

	Data Rate				
	1.2 kbps	9.6 kbps	50 kbps	100 kbps	
	5.8	6.1	7.6	9.3	mA
	3.6	3.7	4.3	5.0	mA

Note

Typical values. Active RX current is 10.9 mA.

4.2.4 Controller Interface

4.2.4.1 Serial Peripheral Interface

The EZRadio communicates with the host MCU over a standard 4-wire serial peripheral interface (SPI): SCLK, SDI, SDO, and nSEL. The SPI interface is designed to operate at a maximum of 10 MHz. The SPI timing parameters are listed in Table 4.13 (p. 46). The host MCU writes data over the SDI pin and can read data from the device on the SDO output pin. Figure 4.25 (p. 46) shows an SPI write command. The nSEL pin should go low to initiate the SPI command. The first byte of SDI data will be one of the API commands followed by n bytes of parameter data which will be variable depending on the specific command. The rising edges of SCLK should be aligned with the center of the SDI data. For details regarding pin setup, see datasheet for the specific part.

Table 4.13. Serial Interface Timing Parameters

Symbol	Parameter	Min (ns)	Max (ns)
tCH	Clock high time	40	
tCL	Clock low time	40	
tDS	Data setup time	20	
tDH	Data hold time	20	
tDD	Output data delay time		43
tDE	Output disable time		45
tSS	Select setup time	20	
tSH	Select hold time	50	
tSW	Select high period	80	

Note

CL = 10 pF; VDD = 1.8 V; SDO Drive strength setting = 10.

Figure 4.24. Serial Interface Timing

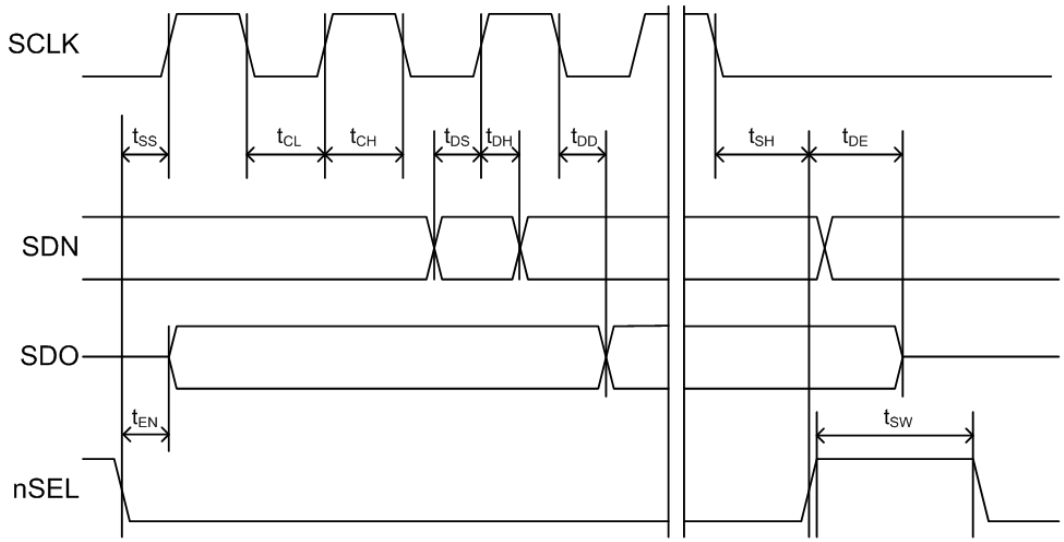
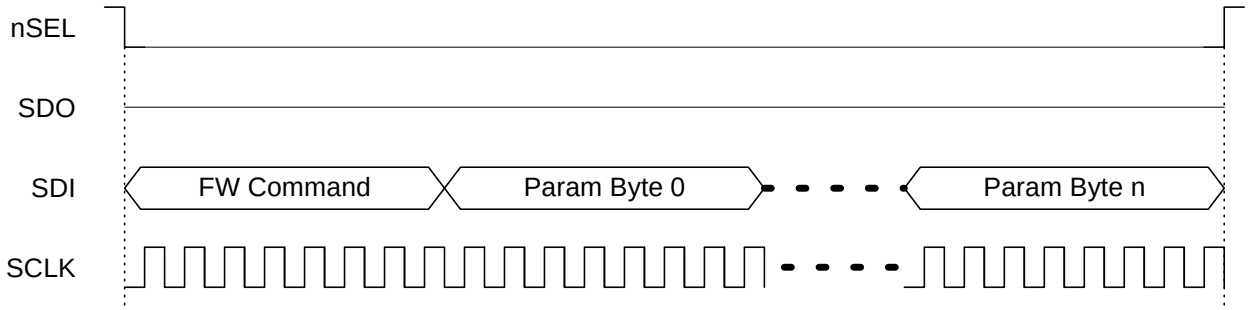


Figure 4.25. SPI Write Command



The EZRadio contains an internal MCU which controls all the internal functions of the radio. For SPI read commands, a typical communication flow of checking clear-to-send (CTS) is used to make sure the internal MCU has executed the command and prepared the data to be output over the SDO pin. Figure 4.26 (p. 47) demonstrates the general flow of an SPI read command. Once the CTS value reads FFh, then the read data is ready to be clocked out to the host MCU. The typical time for a valid FFh CTS reading is 20 µs. Figure 4.27 (p. 47) demonstrates the remaining read cycle after CTS is

set to FFh. The internal MCU will clock out the SDO data on the negative edge so the host MCU should process the SDO data on the rising edge of SCLK.

Figure 4.26. SPI Read Command-Check CTS Value

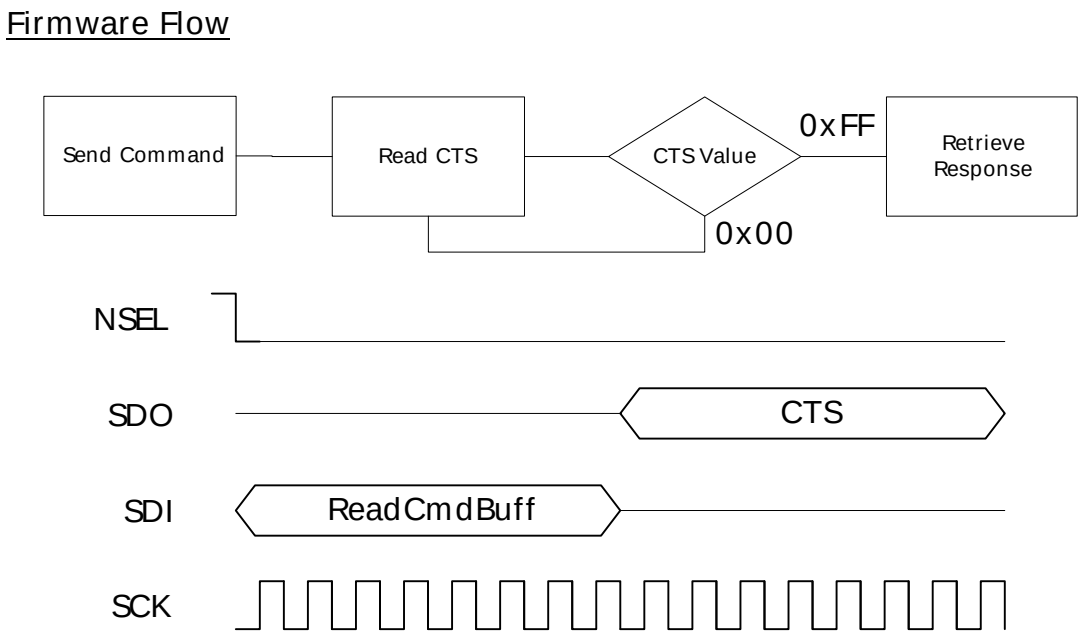
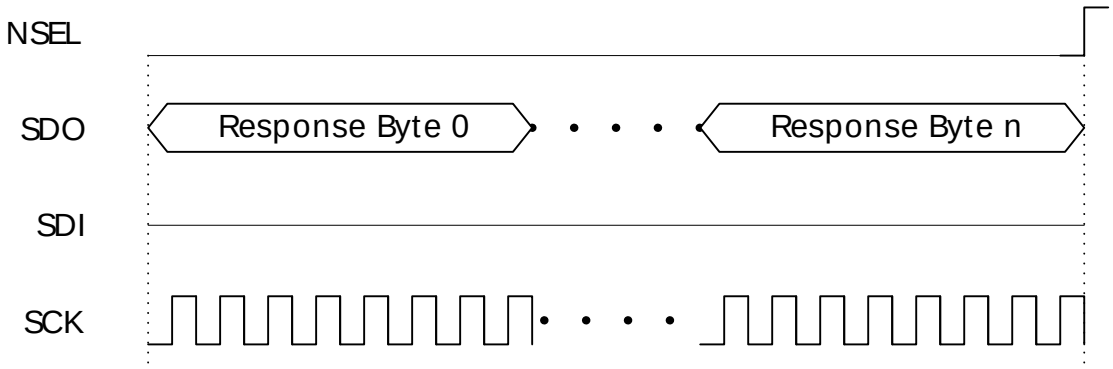


Figure 4.27. SPI Read Command-Clock Out Read Data



4.2.4.2 Operating Modes and Timing

The primary states of the EZRadio are shown in Figure 4.28 (p. 48). The shutdown state completely shuts down the radio, minimizing current consumption and is controlled using the SDN (pin 2). All other states are controlled using the API commands START_RX, START_TX and CHANGE_STATE. Table 4.14 (p. 48) shows each of the operating modes with the time required to reach either RX or TX state as well as the current consumption of each state. The times in Table 4.14 (p. 48) are measured from the rising edge of nSEL until the chip is in the desired state. This information is included for reference only since an automatic sequencer moves the chip from one state to another and so it is not necessary to manually step through each state. Figure 4.29 (p. 49) and Figure 4.30 (p. 49) demonstrate this timing and the current consumption for each radio state as the chip moves from shutdown or standby to TX and back. Most applications will utilize the standby mode since this provides the fastest transition response time, maintains all register values, and results in nearly the same current consumption as shutdown.

Figure 4.28. State Machine Diagram

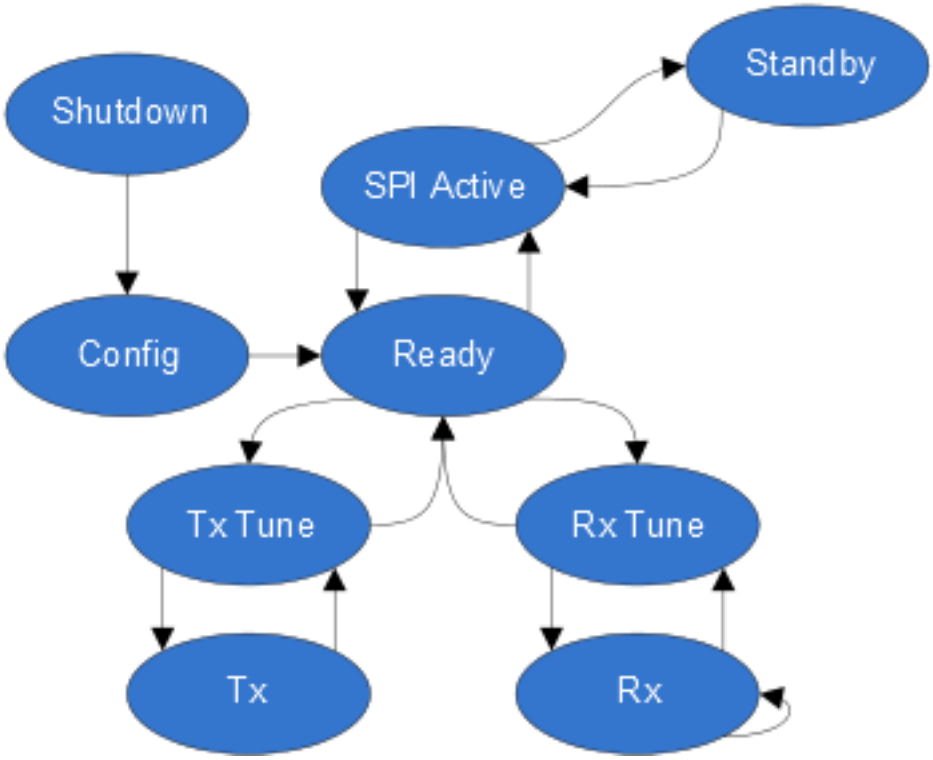


Table 4.14. Operating State Response Time and Current Consumption

State / Mode	Response Time to		Current in State / Mode
	Tx	Rx	
Shutdown	30 ms	30 ms	30 nA
Standby	504 µs	516 µs	40 nA
SPI Active	288 µs	296 µs	1.5 mA
Ready	108 µs	120 µs	1.8 mA
Tx Tune	60 µs		6.8 mA
Rx Tune		84 µs	7.1 mA
Tx		132 µs	18 mA @ +10 dBm
Rx	120 µs	108 µs	10.9 mA

Figure 4.29. Start-Up Timing and Current Consumption using Shutdown State

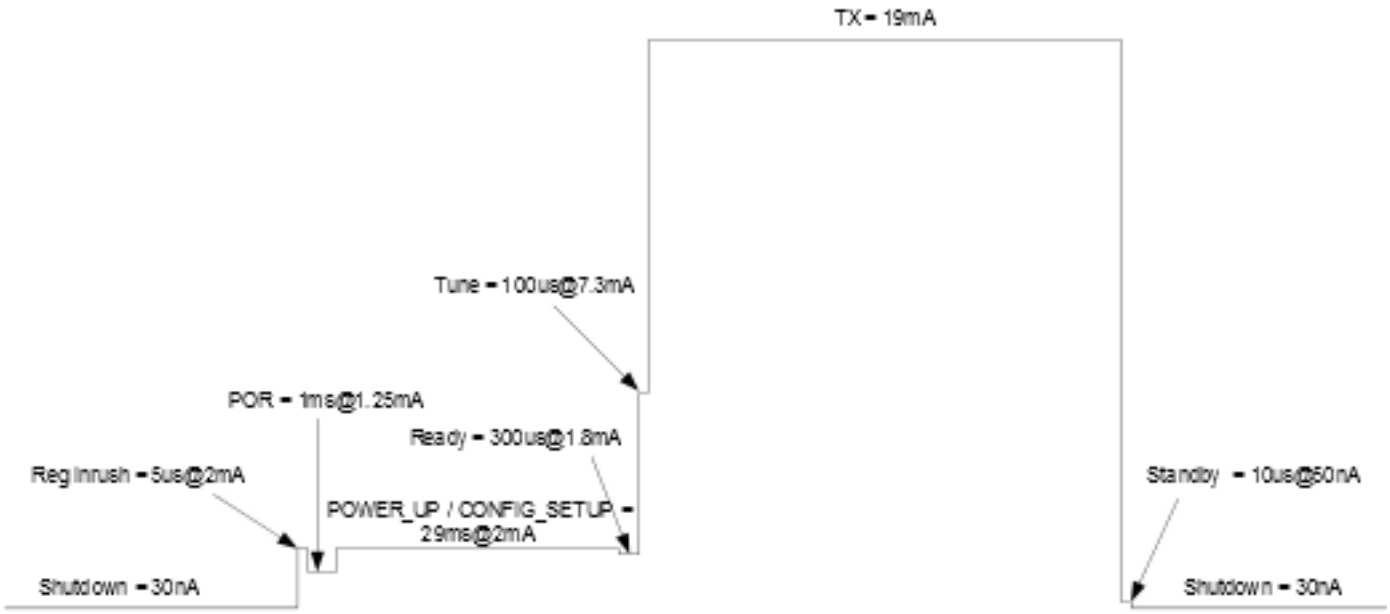
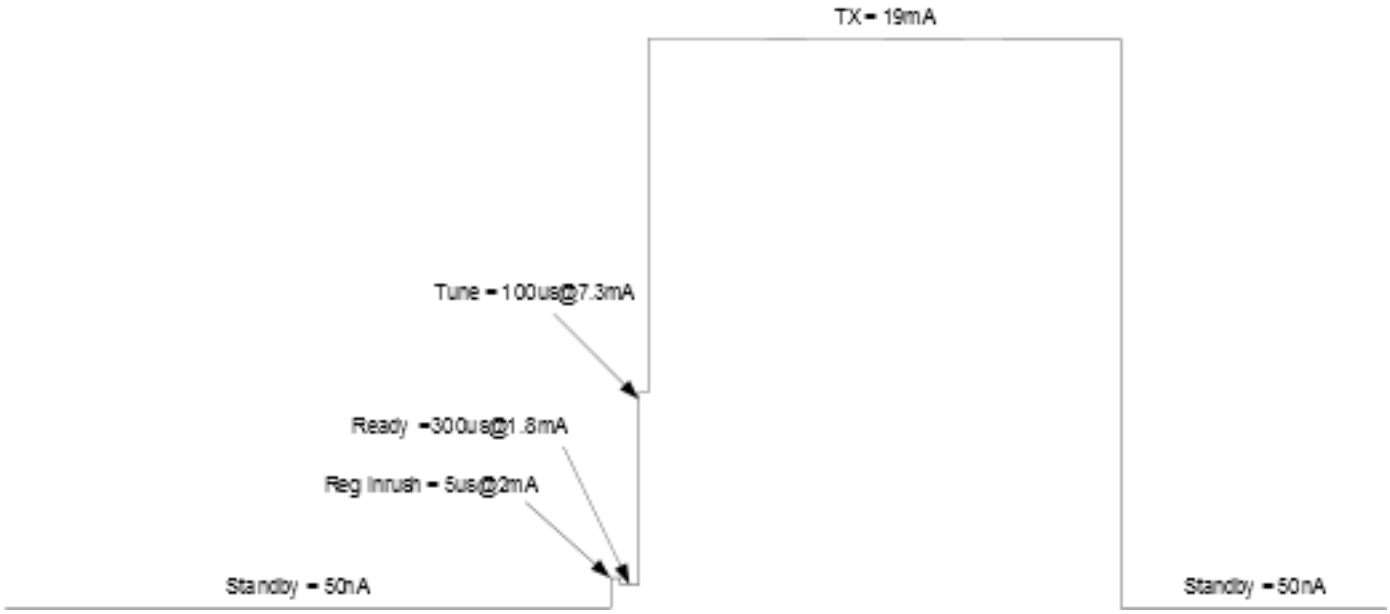


Figure 4.30. Start-Up Timing and Current Consumption using Standby State



4.2.4.2.1 Shutdown State

The shutdown state is the lowest current consumption state of the device and is entered by driving SDN (Pin 2) high. In this state, all register contents are lost and there is no SPI access. To exit this mode, drive SDN low. The device will then initiate a power on reset (POR) along with internal calibrations. Once this POR period is complete, the POWER_UP command is required to initialize the radio and the configuration can then be loaded into the device. The SDN pin must be held high for at least 10 µs before driving it low again to insure the POR can be executed correctly. The shutdown state can be used to fully reset the part. If POR timing and voltage requirements cannot be met, it is highly recommended that SDN be controlled using the host processor rather than tying it to GND on the board.

4.2.4.2.2 Standby State

The standby state has similar current consumption to the shutdown state but retains all register values, allowing for a much faster response time. Because of these benefits, most applications will want to use standby mode rather than shutdown. The standby state is entered by using the CHANGE_STATE API

command. While in this state, the SPI is accessible but any SPI event will automatically transition the chip to the SPI active state. After the SPI event, the host will need to re-command the device to standby mode.

4.2.4.2.3 SPI Active State

The SPI active state enables the device to process any SPI events, such as API commands. In this state, the SPI and boot up oscillator are enabled. The SPI active state is entered by using the CHANGE_STATE command or automatically through an SPI event while in standby mode. If the SPI active state was entered automatically from standby mode, a CHANGE_STATE command will be needed to return the device to standby mode.

4.2.4.2.4 Power on Reset

A Power On Reset (POR) sequence is used to boot the device up from a fully off or shutdown state. To execute this process, VDD must ramp within 1 ms and must remain applied to the device for at least 10 ms. If VDD is removed, then it must stay below 0.15 V for at least 10 ms before being applied again. Refer to Figure 4.31 (p. 50) and Table 4.15 (p. 50) for details.

Figure 4.31. POR Timing Diagram

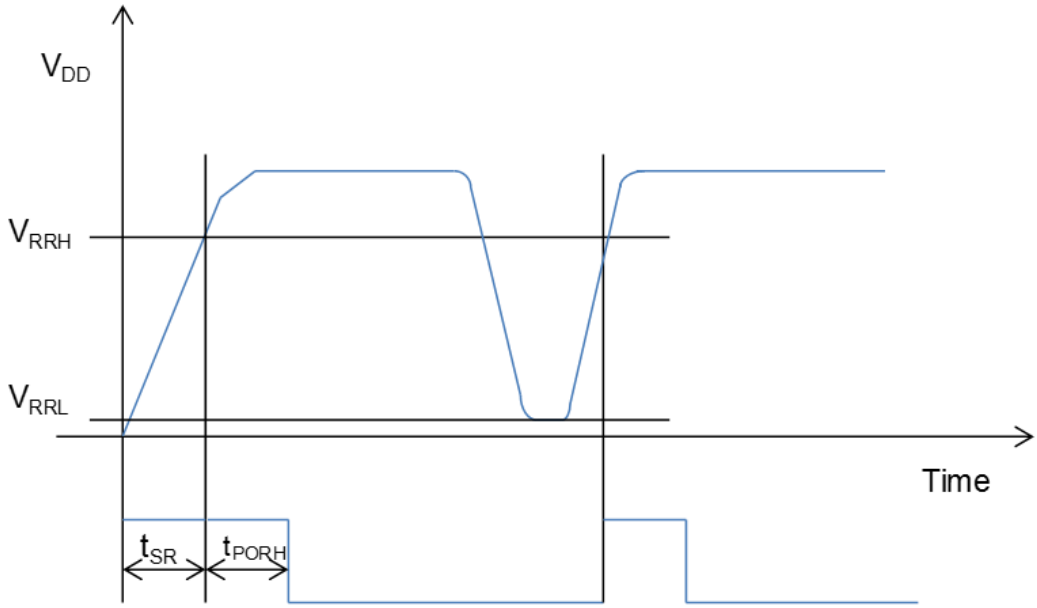


Table 4.15. POR Timing

Variable	Description	Min	Typ	Max	Units
tPORH		10			ms
tPORL		10			ms
VRRH		90%*Vdd			V
VRRL		0		150	mV
tSR				1	ms

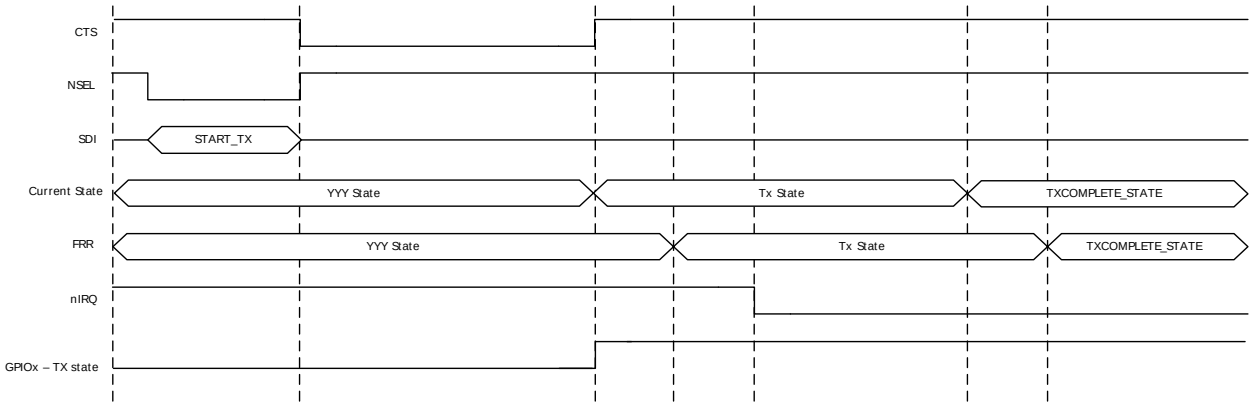
4.2.4.2.5 TX State

The TX state is used whenever the device is required to transmit data. It is entered using either the START_TX or CHANGE_STATE command. With the START_TX command, the next state can be defined to insure optimal timing. When either command is sent to enter TX state, an internal sequencer automatically takes care of all actions required to move between states with no additional user commands needed. Examples of the timing of this transition can be seen in Figure 4.29 (p. 49)

and Figure 4.30 (p. 49). The specific sequencer controlled events that take place during this time can include enable internal LDOs, start up crystal oscillator, enable PLL, calibrate VCO/PLL, active power amplifier, and transmit packet.

Figure 4.32 (p. 51) shows an example of the commands and timing for the START_TX command. CTS will go high as soon as the sequencer puts the part into TX state. As the sequencer is stepping through the events listed above, CTS will be low and no new commands or property changes are allowed. If the nIRQ is used to monitor the current state, there will be a slight delay caused by the internal hardware from when the event actually occurs to when the transition occurs on the nIRQ. The time from entering TX state to when the nIRQ will transition is 13 µs. If a GPIO is programmed for TX state or used as control for a transmit/receive switch (TR switch), there is no delay.

Figure 4.32. START_TX Commands and Timing



4.2.4.2.6 RX State

The RX state is used whenever the device is required to receive data. It is entered using either the START_RX or CHANGE_STATE commands. With the START_RX command, the next state can be defined to insure optimal timing. When either command is sent to enter RX state, an internal sequencer automatically takes care of all actions required to move between states with no additional user commands needed. The sequencer controlled events can include enable the digital and analog LDOs, start up the crystal oscillator, enable PLL, calibrate VCO, enable receiver circuits, and enable receive mode. The device will also automatically set up all receiver features such as packet handling based upon the initial configuration of the device.

4.2.4.3 Interrupts

The EZRadio is capable of generating an interrupt signal when certain events occur. The chip notifies the microcontroller that an interrupt event has occurred by setting the nIRQ output pin LOW = 0. This interrupt signal will be generated when any one (or more) of the interrupt events occur. The nIRQ pin will remain low until the microcontroller reads the Interrupt Status Registers. The nIRQ output signal will then be reset until the next change in status is detected.

The interrupt sources are grouped into three categories: packet handler, chip status, and modem. The individual interrupts in these groups can be enabled/disabled in the interrupt property registers, 0x0101, 0x0102, and 0x0103. An interrupt must be enabled for it to trigger an event on the nIRQ pin. The interrupt group must be enabled as well as the individual interrupts in API property 0x0100.

- Chip status
- Modem status
- Packet handler status
- Packet sent
- Packet received

- CRC error
- Invalid preamble detected
- Invalid sync detected
- Preamble detected
- Sync detected
- State change
- Command error
- Chip ready
- TX FIFO almost empty
- RX FIFO almost full
- RSSI interrupt

4.2.4.4 GPIO

Four General Purpose IO (GPIO) pins are available for use in the application. The GPIOs are configured using the GPIO_PIN_CFG command. GPIO pins 0 and 1 should be used for active signals such as data or clock. GPIO pins 2 and 3 have more susceptibility to generating spurious components in the synthesizer than pins 0 and 1. The drive strength of the GPIO's can be adjusted with the GEN_CONFIG parameter in the GPIO_PIN_CFG command. By default, the drive strength is set to the minimum. The default configuration and the state of the GPIO during shutdown are shown in Table 4.16 (p. 52). For a complete list of the GPIO options, please refer to the EZRadio API documentation zip file available from www.silabs.com.

Table 4.16. GPIOs

Pin	SDN State	POR Default
GPIO0	0	POR
GPIO1	0	CTS
GPIO2	0	POR
GPIO3	0	POR
nIRQ	Resistive VDD pull-up	nIRQ
SDO	Resistive VDD pull-up	SDO
SDI	High Z	SDI
SCLK	High Z	SCLK
NSEL	High Z	NSEL

4.2.5 Data Handling and Packet Handler

4.2.5.1 RX and TX FIFOs

Two 64-byte FIFOs are integrated into the chip, one for RX and one for TX. Writing to command register 66h loads data into the TX FIFO and reading from command register 77h reads data from the RX FIFO. For packet lengths greater than 64 bytes, RX_FIFO_ALMOST_FULL and TX_FIFO_ALMOST_EMPTY status bits and interrupts can be used to manage the FIFO. The maximum payload length supported in packet handler mode is 255 bytes.

The EZRadio includes integrated packet handler features such as preamble and sync word detection as well as CRC calculation. This allows the chip to qualify and synchronize with legitimate transmissions independent of the microcontroller. In this setup, the preamble and sync word length can be modified and the sync word pattern can be selected. If the preamble is greater than or equal to 4 bytes, the device uses the preamble detection circuit with a 2-byte detection threshold. If the preamble is less than 32 bits, then at least two bytes of sync word are required plus at least one byte of 0101 pattern (3 bytes total).

In this case, preamble detection is skipped, and only sync word detection is used. For any combination of preamble and sync word less than three bytes, the device will use direct mode. The general packet structure is shown in Table 4.17 (p. 53) .

The EZConfig setup also provides the option to select a variable packet length. With this setting, the receiver is not required to know the packet length ahead of time. The transmitter sends the length of the packet immediately after the sync word. The packet structure for variable length packets is shown in Table 4.16 (p. 52) .

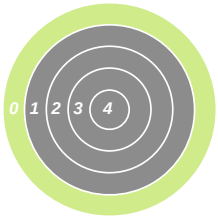
Table 4.17. Packet Structure for Fixed Packet Length

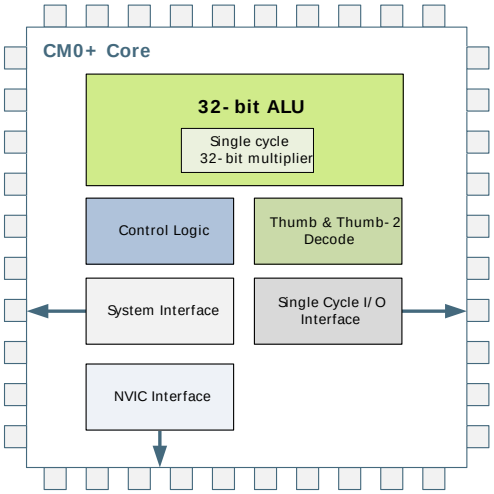
Preamble	Sync Word	Data	CRC
0 - 255 Bytes	1 - 4 Bytes	1 - 255 Bytes	2 Bytes

4.2.5.2 Direct Mode

In direct mode, the packet handler (including FIFO) is bypassed, and the host MCU must feed the data stream to the device in TX mode and read out the data stream in RX mode via GPIOs. The host MCU will process the data and perform packet handler functions. This is commonly used to support legacy implementations where host MCU software exists or to support non-standard packet structures. Some examples are packets with non 1010 preamble pattern, no preamble or sync word, or sync word with no edge transitions.

5 System Processor





Quick Facts

What?

The industry leading Cortex-M0+ processor from ARM is the CPU in the EZR32HG wireless microcontrollers.

Why?

The ARM Cortex-M0+ is designed for exceptional short response time, high code density, and high 32-bit throughput while maintaining a strict cost and power consumption budget.

How?

Combined with the ultra low energy peripherals available, the Cortex-M0+ makes the EZR32HG devices perfect for 8- to 32-bit applications. The processor is featuring a 2 stage pipeline, dedicated single cycle I/O interface, efficient single cycle instructions, Thumb/Thumb-2 instruction set support, and fast interrupt handling.

5.1 Introduction

The ARM Cortex-M0+ 32-bit RISC processor provides outstanding computational performance and exceptional system response to interrupts while meeting low cost requirements and low power consumption.

The ARM Cortex-M0+ implemented is revision r0p1.

5.2 Features

- 2-stage pipeline
- Thumb/Thumb-2 instruction subset
  - Enhanced levels of performance, energy efficiency, and code density
  - Enables direct portability to other ARM Cortex-M processors
- Hardware single-cycle multiplication
  - Enables 32-bit multiplication in a single cycle
- Dedicated Single-cycle I/O interface
  - Provides immediate acces to all GPIO-registers
  - Enables the processor to simultanously fetch the next instructions over the System bus
- Configurable IRQ-latency
  - Allows developers to select a trade-off between interrupt response time and predictability
- Up to 1.08 DMIPS/MHz
- 24-bit System Tick Timer for Real-Time Operating System (RTOS)
- Excellent 32-bit migration choice for 8/16 bit architecture based designs
  - Simplified stack-based programmer's model is compatible with traditional ARM architecture and retains the programming simplicity of legacy 8- and 16-bit architectures
- Integrated power modes

- Sleep Now mode for immediate transfer to low power state
- Sleep on Exit mode for entry into low power state after the servicing of an interrupt
- Ability to extend power savings to other system components

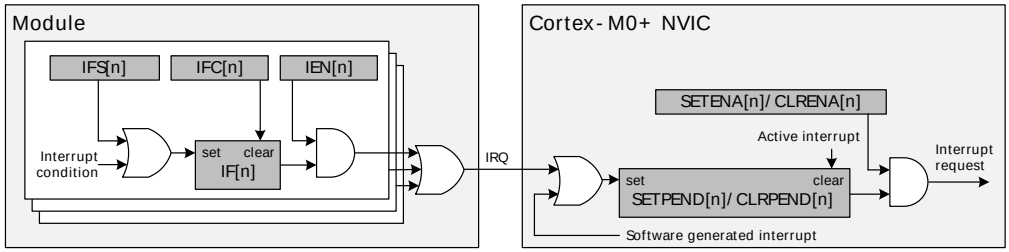
- Optimized for low latency, nested interrupts

5.3 Functional Description

For a full functional description of the ARM Cortex-M0+ (r0p1) implementation in the EZR32HG family, the reader is referred to the *ARM Cortex-M0+ Devices Generic User Guide*.

5.3.1 Interrupt Operation

Figure 5.1. Interrupt Operation



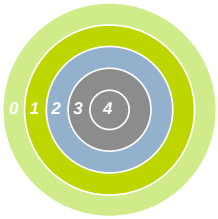
The EZR32HG devices have up to interrupt request lines (IRQ) which are connected to the Cortex-M0+. Each of these lines (shown in Table 5.1 (p. 55)) are connected to one or more interrupt flags in one or more modules. The interrupt flags are set by hardware on an interrupt condition. It is also possible to set/clear the interrupt flags through the IFS/IFC registers. Each interrupt flag is then qualified with its own interrupt enable bit (IEN register), before being OR'ed with the other interrupt flags to generate the IRQ. A high IRQ line will set the corresponding pending bit (can also be set/cleared with the SETPEND/CLRPEND bits in ISPR0/ICPR0) in the Cortex-M0+ NVIC. The pending bit is then qualified with an enable bit (set/cleared with SETENA/CLRENA bits in ISER0/ICER0) before generating an interrupt request to the core. Figure 5.1 (p. 55) illustrates the interrupt system. For more information on how the interrupts are handled inside the Cortex-M0+, the reader is referred to the *ARM Cortex-M0+ Devices Generic User Guide*.

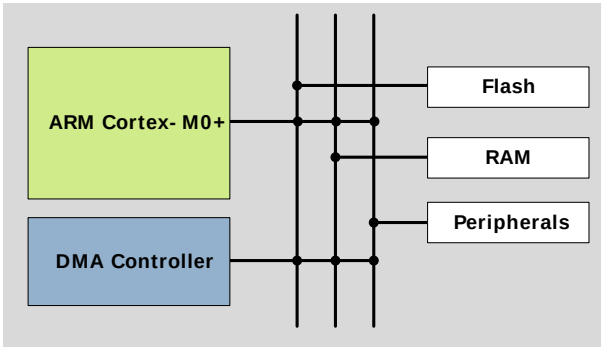
Table 5.1. Interrupt Request Lines (IRQ)

IRQ #	Source
0	DMA
1	GPIO_EVEN
2	TIMER0
3	ACMP0
4	ADC0
5	I2C0
6	GPIO_ODD
7	TIMER1
8	USART1_RX
9	USART1_TX
10	LEUART0
11	PCNT0

IRQ #	Source
12	RTC
13	CMU
14	VCMP
15	MSC
16	AES
17	USART0_RX
18	USART0_TX
19	USB
20	TIMER2

6 Memory and Bus System





Quick Facts

What?

A low latency memory system, including low energy flash and RAM with data retention, makes extended use of low-power energy-modes possible.

Why?

RAM retention reduces the need for storing data in flash and enables frequent use of the ultra low energy modes EM2 and EM3 with as little as 0.5 µA current consumption.

How?

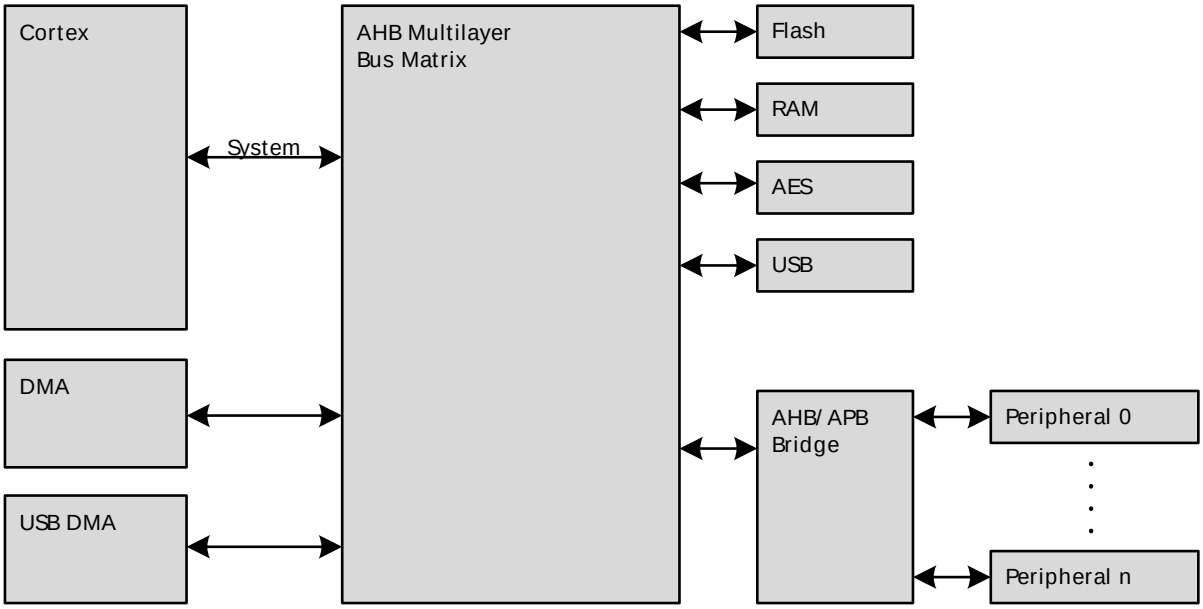
Low energy and non-volatile flash memory stores program and application data in all energy modes and can easily be reprogrammed in system. Low leakage RAM, with data retention in EM0 to EM3, removes the data restore time penalty, and the DMA ensures fast autonomous transfers with predictable response time.

6.1 Introduction

The EZR32HG contains an AMBA AHB Bus system allowing bus masters to access the memory mapped address space. A multilayer AHB bus matrix, using a Round-robin arbitration scheme, connects the master bus interfaces to the AHB slaves (Figure 6.1 (p. 58)). The bus matrix allows several AHB slaves to be accessed simultaneously. An AMBA APB interface is used for the peripherals, which are accessed through an AHB-to-APB bridge connected to the AHB bus matrix. The AHB bus masters are:

- **Cortex-M0+ System:** Used for instruction fetches, data and debug access (0x00000000 - 0xDFFFFFFF).
- **DMA:** Can access SRAM, Flash and peripherals (0x00000000 - 0xDFFFFFFF), except GPIO (0x40006000 - 0x40007000).
- **USB DMA:** Can access SRAM and Flash (0x00000000 - 0x3FFFFFFF), and the AHB-peripherals: USB and AES.

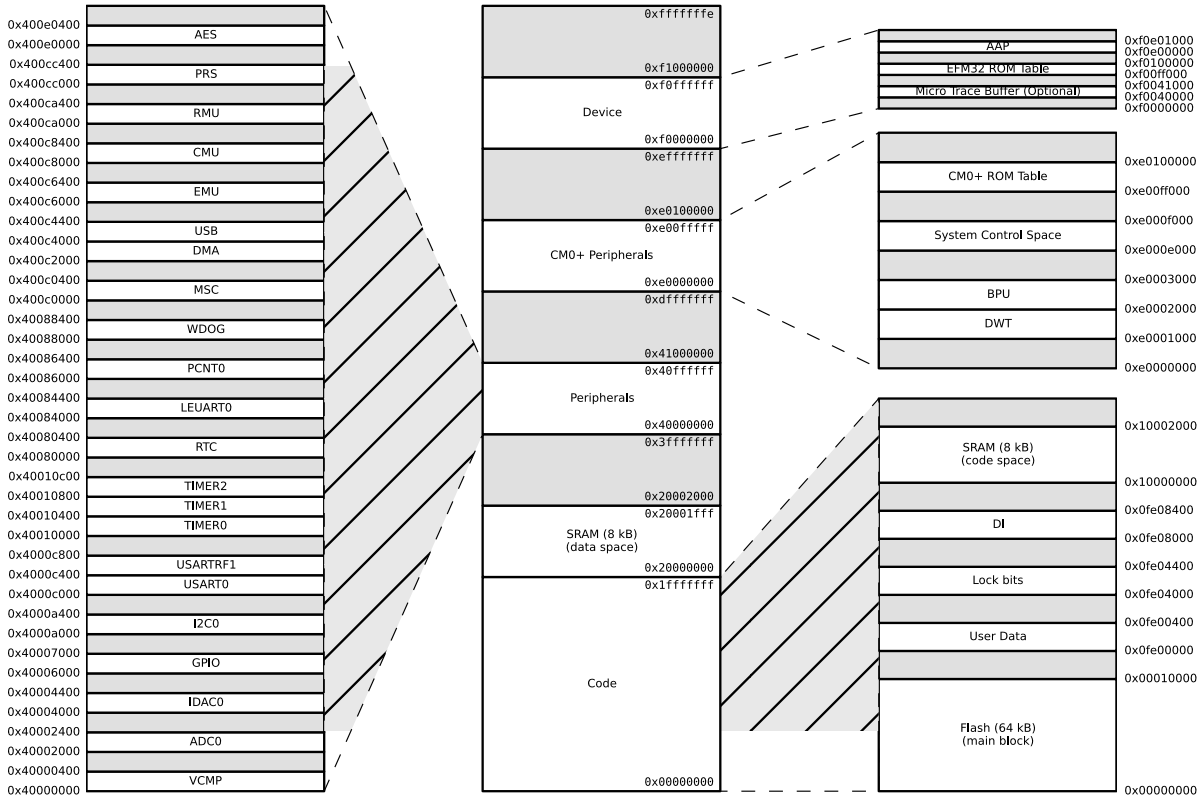
Figure 6.1. EZR32HG Bus System



6.2 Functional Description

The memory segments are mapped together with the internal segments of the Cortex-M0+ into the system memory map shown by Figure 6.2 (p. 59)

Figure 6.2. System Address Space



The embedded SRAM is located at address 0x20000000 in the memory map of the EZR32HG. It is also mapped in code space at address 0x10000000 to keep compatibility towards Cortex-M3 and Cortex-M4 EZR32-devices, that uses this code-space mapped SRAM for faster instruction fetching.

6.2.1 Peripherals

The peripherals are mapped into the peripheral memory segment, each with a fixed size address range according to Table 6.1 (p. 59) , Table 6.2 (p. 60) and Table 6.3 (p. 60) .

Table 6.1. Memory System Core Peripherals

Core peripherals	
Address Range	**Module Name**
0xF0040000 - 0xF007FFFF	MTB
0x400E0000 - 0x400E03FF	AES
0x400CA000 - 0x400CA3FF	RMU
0x400C8000 - 0x400C83FF	CMU
0x400C6000 - 0x400C63FF	EMU
0x400C4000 - 0x400C43FF	USB
0x400C2000 - 0x400C3FFF	DMA
0x400C0000 - 0x400C03FF	MSC

Table 6.2. Memory System Low Energy Peripherals

Low Energy peripherals	
Address Range	**Module Name**
0x40088000 - 0x400883FF	WDOG
0x40086000 - 0x400863FF	PCNT0
0x40084000 - 0x400843FF	LEUART0
0x40080000 - 0x400803FF	RTC

Table 6.3. Memory System Peripherals

Peripherals	
Address Range	**Module Name**
0x400CC000 - 0x400CC3FF	PRS
0x40010800 - 0x40010BFF	TIMER2
0x40010400 - 0x400107FF	TIMER1
0x40010000 - 0x400103FF	TIMER0
0x4000C400 - 0x4000C7FF	USARTRF1
0x4000C000 - 0x4000C3FF	USART0
0x4000A000 - 0x4000A3FF	I2C0
0x40006000 - 0x40006FFF	GPIO
0x40004000 - 0x400043FF	IDAC0
0x40002000 - 0x400023FF	ADC0
0x40000000 - 0x400003FF	VCMP

6.2.2 Bus Matrix

The Bus Matrix connects the memory segments to the bus masters:

- Code: CPU instruction or data fetches from the code space
- System: CPU read and write to the SRAM and peripherals
- DMA: Access to SRAM, Flash and peripherals
- USB DMA: Access to SRAM and Flash

6.2.2.1 Arbitration

The Bus Matrix uses a round-robin arbitration algorithm which enables high throughput and low latency while starvation of simultaneous accesses to the same bus slave are eliminated. Round-robin does not assign a fixed priority to each bus master. The arbiter does not insert any bus wait-states.

6.2.2.2 Access Performance

The Bus Matrix is a multi-layer energy optimized AMBA AHB compliant bus with an internal bandwidth equal to 4 times a single AHB-bus.

The Bus Matrix accepts new transfers initiated by each master in every clock cycle without inserting any wait-states. The slaves, however, may insert wait-states depending on their internal throughput and the clock frequency.

The Cortex-M0+, the DMA Controller, and the peripherals run on clocks that can be prescaled separately. When accessing a peripheral which runs on a frequency equal to or faster than the HFCORECLK, the number of wait cycles per access, in addition to master arbitration, is given by:

Memory Wait Cycles with Clock Equal or Faster than HFCORECLK

$$N_{cycles} = 2 + N_{slave\ cycles}, \quad (6.1)$$

where $N_{slave\ cycles}$ is the wait cycles introduced by the slave.

When accessing a peripheral running on a clock slower than the HFCORECLK, wait-cycles are introduced to allow the transfer to complete on the peripheral clock. The number of wait cycles per access, in addition to master arbitration, is given by:

Memory Wait Cycles with Clock Slower than CPU

$$N_{cycles} = (2 + N_{slave\ cycles}) \times f_{HFCORECLK}/f_{HFPERCLK}, \quad (6.2)$$

where $N_{slave\ cycles}$ is the number of wait cycles introduced by the slave.

For general register access, $N_{slave\ cycles} = 1$.

More details on clocks and prescaling can be found in Chapter 12 (p. 139) .

6.3 Access to Low Energy Peripherals (Asynchronous Registers)

6.3.1 Introduction

The Low Energy Peripherals are capable of running when the high frequency oscillator and core system is powered off, i.e. in energy mode EM2 and in some cases also EM3. This enables the peripherals to perform tasks while the system energy consumption is minimal.

The Low Energy Peripherals are:

- Low Energy UART - LEUART
- Pulse Counter - PCNT
- Real Time Counter - RTC
- Watchdog - WDOG

All Low Energy Peripherals are memory mapped, with automatic data synchronization. Because the Low Energy Peripherals are running on clocks asynchronous to the core clock, there are some constraints on how register accesses can be done, as described in the following sections.

6.3.1.1 Writing

Every Low Energy Peripheral has one or more registers with data that needs to be synchronized into the Low Energy clock domain to maintain data consistency and predictable operation. There are two different synchronization mechanisms on the Happy Gecko; immediate synchronization, and delayed synchronization. Immediate synchronization is available for the RTC and results in an immediate update of the target registers. Delayed synchronization is used for the other Low Energy Peripherals, and for these peripherals, a write operation requires 3 positive edges on the clock of the Low Energy Peripheral being accessed. Registers requiring synchronization are marked "Asynchronous" in their description header.

6.3.1.1.1 Delayed synchronization

After writing data to a register which value is to be synchronized into the Low Energy Peripheral using delayed synchronization, a corresponding busy flag in the <module_name>_SYNCBUSY register (e.g.

LEUART_SYNCBUSY) is set. This flag is set as long as synchronization is in progress and is cleared upon completion.

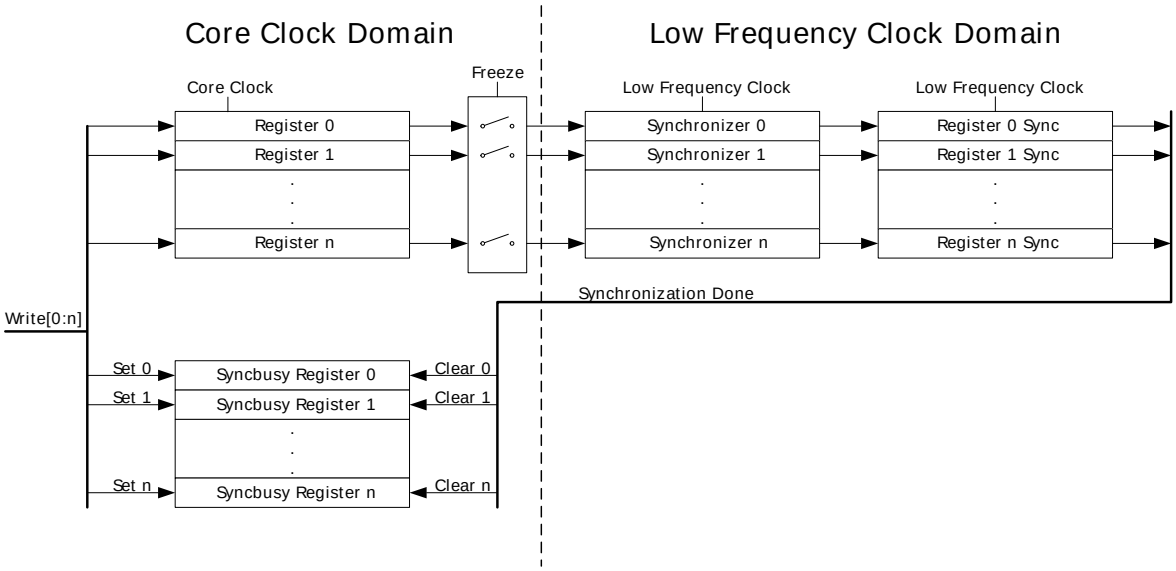
Note

Subsequent writes to the same register before the corresponding busy flag is cleared is not supported. Write before the busy flag is cleared may result in undefined behavior.

In general, the SYNCBUSY register only needs to be observed if there is a risk of multiple write access to a register (which must be prevented). It is not required to wait until the relevant flag in the SYNCBUSY register is cleared after writing a register. E.g EM2 can be entered immediately after writing a register.

See Figure 6.3 (p. 62) for a more detailed overview of the write operation.

Figure 6.3. Write operation to Low Energy Peripherals



6.3.1.1.2 Immediate synchronization

Contrary to the peripherals with delayed synchronization, data written to peripherals with immediate synchronization, takes effect in the peripheral immediately. They are updated immediately on the peripheral write access. If a write is set up close to a peripheral clock edge, the write is delayed to after the clock edge. This will introduce wait-states on peripheral access. In the worst case, there can be three wait-state cycles of the HFCORECLK_LE and an additional wait-state equivalent of up to 315 ns.

For peripherals with immediate synchronization, the SYNCBUSY registers are still present and serve two purposes: (1) commands written to a peripheral with immediate synchronization are not executed before the first peripheral clock after the write. During this period, the SYNCBUSY flag in the command register is set, indicating that the command has not yet been executed; (2) to maintain backwards compatibility with the EFM32G series, SYNCBUSY registers are also present for other registers. These are however, always 0, indicating that register writes are always safe.

Note

If the application must be compatible with the EFM32G series, all Low Energy Peripherals should be accessed as if they only had delayed synchronization, i.e. using SYNCBUSY.

6.3.1.2 Reading

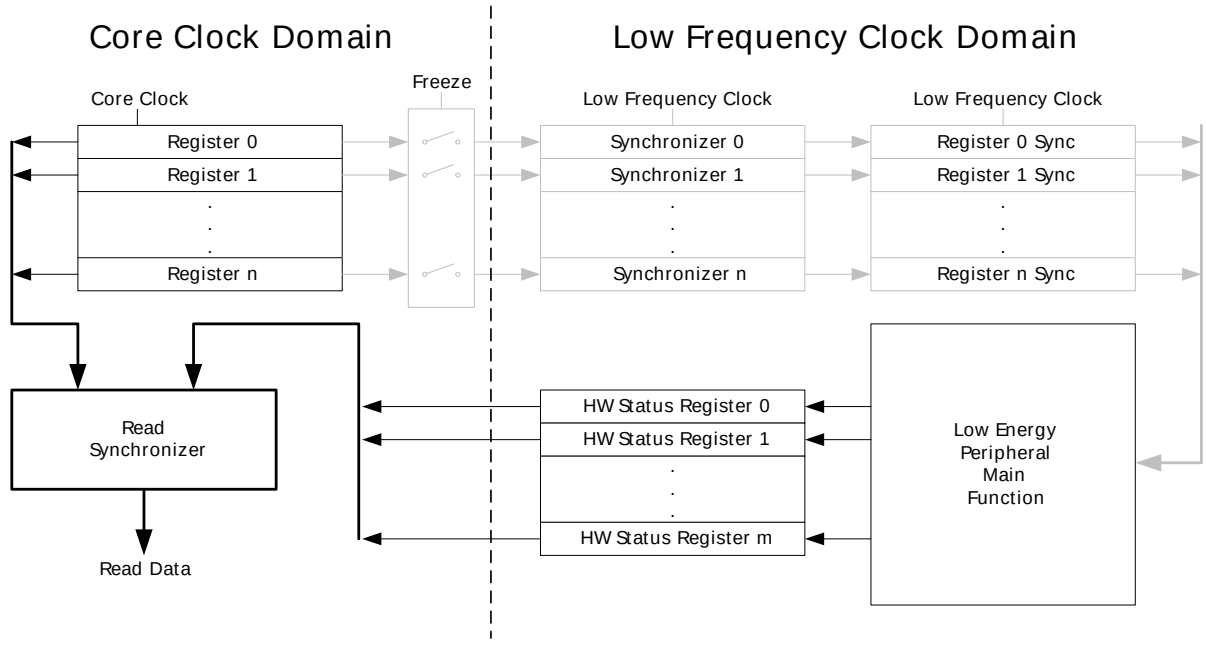
When reading from Low Energy Peripherals, the data is synchronized regardless of the originating clock domain. Registers updated/maintained by the Low Energy Peripheral are read directly from the Low

Energy clock domain. Registers residing in the core clock domain, are read from the core clock domain. See Figure 6.4 (p. 63) for a more detailed overview of the read operation.

Note

Writing a register and then immediately reading back the value of the register may give the impression that the write operation is complete. This is not necessarily the case. Please refer to the SYNCBUSY register for correct status of the write operation to the Low Energy Peripheral.

Figure 6.4. Read operation from Low Energy Peripherals



6.3.2 FREEZE register

For Low Energy Peripherals with delayed synchronization there is a <module_name>_FREEZE register (e.g. RTC_FREEZE), containing a bit named REGFREEZE. If precise control of the synchronization process is required, this bit may be utilized. When REGFREEZE is set, the synchronization process is halted, allowing the software to write multiple Low Energy registers before starting the synchronization process, thus providing precise control of the module update process. The synchronization process is started by clearing the REGFREEZE bit.

Note

The FREEZE register is also present on peripherals with immediate synchronization, but has no effect.

6.4 Flash

The Flash retains data in any state and typically stores the application code, special user data and security information. The Flash memory is typically programmed through the debug interface, but can also be erased and written to from software.

- Up to 64 kB of memory
- Page size of 1024 bytes (minimum erase unit)
- Minimum 20 000 erase cycles
- More than 10 years data retention at 85°C
- Lock-bits for memory protection

- Data retention in any state

6.5 SRAM

The primary task of the SRAM memory is to store application data. Additionally, it is possible to execute instructions from SRAM, and the DMA may used to transfer data between the SRAM, Flash and peripherals.

- Up to 8 kB memory
- Data retention of the entire memory in EM0 to EM3

6.6 Device Information (DI) Page

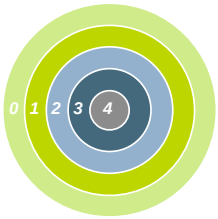
The DI page contains calibration values, a unique identification number and other useful data. See the table below for a complete overview.

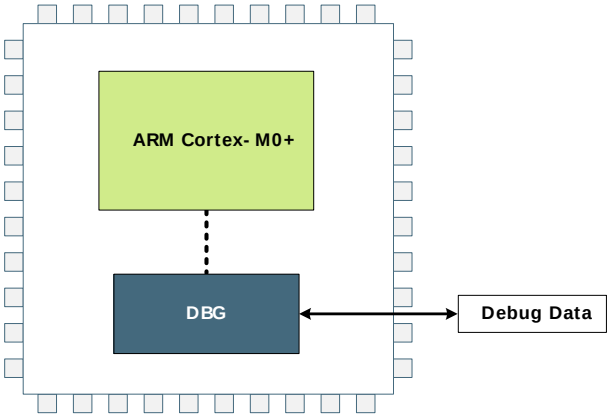
Table 6.4. Device Information Page Contents

DI Address	Register	Description
0x0FE08020	CMU_LFRCOCTRL	Register reset value.
0x0FE08028	CMU_HFRCOCTRL	Register reset value.
0x0FE08030	CMU_AUXHFRCOCTRL	Register reset value.
0x0FE08040	ADC0_CAL	Register reset value.
0x0FE08048	ADC0_BIASPROG	Register reset value.
0x0FE08058	CMU_LCDCTRL	Register reset value.
0x0FE08078	IDAC0_CAL	Register reset value.
0x0FE08098	CMU_USHFRCOCTRL	Register reset value.
0x0FE081AA	DI_CRC	[15:0]: DI data CRC-16.
0x0FE081AC	MCM_REV_MIN	MCM minor revision (0, 1, 2, …).
0x0FE081AD	MCM_REV_MAJ	MCM major revision (A=1, B=2, C=3, …).
0x0FE081AE	RADIO_REV_MIN	Radio minor revision. (0, 1, 2, ...).
0x0FE081AF	RADIO_REV_MAJ	Radio major revision. (A = 1, B = 2, C = 3, …).
0x0FE081B0	RADIO_OPN	4 digit part number for the radio die. Example: EZR32WG330F256R67 => 4467d.
0x0FE081B2	CAL_TEMP_0	[7:0] Calibration temperature (Â°C).
0x0FE081B4	ADC0_CAL_1V25	[14:8]: Gain for 1V25 reference, [6:0]: Offset for 1V25 reference.
0x0FE081B6	ADC0_CAL_2V5	[14:8]: Gain for 2V5 reference, [6:0]: Offset for 2V5 reference.
0x0FE081B8	ADC0_CAL_VDD	[14:8]: Gain for VDD reference, [6:0]: Offset for VDD reference.
0x0FE081BA	ADC0_CAL_5VDIFF	[14:8]: Gain for 5VDIFF reference, [6:0]: Offset for 5VDIFF reference.
0x0FE081BC	ADC0_CAL_2XVDD	[14:8]: Reserved (gain for this reference cannot be calibrated), [6:0]: Offset for 2XVDD reference.
0x0FE081BE	ADC0_TEMP_0_READ_1V25	[15:4] Temperature reading at 1V25 reference, [3:0]: Reserved.

DI Address	Register	Description
0x0FE081C8	IDAC0_CAL_RANGE0	[7:0]: Current range 0 tuning.
0x0FE081C9	IDAC0_CAL_RANGE1	[7:0]: Current range 1 tuning.
0x0FE081CA	IDAC0_CAL_RANGE2	[7:0]: Current range 2 tuning.
0x0FE081CB	IDAC0_CAL_RANGE3	[7:0]: Current range 3 tuning.
0x0FE081CC	USHFRCO_COARSECAL_BAND_25	[6:0]: Coarse tuning for the 24 MHz USHFRCO band.
0x0FE081CD	USHFRCO_FINECAL_BAND_25	[5:0]: Fine tuning for the 24 MHz USHFRCO band.
0x0FE081CE	USHFRCO_COARSECAL_BAND_48	[6:0]: Coarse tuning for the 48 MHz USHFRCO band.
0x0FE081CF	USHFRCO_FINECAL_BAND_48	[5:0]: Fine tuning for the 48 MHz USHFRCO band.
0x0FE081D4	AUXHFRCO_CALIB_BAND_1	[7:0]: Tuning for the 1.2 MHz AUXHFRCO band.
0x0FE081D5	AUXHFRCO_CALIB_BAND_7	[7:0]: Tuning for the 6.6 MHz AUXHFRCO band.
0x0FE081D6	AUXHFRCO_CALIB_BAND_11	[7:0]: Tuning for the 11 MHz AUXHFRCO band.
0x0FE081D7	AUXHFRCO_CALIB_BAND_14	[7:0]: Tuning for the 14 MHz AUXHFRCO band.
0x0FE081D8	AUXHFRCO_CALIB_BAND_21	[7:0]: Tuning for the 21 MHz AUXHFRCO band.
0x0FE081DC	HFRCO_CALIB_BAND_1	[7:0]: Tuning for the 1.2 MHz HFRCO band.
0x0FE081DD	HFRCO_CALIB_BAND_7	[7:0]: Tuning for the 6.6 MHz HFRCO band.
0x0FE081DE	HFRCO_CALIB_BAND_11	[7:0]: Tuning for the 11 MHz HFRCO band.
0x0FE081DF	HFRCO_CALIB_BAND_14	[7:0]: Tuning for the 14 MHz HFRCO band.
0x0FE081E0	HFRCO_CALIB_BAND_21	[7:0]: Tuning for the 21 MHz HFRCO band.
0x0FE081EE	RADIO_ID	RADIO_ID[15:0]: EZR32WG/EZR32LG/EZR32HG = 1d.
0x0FE081F0	UNIQUE_0	[31:0] Unique number.
0x0FE081F4	UNIQUE_1	[63:32] Unique number.
0x0FE081F8	MEM_INFO_FLASH	[15:0]: Flash size, kbyte count as unsigned integer (e.g. 128).
0x0FE081FA	MEM_INFO_RAM	[15:0]: Ram size, kbyte count as unsigned integer (e.g. 16).
0x0FE081FC	PART_NUMBER	[15:0]: EZR32 part number as unsigned integer (e.g. 230).
0x0FE081FE	PART_FAMILY	[7:0]: EZR32 part family number (Gecko = 71, Giant Gecko = 72, Tiny Gecko = 73, Leopard Gecko=74, Wonder Gecko=75, Zero Gecko=76, Happy Gecko=77, EZR32WG=120, EZR32LG=121, EZR32HG=122).
0x0FE081FF	PROD_REV	[7:0]: EZR32 Production ID.

7 DBG - Debug Interface





Quick Facts

What?

The DBG (Debug Interface) is used to program and debug EZR32HG devices.

Why?

The Debug Interface makes it easy to re-program and update the system in the field, and allows debugging with minimal I/O pin usage.

How?

The Cortex-M0+ supports advanced debugging features. EZR32HG devices only use two port pins for debugging or programming. The internal and external state of the system can be examined with debug extensions supporting instruction or data access break- and watch points.

7.1 Introduction

The EZR32HG devices include hardware debug support through a 2-pin serial-wire debug (SWD) interface.

For more technical information about the debug interface the reader is referred to:

- ARM Cortex-M0+ Technical Reference Manual
- ARM CoreSight Components Technical Reference Manual
- ARM Debug Interface v5 Architecture Specification

7.2 Features

- Flash Patch and Breakpoint (FPB) unit
  - Implement breakpoints and code patches
- Data Watch point and Trace (DWT) unit
  - Implement watch points, trigger resources and system profiling

7.3 Functional Description

There are two debug pins available on the device. Their operation is described in the following section.

7.3.1 Debug Pins

The following pins are the debug connections for the device:

- Serial Wire Clock input (SWCLK): This pin is enabled after reset and has a built-in pull down.
- Serial Wire Data Input/Output (SWDIO): This pin is enabled after reset and has a built-in pull-up.

The debug pins can be enabled and disabled through GPIO_ROUTE, see Section 26.3.4.1 (p. 560) . Please remeberer that upon disabling, debug contact with the device is lost. Also note that, because the debug pins have pull-down and pull-up enabled by default, leaving them enabled might increase the current consumption with up to 200 µA if left connected to supply or ground.

7.3.2 Debug and EM2/EM3

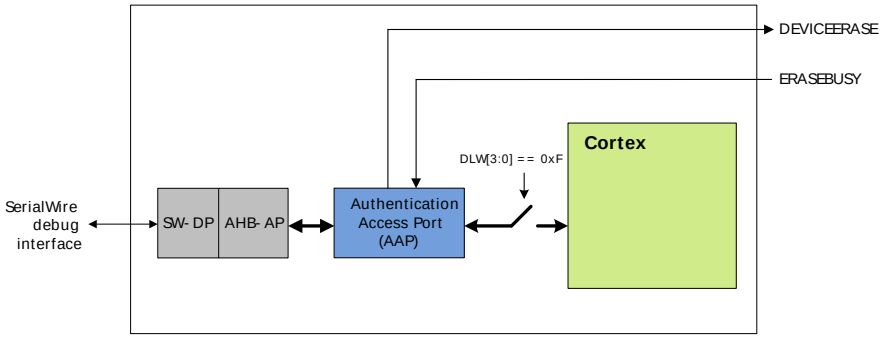
Leaving the debugger connected when issuing a WFI or WFE to enter EM2 or EM3 will make the system enter a special EM2. This mode differs from regular EM2 and EM3 in that the high frequency clocks are still enabled, and certain core functionality is still powered in order to maintain debug-functionality. Because of this, the current consumption in this mode is closer to EM1 and it is therefore important to disconnect the debugger before doing current consumption measurements.

7.4 Debug Lock and Device Erase

The debug access to the Cortex-M0+ is locked by clearing the Debug Lock Word (DLW) and resetting the device, see Section 8.3.2 (p. 73) .

When debug access is locked, the debug interface remains accessible but the connection to the Cortex-M0+ core and the whole bus-system is blocked as shown in Figure 7.2 (p. 68). This mechanism is controlled by the Authentication Access Port (AAP) as illustrated by Figure 7.1 (p. 67). The AAP is only accessible from a debugger and not from the core.

Figure 7.1. AAP - Authentication Access Port



As seen from Figure 7.1 (p. 67), the AAP is situated after the AHB-AP, meaning it should be accessed like any other peripheral from the debug. The address of the AAP is 0xF0E00000 as can also be seen from Figure 6.2 (p. 59) .

Note

This is different from some other EZR32 devices, where the AAP is integrated as a separate AP (Access Port), please see the reference manual of the respective devices.

The debugger can access the AAP-registers, and only these registers just after reset, for the time of the AAP-window outlined in Figure 7.2 (p. 68). If the device is locked, access to the core and bus-system is blocked even after code execution starts, and the debugger can only access the AAP-registers. If the device is not locked, the AAP is no longer accessible after code execution starts, and the debugger can access the core and bus-system normally. The AAP window can be extended by issuing the bit pattern on SWDIO/SWCLK as shown in Figure 7.3 (p. 68). This pattern should be applied just before reset is deasserted, and will give the debugger more time to access the AAP.

Figure 7.2. Device Unlock

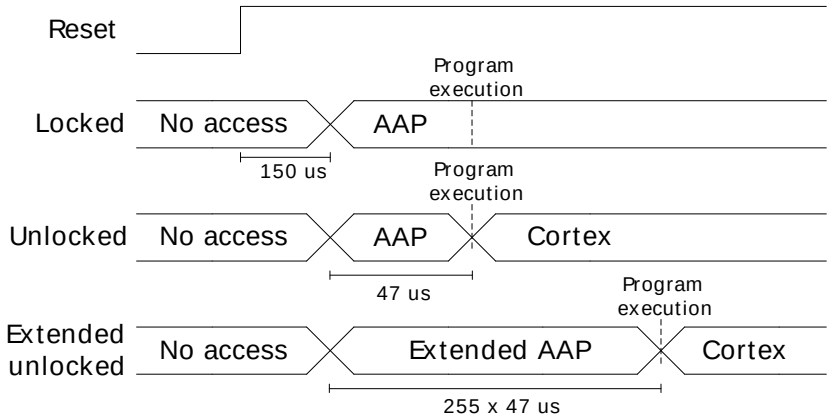
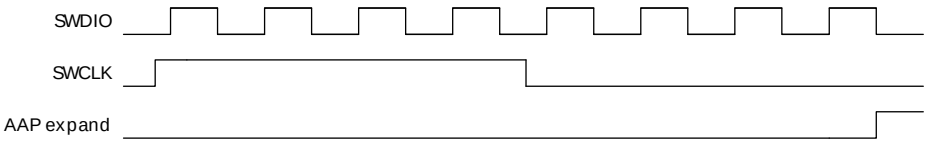


Figure 7.3. AAP Expansion



If the device is locked, it can be unlocked by writing a valid key to the AAP_CMDKEY register and then setting the DEVICEERASE bit of the AAP_CMD register via the debug interface. The commands are not executed before AAP_CMDKEY is invalidated, so this register should be cleared to to start the erase operation. This operation erases the main block of flash, all lock bits are reset and debug access through the AHB-AP is enabled. The operation takes 40 ms to complete. Note that the SRAM contents will also be deleted during a device erase, while the UD-page is not erased.

Even if the device is not locked, the can device can be erased through the AAP, using the above procedure during the AAP window. This can be useful if the device has been programmed with code that, e.g., disables the debug interface pins on start-up, or does something else that prevents communication with a debugger.

If the device is locked, the debugger may read the status from the AAP_STATUS register. When the ERASEBUSY bit is set low after DEVICEERASE of the AAP_CMD register is set, the debugger may set the SYSRESETREQ bit in the AAP_CMD register. After reset, the debugger may resume a normal debug session through the AHB-AP. If the device is not locked, the device erase starts when the AAP window closes, so it is not possible to poll the status.

7.5 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	AAP_CMD	W1	Command Register
0x004	AAP_CMDKEY	W1	Command Key Register
0x008	AAP_STATUS	R	Status Register
0x0FC	AAP_IDR	R	AAP Identification Register

7.6 Register Description

7.6.1 AAP_CMD - Command Register

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															W1	W1
Name																															SYSRESETREQ	DEVICEERASE

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	SYSRESETREQ	0	W1	**System Reset Request**
	A system reset request is generated when set to 1. This register is write enabled from the AAP_CMDKEY register.			
0	DEVICEERASE	0	W1	**Erase the Flash Main Block, SRAM and Lock Bits**
	When set, all data and program code in the main block is erased, the SRAM is cleared and then the Lock Bit (LB) page is erased. This also includes the Debug Lock Word (DLW), causing debug access to be enabled after the next reset. The information block User Data page (UD) is left unchanged, but the User data page Lock Word (ULW) is erased. This register is write enabled from the AAP_CMDKEY register.			

7.6.2 AAP_CMDKEY - Command Key Register

Offset	Bit Position																													
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x00000000															
Access																	W1															
Name																	WRITEKEY															

Bit	Name	Reset	Access	Description
31:0	WRITEKEY	0x00000000	W1	**CMD Key Register**

Bit	Name	Reset	Access	Description
	The key value must be written to this register to write enable the AAP_CMD register. After AAP_CMD is written, this register should be cleared to excecute the command.			

Value	Mode	Description
0xCFACC118	WRITEEN	Enable write to AAP_CMD

7.6.3 AAP_STATUS - Status Register

Offset	Bit Position																														
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																R
Name																																ERASEBUSY

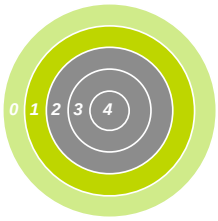
Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	ERASEBUSY	0	R	**Device Erase Command Status**
	This bit is set when a device erase is executing.			

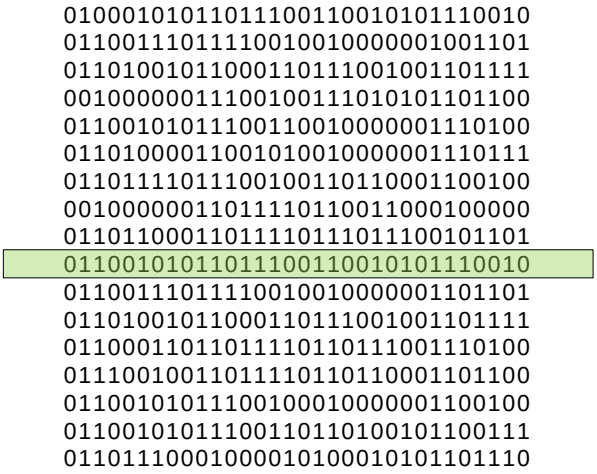
7.6.4 AAP_IDR - AAP Identification Register

Offset	Bit Position																														
0x0FC	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																0x16E60001																
Access																R																
Name																ID																

Bit	Name	Reset	Access	Description
31:0	ID	0x16E60001	R	**AAP Identification Register**
	Access port identification register in compliance with the ARM ADI v5 specification (JEDEC Manufacturer ID) .			

8 MSC - Memory System Controller





Quick Facts

What?

The user can perform Flash memory read, read configuration and write operations through the Memory System Controller (MSC) .

Why?

The MSC allows the application code, user data and flash lock bits to be stored in non-volatile Flash memory. Certain memory system functions, such as program memory wait-states and bus faults are also configured from the MSC peripheral register interface, giving the developer the ability to dynamically customize the memory system performance, security level, energy consumption and error handling capabilities to the requirements at hand.

How?

The MSC integrates a low-energy Flash IP with a charge pump, enabling minimum energy consumption while eliminating the need for external programming voltage to erase the memory. An easy to use write and erase interface is supported by an internal, fixed-frequency oscillator and autonomous flash timing and control reduces software complexity while not using other timer resources.

Application code may dynamically scale between high energy optimization and high code execution performance through advanced read modes.

A highly efficient low energy instruction cache reduces the number of flash reads significantly, thus saving energy. Performance is also improved when wait-states are used, since many of the wait-states are eliminated. Built-in performance counters can be used to measure the efficiency of the instruction cache.

8.1 Introduction

The Memory System Controller (MSC) is the program memory unit of the EZR32HG microcontroller. The flash memory is readable and writable from both the Cortex-M0+ and DMA. The flash memory is divided into two blocks; the main block and the information block. Program code is normally written to the main block. Additionally, the information block is available for special user data and flash lock bits. There is also a read-only page in the information block containing system and device calibration data. Read and write operations are supported in the energy modes EM0 and EM1.

8.2 Features

- AHB read interface
  - Scalable access performance to optimize the Cortex-M0+ code interface
    - Zero wait-state access up to 16 MHz and one wait-state for 16 MHz and above
    - Advanced energy optimization functionality
      - Instruction Cache
  - DMA read support in EM0 and EM1
- Command and status interface
  - Flash write and erase
    - Accessible from Cortex-M0+ in EM0
    - DMA write support in EM0 and EM1
  - Core clock independent Flash timing
    - Internal oscillator and internal timers for precise and autonomous Flash timing
      - General purpose timers are not occupied during Flash erase and write operations
  - Configurable interrupt erase abort
    - Improved interrupt predictability
  - Memory and bus fault control
- Security features
  - Lockable debug access
  - Page lock bits
  - SW Mass erase Lock bits
  - User data lock bits
- End-of-write and end-of-erase interrupts

8.3 Functional Description

The size of the main block is device dependent. The largest size available is 64 kB (64 pages). The information block has 1024 bytes available for user data. The information block also contains chip configuration data located in a reserved area. The main block is mapped to address 0x00000000 and the information block is mapped to address 0x0FE00000. Table 8.1 (p. 73) outlines how the Flash is mapped in the memory space. All Flash memory is organized into 1024 byte pages.

Table 8.1. MSC Flash Memory Mapping

Block	Page	Base address	Write/Erase by	Software readable	Purpose/Name	Size
Main[1]	0	0x00000000	Software, debug	Yes	User code and data	32 kB - 64 kB
	.		Software, debug	Yes		
	63	Software, debug	Yes			
Reserved	-	-	-	Reserved for flash expansion	~24 MB	
Information	0	0x0FE00000	Software, debug	Yes	User Data (UD)	1 kB
	-	-	-	Reserved		
	1	0x0FE04000	Write: Software, debug Erase: Debug only	Yes	Lock Bits (LB)	1 kB
	-	-	-	Reserved		
	2	0x0FE08000	-	Yes	Device Information (DI)	1 kB
	-	-	-	Reserved		
Reserved	-	0x0FE10000	-	-	Reserved for flash expansion	Rest of code space

[1]Block/page erased by a device erase

8.3.1 User Data (UD) Page Description

This is the user data page in the information block. The page can be erased and written by software. The page is erased by the ERASEPAGE command of the MSC_WRITECMD register. Note that the page is not erased by a device erase operation. The device erase operation is described in Section 7.4 (p. 67) .

8.3.2 Lock Bits (LB) Page Description

This page contains the following information:

- Debug Lock Word (DLW)
- User data page Lock Word (ULW)
- Mass erase Lock Word (MLW)
- Main block Page Lock Words (PLWs)

The words in this page are organized as shown in Table 8.2 (p. 73) :

Table 8.2. Lock Bits Page Structure

127	DLW
126	ULW
125	MLW
N	PLW[N]
…	…
1	PLW[1]
0	PLW[0]

Word 127 is the debug lock word (DLW). The four LSBs of this word are the debug lock bits. If these bits are 0xF, then debug access is enabled. If the bits are not 0xF, then debug access to the core is locked. See Section 7.4 (p. 67) for details on how to unlock the debug access.

Word 126 is the user page lock word (ULW). Bit 0 of this word is the User Data Page lock bit. Bit 1 in this word locks the Lock Bits Page.

Word 125 is the mass erase lock word (MLW). Bit 0 locks the entire flash. The mass erase lock bits will not have any effect on device erases initiated from the Authentication Access Port (AAP) registers. The AAP is described in more detail in Section 7.4 (p. 67) .

There are 32 page lock bits per page lock word (PLW). Bit 0 refers to the first page and bit 31 refers to the last page within a PLW. Thus, PLW[0] contains lock bits for page 0-31 in the main block. Similarly, PLW[1] contains lock bits for page 32-63 and so on. A page is locked when the bit is 0. A locked page cannot be erased or written.

The lock bits can be reset by a device erase operation initiated from the Authentication Access Port (AAP) registers. The AAP is described in more detail in Section 7.4 (p. 67) . Note that the AAP is only accessible from the debug interface, and cannot be accessed from the Cortex-M0+ core.

8.3.3 Device Information (DI) Page

This read-only page holds the calibration data for the oscillator and other analog peripherals from the production test as well as a unique device ID. The page is further described in Section 6.6 (p. 64) .

8.3.4 Post-reset Behavior

Calibration values are automatically written to registers by the MSC before application code startup. The values are also available to read from the DI page for later reference by software. Other information such as the device ID and production date is also stored in the DI page and is readable from software.

8.3.4.1 One Wait-state Access

After reset, the HFCORECLK is normally 14 MHz from the HFRCO and the MODE field of the MSC_READCTRL register is set to WS1 (one wait-state). The reset value must be WS1 as an uncalibrated HFRCO may produce a frequency higher than 16 MHz. Software must not select a zero wait-state mode unless the clock is guaranteed to be 16 MHz or below, otherwise the resulting behavior is undefined. If a HFCORECLK frequency above 16 MHz is to be set by software, the MODE field of the MSC_READCTRL register must be set to WS1 before the core clock is switched to the higher frequency clock source.

When changing to a lower frequency, the MODE field of the MSC_READCTRL register can be set to WS0, but only after the frequency transition is completed. If the HFRCO is used, wait until the oscillator is stable on the new frequency. Otherwise, the behavior is unpredictable.

8.3.4.2 Zero Wait-state Access

At 16 MHz and below, read operations from flash may be performed without any wait-states. Zero wait-state access greatly improves code execution performance at frequencies from 16 MHz and below.

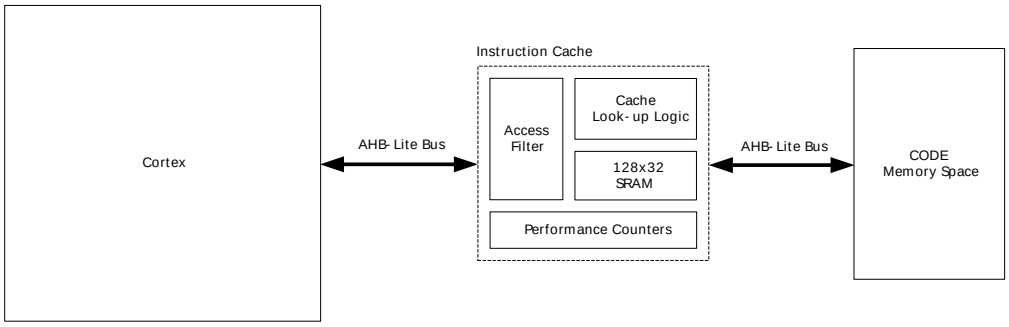
8.3.4.3 Instruction Cache

The MSC includes an instruction cache. The instruction cache for the internal flash memory is enabled by default, but can be disabled by setting IFCDIS in MSC_READCTRL. When enabled, the instruction cache typically reduces the number of flash reads significantly, thus saving energy. In most cases a cache hit-rate of more than 70 % is achievable. When a 32-bit instruction fetch hits in the cache the data

is returned to the processor in one clock cycle. Thus, performance is also improved when wait-states are used (i.e. running at frequencies above 16 MHz).

The instruction cache is connected directly to the Cortex-M0+ and functions as a memory access filter between the processor and the memory system, as illustrated in Figure 8.1 (p. 75) . The cache consists of an access filter, lookup logic, a 128x32 SRAM (512 bytes) and two performance counters. The access filter checks that the address for the access is of an instruction in the code space (instructions in RAM outside the code space are not cached). If the address matches, the cache lookup logic and SRAM is enabled. Otherwise, the cache is bypassed and the access is forwarded to the memory system. The cache is then updated when the memory access completes. The performance counters, when enabled, keep track of the number of cache hits and misses. The cache consists of 16 8-word cachelines organized as 4 sets with 4 ways. The cachelines are filled up continuously one word at a time as the individual words are requested by the processor. Thus, not all words of a cacheline might be valid at a given time.

Figure 8.1. Instruction Cache



By default, the instruction cache is automatically invalidated when the contents of the flash is changed (i.e. written or erased). In many cases, however, the application only makes changes to data in the flash, not code. In this case, the automatic invalidate feature can be disabled by setting AIDIS in MSC_READCTRL. The cache can (independent of the AIDIS setting) be manually invalidated by writing 1 to INVCACHE in MSC_CMD.

In general it is highly recommended to keep the cache enabled all the time. However, for some sections of code with very low cache hit-rate more energy-efficient execution can be achieved by disabling the cache temporarily. To measure the hit-rate of a code-section, the built-in performance counters can be used. Before the section, start the performance counters by writing 1 to STARTPC in MSC_CMD. This starts the performance counters, counting from 0. At the end of the section, stop the performance counters by writing 1 to STOPPC in MSC_CMD. The number of cache hits and cache misses for that section can then be read from MSC_CACHEHITS and MSC_CACHEMISSES respectively. The total number of 32-bit instruction fetches will be MSC_CACHEHITS + MSC_CACHEMISSES. Thus, the cache hit-ratio can be calculated as MSC_CACHEHITS / (MSC_CACHEHITS + MSC_CACHEMISSES). When MSC_CACHEHITS overflows the CHOF interrupt flag is set. When MSC_CACHEMISSES overflows the CMOF interrupt flag is set. These flags must be cleared explicitly by software. The range of the performance counters can thus be extended by increasing a counter in the MSC interrupt routine. The performance counters only count when a cache lookup is performed. If the lookup fails, MSC_CACHEMISSES is increased. If the lookup is successful, MSC_CACHEHITS is increased. For example, a cache lookup is not performed if the cache is disabled or the code is executed from RAM outside the code space.

The cache content is not retained in EM2, EM3 and EM4. The cache is therefore invalidated regardless of the setting of AIDIS in MSC_READCTRL when entering these energy modes. Applications that switch frequently between EM0 and EM2/3 and execute the very same non-looping code almost every time will most likely benefit from putting this code in RAM. The interrupt vectors can also be put in RAM to reduce current consumption even further.

8.3.5 Erase and Write Operations

The AUXHFRCO is used for timing during flash write and erase operations. To achieve correct timing, the MSC_TIMEBASE register has to be configured according to the settings in CMU_AUXHFRCOCTRL. BASE in MSC_TIMEBASE defines how many AUXCLK cycles - 1 there is in 1 us or 5 us, depending on the configuration of PERIOD. To ensure that timing of flash write and erase operations is within the specification of the flash, the value written to BASE should give at least a 10% margin with respect to the period, i.e. for the 1 us PERIOD, the number of cycles should at least span 1.1 us, and for the 5 us period they should span at least 5.5 us. For the 1 MHz band, PERIOD in MSC_TIMEBASE should be set to 5US, while it should be set to 1US for all other AUXHFRCO bands.

Both page erase and write operations require that the address is written into the MSC_ADDRB register. For erase operations, the address may be any within the page to be erased. Load the address by writing 1 to the LADDRIM bit in the MSC_WRITECMD register. The LADDRIM bit only has to be written once when loading the first address. After each word is written the internal address register ADDR will be incremented automatically by 4. The INVADDR bit of the MSC_STATUS register is set if the loaded address is outside the flash and the LOCKED bit of the MSC_STATUS register is set if the page addressed is locked. Any attempts to command erase of or write to the page are ignored if INVADDR or the LOCKED bits of the MSC_STATUS register are set. To abort an ongoing erase, set the ERASEABORT bit in the MSC_WRITECMD register.

When a word is written to the MSC_WDATA register, the WDATAREADY bit of the MSC_STATUS register is cleared. When this status bit is set, software or DMA may write the next word.

A single word write is commanded by setting the WRITEONCE bit of the MSC_WRITECMD register. The operation is complete when the BUSY bit of the MSC_STATUS register is cleared and control of the flash is handed back to the AHB interface, allowing application code to resume execution.

For a DMA write the software must write the first word to the MSC_WDATA register and then set the WRITETRIG bit of the MSC_WRITECMD register. DMA triggers when the WDATAREADY bit of the MSC_STATUS register is set.

It is possible to write words twice between each erase by keeping at 1 the bits that are not to be changed. Let us take as an example writing two 16 bit values, 0xAAAA and 0x5555. To safely write them in the same flash word this method can be used:

- Write 0xFFFFAAAA (word in flash becomes 0xFFFFAAAA)
- Write 0x5555FFFF (word in flash becomes 0x5555AAAA)

Note that there is a maximum of two writes to the same word between each erase due to a physical limitation of the flash.

Note

During a write or erase, flash read accesses will be stalled, effectively halting code execution from flash. Code execution continues upon write/erase completion. Code residing in RAM may be executed during a write/erase operation.

Note

The MSC_WDATA and MSC_ADDRB registers are not retained when entering EM2 or lower energy modes.

8.3.5.1 Mass erase

A mass erase can be initiated from software using ERASEMAIN0 in MSC_WRITECMD. This command will start a mass erase of the entire flash. Prior to initiating a mass erase, MSC_MASSLOCK must be unlocked by writing 0x631A to it. After a mass erase has been started, this register can be locked again to prevent runaway code from accidentally triggering a mass erase.

The regular flash page lock bits will not prevent a mass erase. To prevent software from initiating mass erases, use the mass erase lock bits in the mass erase lock word (MLW).

8.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	MSC_CTRL	RW	Memory System Control Register
0x004	MSC_READCTRL	RW	Read Control Register
0x008	MSC_WRITECTRL	RW	Write Control Register
0x00C	MSC_WRITECMD	W1	Write Command Register
0x010	MSC_ADDRB	RW	Page Erase/Write Address Buffer
0x018	MSC_WDATA	RW	Write Data Register
0x01C	MSC_STATUS	R	Status Register
0x02C	MSC_IF	R	Interrupt Flag Register
0x030	MSC_IFS	W1	Interrupt Flag Set Register
0x034	MSC_IFC	W1	Interrupt Flag Clear Register
0x038	MSC_IEN	RW	Interrupt Enable Register
0x03C	MSC_LOCK	RW	Configuration Lock Register
0x040	MSC_CMD	W1	Command Register
0x044	MSC_CACHEHITS	R	Cache Hits Performance Counter
0x048	MSC_CACHEMISSES	R	Cache Misses Performance Counter
0x050	MSC_TIMEBASE	RW	Flash Write and Erase Timebase
0x054	MSC_MASSLOCK	RW	Mass Erase Lock Register
0x058	MSC_IRQLATENCY	RW	Irq Latency Register

8.5 Register Description

8.5.1 MSC_CTRL - Memory System Control Register

Offset	Bit Position																														
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																1
Access																																RW
Name																																BUSFAULT

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	BUSFAULT	1	RW	**Bus Fault Response Enable**

When this bit is set, the memory system generates bus error response.

Value	Mode	Description
0	GENERATE	A bus fault is generated on access to unmapped code and system space.
1	IGNORE	Accesses to unmapped address space is ignored.

8.5.2 MSC_READCTRL - Read Control Register

Offset	Bit Position																															
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0			0	0		0x1	
Access																									RW			RW	RW		RW	
Name																									RAMCEN			AIDIS	IFCDIS		MODE	

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7	RAMCEN	0	RW	**RAM Cache Enable**
	Enable instruction caching for RAM in code-space.			
6:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	AIDIS	0	RW	**Automatic Invalidate Disable**
	When this bit is set the cache is not automatically invalidated when a write or page erase is performed.			
3	IFCDIS	0	RW	**Internal Flash Cache Disable**
	Disable instruction cache for internal flash memory.			
2:0	MODE	0x1	RW	**Read Mode**
	If software wants to set a core clock frequency above 16 MHz, this register must be set to WS1 before the core clock is switched to the higher frequency. When changing to a lower frequency, this register can be set to WS0 after the frequency transition has been completed. After reset, the core clock is 14 MHz from the HFRCO but the MODE field of MSC_READCTRL register is set to WS1. This is because the HFRCO may produce a frequency above 16 MHz before it is calibrated. If the HFRCO is used as clock source, wait until the oscillator is stable on the new frequency to avoid unpredictable behavior.			

Value	Mode	Description
0	WS0	Zero wait-states inserted in fetch or read transfers.
1	WS1	One wait-state inserted for each fetch or read transfer. This mode is required for a core frequency above 16 MHz.

8.5.3 MSC_WRITECTRL - Write Control Register

Offset	Bit Position																															
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															RW	RW
Name																															IRQERASEABORT	WREN

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	IRQERASEABORT	0	RW	**Abort Page Erase on Interrupt**
	When this bit is set to 1, any Cortex interrupt aborts any current page erase operation.			
0	WREN	0	RW	**Enable Write/Erase Controller**

Bit	Name	Reset	Access	Description
	When this bit is set, the MSC write and erase functionality is enabled.			

8.5.4 MSC_WRITECMD - Write Command Register

Offset	Bit Position																													
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																				0				0			0	0	0	0	0	0
Access																				W1				W1			W1	W1	W1	W1	W1	W1
Name																				CLEARWDATA				ERASEMAIN0			ERASEABORT	WRITETRIG	WRITEONCE	WRITEEND	ERASEPAGE	LADDRIM

Bit	Name	Reset	Access	Description
31:13	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
12	CLEARWDATA	0	W1	**Clear WDATA state**
	Will set WDATAREADY and DMA request. Should only be used when no write is active.			
11:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8	ERASEMAIN0	0	W1	**Mass erase region 0**
	Initiate mass erase of region 0. For devices supporting read-while-write, this is the lower half of the flash. For other devices it is the entire flash. Before use MSC_MASSLOCK must be unlocked. To completely prevent access from software, clear bit 0 in the mass erase lock-word (MLW).			
7:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	ERASEABORT	0	W1	**Abort erase sequence**
	Writing to this bit will abort an ongoing erase sequence.			
4	WRITETRIG	0	W1	**Word Write Sequence Trigger**
	Functions like MSC_CMD_WRITEONCE, but will set MSC_STATUS_WORDTIMEOUT if no new data is written to MSC_WDATA within the 30 µs timeout.			
3	WRITEONCE	0	W1	**Word Write-Once Trigger**
	Start write of the first word written to MSC_WDATA, then add 4 to ADDR and write the next word if available within a 30 µs timeout. When ADDR is incremented past the page boundary, ADDR is set to the base of the page.			
2	WRITEEND	0	W1	**End Write Mode**
	Write 1 to end write mode when using the WRITETRIG command.			
1	ERASEPAGE	0	W1	**Erase Page**
	Erase any user defined page selected by the MSC_ADDRB register. The WREN bit in the MSC_WRITECTRL register must be set in order to use this command.			
0	LADDRIM	0	W1	**Load MSC_ADDRB into ADDR**
	Load the internal write address register ADDR from the MSC_ADDRB register. The internal address register ADDR is incremented automatically by 4 after each word is written. When ADDR is incremented past the page boundary, ADDR is set to the base of the page.			

8.5.5 MSC_ADDRB - Page Erase/Write Address Buffer

Offset	Bit Position																														
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	RW																															
Name	ADDRB																															

Bit	Name	Reset	Access	Description
31:0	ADDRB	0x00000000	RW	**Page Erase or Write Address Buffer**
	This register holds the page address for the erase or write operation. This register is loaded into the internal MSC_ADDR register when the LADDRIM field in MSC_WRITECMD is set. The MSC_ADDR register is not readable. This register is not retained when entering EM2 or lower energy modes.			

8.5.6 MSC_WDATA - Write Data Register

Offset	Bit Position																														
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	RW																															
Name	WDATA																															

Bit	Name	Reset	Access	Description
31:0	WDATA	0x00000000	RW	**Write Data**
	The data to be written to the address in MSC_ADDR. This register must be written when the WDATAREADY bit of MSC_STATUS is set. This register is not retained when entering EM2 or lower energy modes.			

8.5.7 MSC_STATUS - Status Register

Offset	Bit Position																														
0x01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																										0	0	0	1	0	0	0
Access																										R	R	R	R	R	R	R
Name																										PCRUNNING	ERASEABORTED	WORDTIMEOUT	WDATAREADY	INVADDR	LOCKED	BUSY

Bit	Name	Reset	Access	Description
31:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	PCRUNNING	0	R	**Performance Counters Running**
	This bit is set while the performance counters are running. When one performance counter reaches the maximum value, this bit is cleared.			
5	ERASEABORTED	0	R	**The Current Flash Erase Operation Aborted**
	When set, the current erase operation was aborted by interrupt.			
4	WORDTIMEOUT	0	R	**Flash Write Word Timeout**
	When this bit is set, MSC_WDATA was not written within the timeout. The flash write operation timed out and access to the flash is returned to the AHB interface. This bit is cleared when the ERASEPAGE, WRITETRIG or WRITEONCE commands in MSC_WRITECMD are triggered.			
3	WDATAREADY	1	R	**WDATA Write Ready**
	When this bit is set, the content of MSC_WDATA is read by MSC Flash Write Controller and the register may be updated with the next 32-bit word to be written to flash. This bit is cleared when writing to MSC_WDATA.			
2	INVADDR	0	R	**Invalid Write Address or Erase Page**
	Set when software attempts to load an invalid (unmapped) address into ADDR.			
1	LOCKED	0	R	**Access Locked**
	When set, the last erase or write is aborted due to erase/write access constraints.			
0	BUSY	0	R	**Erase/Write Busy**
	When set, an erase or write operation is in progress and new commands are ignored.			

8.5.8 MSC_IF - Interrupt Flag Register

Offset	Bit Position																														
0x02C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																													0	0	0	0
Access																													R	R	R	R
Name																													CMOF	CHOF	WRITE	ERASE

Bit	Name	Reset	Access	Description
31:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	CMOF	0	R	**Cache Misses Overflow Interrupt Flag**
	Set when MSC_CACHEMISSES overflows.			
2	CHOF	0	R	**Cache Hits Overflow Interrupt Flag**
	Set when MSC_CACHEHITS overflows.			
1	WRITE	0	R	**Write Done Interrupt Read Flag**
	Set when a write is done.			
0	ERASE	0	R	**Erase Done Interrupt Read Flag**
	Set when erase is done.			

8.5.9 MSC_IFS - Interrupt Flag Set Register

Offset	Bit Position																															
0x030	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																													0	0	0	0
Access																													W1	W1	W1	W1
Name																													CMOF	CHOF	WRITE	ERASE

Bit	Name	Reset	Access	Description
31:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	CMOF	0	W1	**Cache Misses Overflow Interrupt Set**
	Set the CMOF flag and generate interrupt.			
2	CHOF	0	W1	**Cache Hits Overflow Interrupt Set**
	Set the CHOF flag and generate interrupt.			
1	WRITE	0	W1	**Write Done Interrupt Set**
	Set the write done bit and generate interrupt.			
0	ERASE	0	W1	**Erase Done Interrupt Set**
	Set the erase done bit and generate interrupt.			

8.5.10 MSC_IFC - Interrupt Flag Clear Register

Offset	Bit Position																															
0x034	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																													0	0	0	0
Access																													W1	W1	W1	W1
Name																													CMOF	CHOF	WRITE	ERASE

Bit	Name	Reset	Access	Description
31:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	CMOF	0	W1	**Cache Misses Overflow Interrupt Clear**
	Clear the CMOF interrupt flag.			
2	CHOF	0	W1	**Cache Hits Overflow Interrupt Clear**
	Clear the CHOF interrupt flag.			
1	WRITE	0	W1	**Write Done Interrupt Clear**
	Clear the write done bit.			
0	ERASE	0	W1	**Erase Done Interrupt Clear**
	Clear the erase done bit.			

8.5.11 MSC_IEN - Interrupt Enable Register

Offset	Bit Position																															
0x038	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																													0	0	0	0
Access																													RW	RW	RW	RW
Name																													CMOF	CHOF	WRITE	ERASE

Bit	Name	Reset	Access	Description
31:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	CMOF	0	RW	**Cache Misses Overflow Interrupt Enable**
	Enable the cache misses performance counter overflow interrupt.			
2	CHOF	0	RW	**Cache Hits Overflow Interrupt Enable**
	Enable the cache hits performance counter overflow interrupt.			
1	WRITE	0	RW	**Write Done Interrupt Enable**
	Enable the write done interrupt.			
0	ERASE	0	RW	**Erase Done Interrupt Enable**
	Enable the erase done interrupt.			

8.5.12 MSC_LOCK - Configuration Lock Register

Offset	Bit Position																															
0x03C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	LOCKKEY															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	LOCKKEY	0x0000	RW	**Configuration Lock**

Write any other value than the unlock code to lock access to MSC_CTRL, MSC_READCTRL, MSC_WRITECTRL and MSC_TIMEBASE. Write the unlock code to enable access. When reading the register, bit 0 is set when the lock is enabled.

Mode	Value	Description
Read Operation		
UNLOCKED	0	MSC registers are unlocked.
LOCKED	1	MSC registers are locked.
Write Operation		
LOCK	0	Lock MSC registers.
UNLOCK	0x1B71	Unlock MSC registers.

8.5.13 MSC_CMD - Command Register

Offset	Bit Position																														
0x040	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																														0	0	0
Access																														W1	W1	W1
Name																														STOPPC	STARTPC	INVCACHE

Bit	Name	Reset	Access	Description
31:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	STOPPC	0	W1	**Stop Performance Counters**
	Use this command bit to stop the performance counters.			
1	STARTPC	0	W1	**Start Performance Counters**
	Use this command bit to start the performance counters. The performance counters always start counting from 0.			
0	INVCACHE	0	W1	**Invalidate Instruction Cache**
	Use this register to invalidate the instruction cache.			

8.5.14 MSC_CACHEHITS - Cache Hits Performance Counter

Offset	Bit Position																														
0x044	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset													0x00000																			
Access													R																			
Name													CACHEHITS																			

Bit	Name	Reset	Access	Description
31:20	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
19:0	CACHEHITS	0x00000	R	**Cache hits since last performance counter start command.**
	Use to measure cache performance for a particular code section.			

8.5.15 MSC_CACHEMISSES - Cache Misses Performance Counter

Offset	Bit Position																															
0x048	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset													0x00000																			
Access													R																			
Name													CACHEMISSES																			

Bit	Name	Reset	Access	Description
31:20	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
19:0	CACHEMISSES	0x00000	R	**Cache misses since last performance counter start command.**
	Use to measure cache performance for a particular code section.			

8.5.16 MSC_TIMEBASE - Flash Write and Erase Timebase

Offset	Bit Position																															
0x050	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																0											0x10					
Access																RW											RW					
Name																PERIOD											BASE					

Bit	Name	Reset	Access	Description
31:17	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
16	PERIOD	0	RW	**Sets the timebase period**
	Decides whether TIMEBASE specifies the number of AUX cycles in 1 us or 5 us. 5 us should only be used with 1 MHz AUXHFRCO band.			

Value	Mode	Description
0	1US	TIMEBASE period is 1 us.
1	5US	TIMEBASE period is 5 us.

15:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5:0	BASE	0x10	RW	**Timebase used by MSC to time flash writes and erases**
	Should be set to the number of full AUX clock cycles in the period given by MSC_TIMEBASE_PERIOD. I.e. 1.1 us or 5.5. us with PERIOD cleared or set, respectively. The resetvalue of the timebase matches a 14 MHz AUXHFRCO, which is the default frequency of the AUXHFRCO.			

8.5.17 MSC_MASSLOCK - Mass Erase Lock Register

Offset	Bit Position																													
0x054	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0001															
Access																	RW															
Name																	LOCKKEY															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	LOCKKEY	0x0001	RW	**Mass Erase Lock**

Write any other value than the unlock code to lock access the the ERASEMAIN0 and ERASEMAIN1 commands. Write the unlock code 631A to enable access. When reading the register, bit 0 is set when the lock is enabled. Locked by default.

Mode	Value	Description
Read Operation		
UNLOCKED	0	Mass erase unlocked.
LOCKED	1	Mass erase locked.
Write Operation		
LOCK	0	Lock mass erase.
UNLOCK	0x631A	Unlock mass erase.

8.5.18 MSC_IRQLATENCY - Irq Latency Register

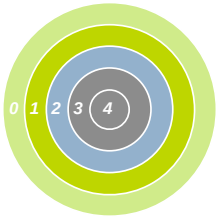
Offset	Bit Position																													
0x058	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0x00							
Access																									RW							
Name																									IRQLATENCY							

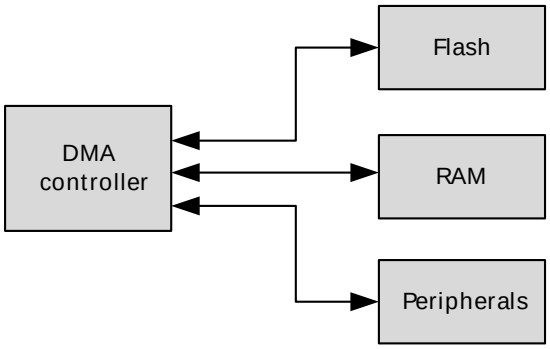
Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:0	IRQLATENCY	0x00	RW	**Irq Latency Register**

Specify the minimum number of HCORECLK-cycles to wait before handling an interrupt after it has been asserted. This can be used to achieve deterministic (zero-jitter) behavior when handling interrupts, at the cost of speed. To achieve zero-jitter with zero wait-states in flash, set this to 9.

IRQLATENCY	Description
0	Interrupts will be handled as quickly as possible.
1 - 255	The CM0+ will use at least IRQLATENCY+6 HFCORECLK-cycles to handle interrupts.

9 DMA - DMA Controller





Quick Facts

What?

The DMA controller can move data without CPU intervention, effectively reducing the energy consumption for a data transfer.

Why?

The DMA can perform data transfers more energy efficiently than the CPU and allows autonomous operation in low energy modes. The LEUART can for instance provide full UART communication in EM2, consuming only a few µA by using the DMA to move data between the LEUART and RAM.

How?

The DMA controller has multiple highly configurable, prioritized DMA channels. Advanced transfer modes such as ping-pong and scatter-gather make it possible to tailor the controller to the specific needs of an application.

9.1 Introduction

The Direct Memory Access (DMA) controller performs memory operations independently of the CPU. This has the benefit of reducing the energy consumption and the workload of the CPU, and enables the system to stay in low energy modes for example when moving data from the USART to RAM. The DMA controller uses the PL230 µDMA controller licensed from ARM[1]. Each of the PL230s channels on the EZR32 can be connected to any of the EZR32 peripherals.

9.2 Features

- The DMA controller is accessible as a memory mapped peripheral
- Possible data transfers include
  - RAM/Flash to peripheral
  - RAM to Flash
  - Peripheral to RAM
  - RAM/Flash to RAM
- The DMA controller has 4 independent channels
- Each channel has one (primary) or two (primary and alternate) descriptors
- The configuration for each channel includes
  - Transfer mode
  - Priority
  - Word-count
  - Word-size (8, 16, 32 bit)
- The transfer modes include
  - Basic (using the primary or alternate DMA descriptor)

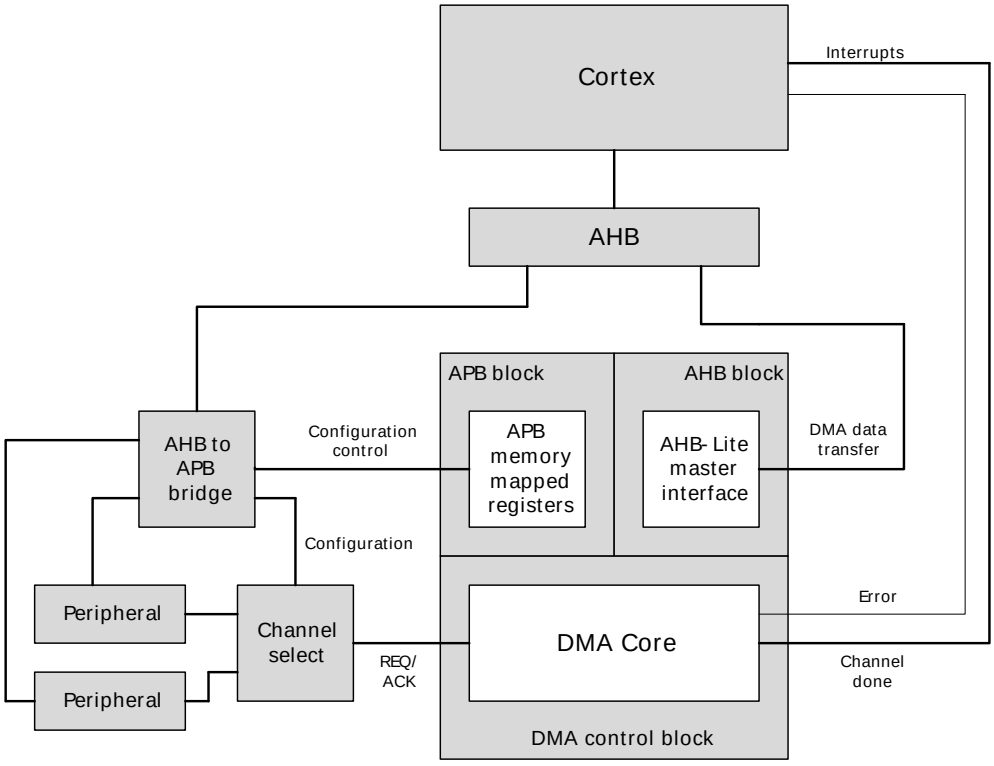
[1]ARM PL230 homepage [http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0417a/index.html]

  - Ping-pong (switching between the primary or alternate DMA descriptors, for continuous data flow to/from peripherals)
  - Scatter-gather (using the primary descriptor to configure the alternate descriptor)
- Each channel has a programmable transfer length
- Channels 0 and 1 support looped transfers
- Channel 0 supports 2D copy
- A DMA channel can be triggered by any of several sources:
  - Communication modules (USART, LEUART)
  - Timers (TIMER)
  - Analog modules (ADC)
  - Software
- Programmable mapping between channel number and peripherals - any DMA channel can be triggered by any of the available sources
- Interrupts upon transfer completion
- Data transfer to/from LEUART in EM2 is supported by the DMA, providing extremely low energy consumption while performing UART communications

9.3 Block Diagram

An overview of the DMA and the modules it interacts with is shown in Figure 9.1 (p. 89) .

Figure 9.1. DMA Block Diagram



The DMA Controller consists of four main parts:

- An APB block allowing software to configure the DMA controller
- An AHB block allowing the DMA to read and write the DMA descriptors and the source and destination data for the DMA transfers
- A DMA control block controlling the operation of the DMA, including request/acknowledge signals for the connected peripherals
- A channel select block routing the right peripheral request to each DMA channel

9.4 Functional Description

The DMA Controller is highly flexible. It is capable of transferring data between peripherals and memory without involvement from the processor core. This can be used to increase system performance by off-loading the processor from copying large amounts of data or avoiding frequent interrupts to service peripherals needing more data or having available data. It can also be used to reduce the system energy consumption by making the DMA work autonomously with the LEUART for data transfer in EM2 without having to wake up the processor core from sleep.

The DMA Controller contains 4 independent channels. Each of these channels can be connected to any of the available peripheral trigger sources by writing to the configuration registers, see Section 9.4.1 (p. 90) . In addition, each channel can be triggered by software (for large memory transfers or for debugging purposes).

What the DMA Controller should do (when one of its channels is triggered) is configured through channel descriptors residing in system memory. Before enabling a channel, the software must therefore take care to write this configuration to memory. When a channel is triggered, the DMA Controller will first read the channel descriptor from system memory, and then it will proceed to perform the memory transfers as specified by the descriptor. The descriptor contains the memory address to read from, the memory address to write to, the number of bytes to be transferred, etc. The channel descriptor is described in detail in Section 9.4.3 (p. 100) .

In addition to the basic transfer mode, the DMA Controller also supports two advanced transfer modes; ping-pong and scatter-gather. Ping-pong transfers are ideally suited for streaming data for high-speed peripheral communication as the DMA will be ready to retrieve the next incoming data bytes immediately while the processor core is still processing the previous ones (and similarly for outgoing communication). Scatter-gather involves executing a series of tasks from memory and allows sophisticated schemes to be implemented by software.

Using different priority levels for the channels and setting the number of bytes after which the DMA Controller re-arbitrates, it is possible to ensure that timing-critical transfers are serviced on time.

9.4.1 Channel Select Configuration

The channel select block allows selecting which peripheral's request lines (dma_req, dma_sreq) to connect to each DMA channel.

This configuration is done by software through the control registers DMA_CH0_CTRL-DMA_CH3_CTRL, with SOURCESEL and SIGSEL components. SOURCESEL selects which peripheral to listen to and SIGSEL picks which output signals to use from the selected peripheral.

All peripherals are connected to dma_req. When this signal is triggered, the DMA performs a number of transfers as specified by the channel descriptor (2^R). The USARTs are additionally connected to the dma_sreq line. When only dma_sreq is asserted but not dma_req, then the DMA will perform exactly one transfer only (given that dma_sreq is enabled by software).

Note

A DMA channel should not be active when the clock to the selected peripheral is off.

9.4.2 DMA control

9.4.2.1 DMA arbitration rate

You can configure when the controller arbitrates during a DMA transfer. This enables you to reduce the latency to service a higher priority channel.

The controller provides four bits that configure how many AHB bus transfers occur before it re-arbitrates. These bits are known as the R_power bits because the value you enter, R, is raised to the power of two

and this determines the arbitration rate. For example, if R = 4 then the arbitration rate is 2^4, that is, the controller arbitrates every 16 DMA transfers.

Table 9.1 (p. 91) lists the arbitration rates.

Table 9.1. AHB bus transfer arbitration interval

R_power	Arbitrate after x DMA transfers
b0000	x = 1
b0001	x = 2
b0010	x = 4
b0011	x = 8
b0100	x = 16
b0101	x = 32
b0110	x = 64
b0111	x = 128
b1000	x = 256
b1001	x = 512
b1010 - b1111	x = 1024

Note

You must take care not to assign a low-priority channel with a large R_power because this prevents the controller from servicing high-priority requests, until it re-arbitrates.

The number of dma transfers N that need to be done is specified by the user. When $N > 2^R$ and is not an integer multiple of 2^R then the controller always performs sequences of 2^R transfers until $N < 2^R$ remain to be transferred. The controller performs the remaining N transfers at the end of the DMA cycle.

You store the value of the R_power bits in the channel control data structure. See Section 9.4.3.3 (p. 103) for more information about the location of the R_power bits in the data structure.

9.4.2.2 Priority

When the controller arbitrates, it determines the next channel to service by using the following information:

- the channel number
- the priority level, default or high, that is assigned to the channel.

You can configure each channel to use either the default priority level or a high priority level by setting the DMA_CHPRIS register.

Channel number zero has the highest priority and as the channel number increases, the priority of a channel decreases. Table 9.2 (p. 91) lists the DMA channel priority levels in descending order of priority.

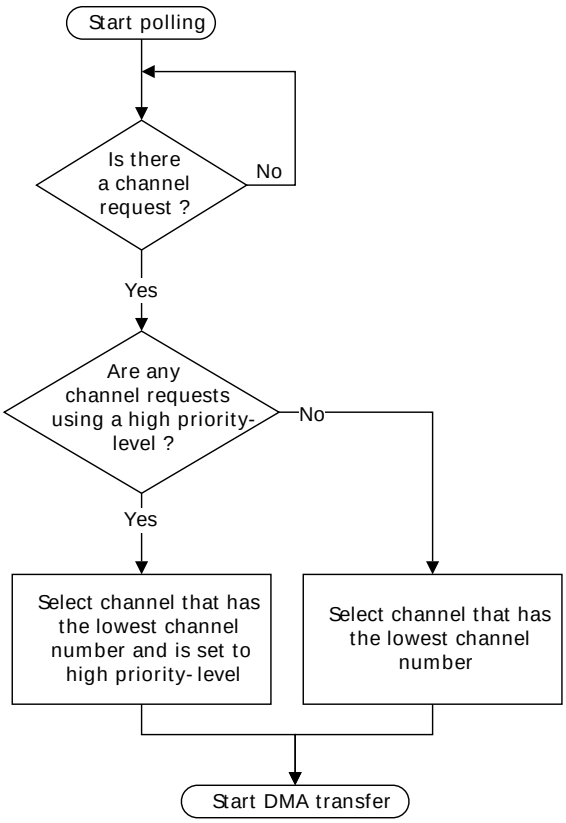
Table 9.2. DMA channel priority

Channel number	Priority level setting	Descending order of channel priority
0	High	Highest-priority DMA channel
1	High	-
2	High	-

Channel number	Priority level setting	Descending order of channel priority
3	High	-
0	Default	-
1	Default	-
2	Default	-
3	Default	Lowest-priority DMA channel

After a DMA transfer completes, the controller polls all the DMA channels that are available. Figure 9.2 (p. 92) shows the process it uses to determine which DMA transfer to perform next.

Figure 9.2. Polling flowchart



9.4.2.3 DMA cycle types

The cycle_ctrl bits control how the controller performs a DMA cycle. You can set the cycle_ctrl bits as Table 9.3 (p. 92) lists.

Table 9.3. DMA cycle types

cycle_ctrl	Description
b000	Channel control data structure is invalid
b001	Basic DMA transfer
b010	Auto-request
b011	Ping-pong
b100	Memory scatter-gather using the primary data structure

cycle_ctrl	Description
b101	Memory scatter-gather using the alternate data structure
b110	Peripheral scatter-gather using the primary data structure
b111	Peripheral scatter-gather using the alternate data structure

Note

The cycle_ctrl bits are located in the channel_cfg memory location that Section 9.4.3.3 (p. 103) describes.

For all cycle types, the controller arbitrates after 2^R DMA transfers. If you set a low-priority channel with a large 2^R value then it prevents all other channels from performing a DMA transfer, until the low-priority DMA transfer completes. Therefore, you must take care when setting the R_power, that you do not significantly increase the latency for high-priority channels.

9.4.2.3.1 Invalid

After the controller completes a DMA cycle it sets the cycle type to invalid, to prevent it from repeating the same DMA cycle.

9.4.2.3.2 Basic

In this mode, you configure the controller to use either the primary or the alternate data structure. After you enable the channel C and the controller receives a request for this channel, then the flow for this DMA cycle is as follows:

1. The controller performs 2^R transfers. If the number of transfers remaining becomes zero, then the flow continues at step 3 (p. 93) .
2. The controller arbitrates:
   - if a higher-priority channel is requesting service then the controller services that channel
   - if the peripheral or software signals a request to the controller then it continues at step 1 (p. 93) .
3. The controller sets `dma_done[C]` HIGH for one `HFCORECLK` cycle. This indicates to the host processor that the DMA cycle is complete.

9.4.2.3.3 Auto-request

When the controller operates in this mode, it is only necessary for it to receive a single request to enable it to complete the entire DMA cycle. This enables a large data transfer to occur, without significantly increasing the latency for servicing higher priority requests, or requiring multiple requests from the processor or peripheral.

You can configure the controller to use either the primary or the alternate data structure. After you enable the channel C and the controller receives a request for this channel, then the flow for this DMA cycle is as follows:

1. The controller performs 2^R transfers for channel C. If the number of transfers remaining is zero the flow continues at step 3 (p. 93) .
2. The controller arbitrates. When channel C has the highest priority then the DMA cycle continues at step 1 (p. 93) .
3. The controller sets `dma_done[C]` HIGH for one `HFCORECLK` cycle. This indicates to the host processor that the DMA cycle is complete.

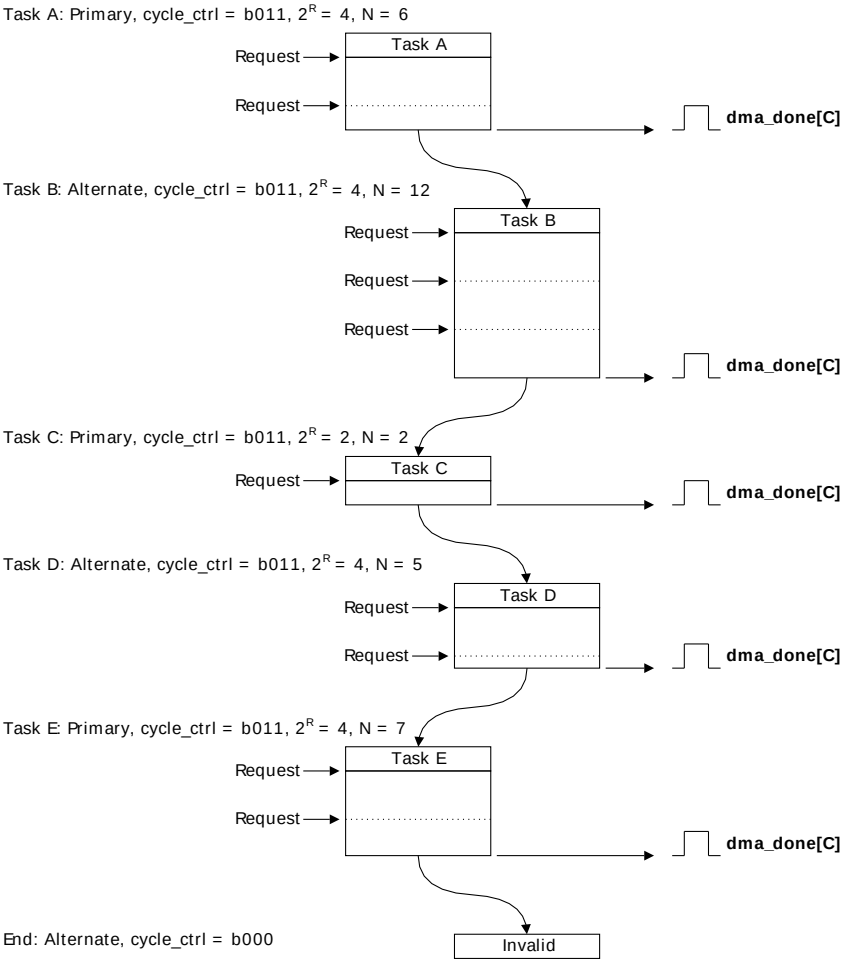
9.4.2.3.4 Ping-pong

In ping-pong mode, the controller performs a DMA cycle using one of the data structures (primary or alternate) and it then performs a DMA cycle using the other data structure. The controller continues to

switch from primary to alternate to primary… until it reads a data structure that is invalid, or until the host processor disables the channel.

Figure 9.3 (p. 94) shows an example of a ping-pong DMA transaction.

Figure 9.3. Ping-pong example



In Figure 9.3 (p. 94) :

Task A
1. The host processor configures the primary data structure for task A.
2. The host processor configures the alternate data structure for task B. This enables the controller to immediately switch to task B after task A completes, provided that a higher priority channel does not require servicing.
3. The controller receives a request and performs four DMA transfers.
4. The controller arbitrates. After the controller receives a request for this channel, the flow continues if the channel has the highest priority.
5. The controller performs the remaining two DMA transfers.
6. The controller sets `dma_done[C]` HIGH for one `HFCORECLK` cycle and enters the arbitration process.

After task A completes, the host processor can configure the primary data structure for task C. This enables the controller to immediately switch to task C after task B completes, provided that a higher priority channel does not require servicing.

After the controller receives a new request for the channel and it has the highest priority then task B commences:

Task B
7. The controller performs four DMA transfers.
8. The controller arbitrates. After the controller receives a request for this channel, the flow continues if the channel has the highest priority.
9. The controller performs four DMA transfers.
10. The controller arbitrates. After the controller receives a request for this channel, the flow continues if the channel has the highest priority.
11. The controller performs the remaining four DMA transfers.
12. The controller sets `dma_done[C]` HIGH for one `HFCORECLK` cycle and enters the arbitration process.

After task B completes, the host processor can configure the alternate data structure for task D.

After the controller receives a new request for the channel and it has the highest priority then task C commences:

Task C
13. The controller performs two DMA transfers.
14. The controller sets `dma_done[C]` HIGH for one `HFCORECLK` cycle and enters the arbitration process.

After task C completes, the host processor can configure the primary data structure for task E.

After the controller receives a new request for the channel and it has the highest priority then task D commences:

Task D
15. The controller performs four DMA transfers.
16. The controller arbitrates. After the controller receives a request for this channel, the flow continues if the channel has the highest priority.
17. The controller performs the remaining DMA transfer.
18. The controller sets `dma_done[C]` HIGH for one `HFCORECLK` cycle and enters the arbitration process.

After the controller receives a new request for the channel and it has the highest priority then task E commences:

Task E
19. The controller performs four DMA transfers.
20. The controller arbitrates. After the controller receives a request for this channel, the flow continues if the channel has the highest priority.
21. The controller performs the remaining three DMA transfers.
22. The controller sets `dma_done[C]` HIGH for one `HFCORECLK` cycle and enters the arbitration process.

If the controller receives a new request for the channel and it has the highest priority then it attempts to start the next task. However, because the host processor has not configured the alternate data structure, and on completion of task D the controller set the cycle_ctrl bits to b000, then the ping-pong DMA transaction completes.

Note

You can also terminate the ping-pong DMA cycle in Figure 9.3 (p. 94) , if you configure task E to be a basic DMA cycle by setting the cycle_ctrl field to 3'b001.

9.4.2.3.5 Memory scatter-gather

In memory scatter-gather mode the controller receives an initial request and then performs four DMA transfers using the primary data structure. After this transfer completes, it starts a DMA cycle using the

alternate data structure. After this cycle completes, the controller performs another four DMA transfers using the primary data structure. The controller continues to switch from primary to alternate to primary… until either:

- the host processor configures the alternate data structure for a basic cycle
- it reads an invalid data structure.

Note

After the controller completes the N primary transfers it invalidates the primary data structure by setting the cycle_ctrl field to b000.

The controller only asserts `dma_done[C]` when the scatter-gather transaction completes using an auto-request cycle.

In scatter-gather mode, the controller uses the primary data structure to program the alternate data structure. Table 9.4 (p. 96) lists the fields of the channel_cfg memory location for the primary data structure, that you must program with constant values and those that can be user defined.

Table 9.4. channel_cfg for a primary data structure, in memory scatter-gather mode

Bit	Field	Value	Description
Constant-value fields:			
[31:30}	dst_inc	b10	Configures the controller to use word increments for the address
[29:28]	dst_size	b10	Configures the controller to use word transfers
[27:26]	src_inc	b10	Configures the controller to use word increments for the address
[25:24]	src_size	b10	Configures the controller to use word transfers
[17:14]	R_power	b0010	Configures the controller to perform four DMA transfers
[3]	next_useburst	0	For a memory scatter-gather DMA cycle, this bit must be set to zero
[2:0]	cycle_ctrl	b100	Configures the controller to perform a memory scatter-gather DMA cycle
User defined values:			
[23:21]	dst_prot_ctrl	-	Configures the state of `HPROT`[1] when the controller writes the destination data
[20:18]	src_prot_ctrl	-	Configures the state of `HPROT` when the controller reads the source data
[13:4]	n_minus_1	N [2]	Configures the controller to perform N DMA transfers, where N is a multiple of four

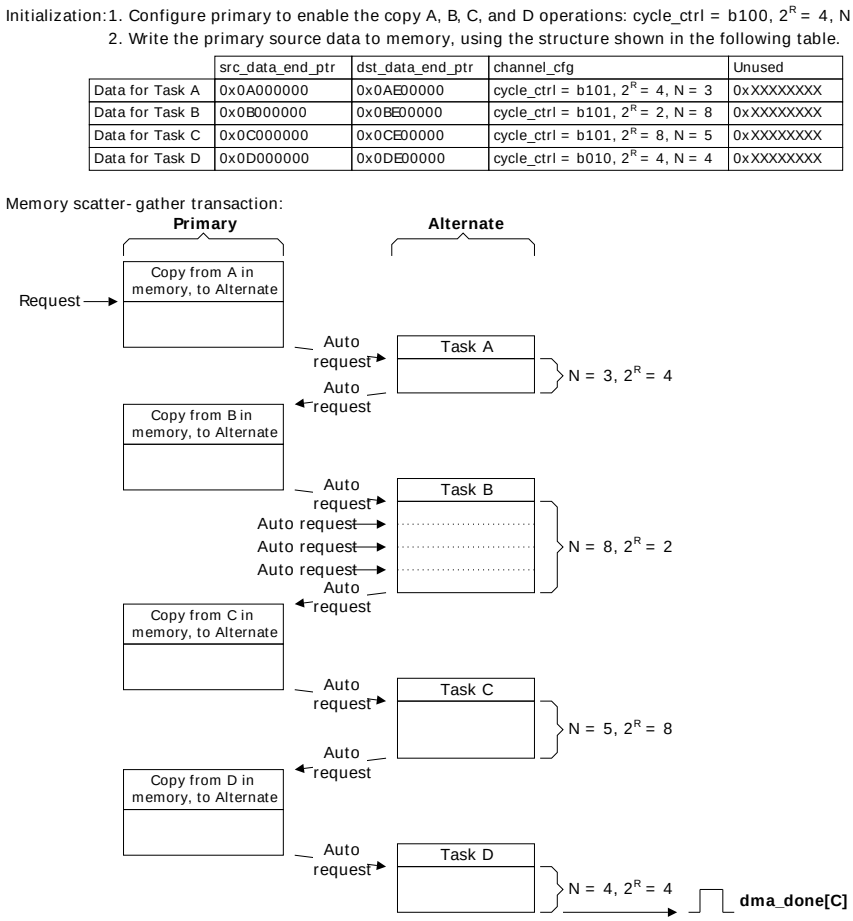
[1] ARM PL230 homepage [http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0417a/index.html]

[2] Because the R_power field is set to four, you must set N to be a multiple of four. The value given by N/4 is the number of times that you must configure the alternate data structure.

See Section 9.4.3.3 (p. 103) for more information.

Figure 9.4 (p. 97) shows a memory scatter-gather example.

Figure 9.4. Memory scatter-gather example



In Figure 9.4 (p. 97) :

Initialization

1. The host processor configures the primary data structure to operate in memory scatter-gather mode by setting cycle_ctrl to b100. Because a data structure for a single channel consists of four words then you must set 2^R to 4. In this example, there are four tasks and therefore N is set to 16.
2. The host processor writes the data structure for tasks A, B, C, and D to the memory locations that the primary src_data_end_ptr specifies.
3. The host processor enables the channel.

The memory scatter-gather transaction commences when the controller receives a request on `dma_req[ ]` or a manual request from the host processor. The transaction continues as follows:

Primary, copy A

1. After receiving a request, the controller performs four DMA transfers. These transfers write the alternate data structure for task A.
2. The controller generates an auto-request for the channel and then arbitrates.

Task A

3. The controller performs task A. After it completes the task, it generates an auto-request for the channel and then arbitrates.

Primary, copy B

4. The controller performs four DMA transfers. These transfers write the alternate data structure for task B.
5. The controller generates an auto-request for the channel and then arbitrates.

Task B

6. The controller performs task B. After it completes the task, it generates an auto-request for the channel and then arbitrates.

Primary, copy C

7. The controller performs four DMA transfers. These transfers write the alternate data structure for task C.

8. The controller generates an auto-request for the channel and then arbitrates.

Task C

9. The controller performs task C. After it completes the task, it generates an auto-request for the channel and then arbitrates.

Primary, copy D

10. The controller performs four DMA transfers. These transfers write the alternate data structure for task D.
11. The controller sets the cycle_ctrl bits of the primary data structure to b000, to indicate that this data structure is now invalid.
12. The controller generates an auto-request for the channel and then arbitrates.

Task D

13. The controller performs task D using an auto-request cycle.
14. The controller sets `dma_done[C]` HIGH for one HFCORECLK cycle and enters the arbitration process.

9.4.2.3.6 Peripheral scatter-gather

In peripheral scatter-gather mode the controller receives an initial request from a peripheral and then it performs four DMA transfers using the primary data structure. It then immediately starts a DMA cycle using the alternate data structure, without re-arbitrating.

Note

These are the only circumstances, where the controller does not enter the arbitration process after completing a transfer using the primary data structure.

After this cycle completes, the controller re-arbitrates and if the controller receives a request from the peripheral that has the highest priority then it performs another four DMA transfers using the primary data structure. It then immediately starts a DMA cycle using the alternate data structure, without re-arbitrating. The controller continues to switch from primary to alternate to primary… until either:

- the host processor configures the alternate data structure for a basic cycle
- it reads an invalid data structure.

Note

After the controller completes the N primary transfers it invalidates the primary data structure by setting the cycle_ctrl field to b000.

The controller asserts `dma_done[C]` when the scatter-gather transaction completes using a basic cycle.

In scatter-gather mode, the controller uses the primary data structure to program the alternate data structure. Table 9.5 (p. 98) lists the fields of the channel_cfg memory location for the primary data structure, that you must program with constant values and those that can be user defined.

Table 9.5. channel_cfg for a primary data structure, in peripheral scatter-gather mode

Bit	Field	Value	Description
Constant-value fields:			
[31:30]	dst_inc	b10	Configures the controller to use word increments for the address
[29:28]	dst_size	b10	Configures the controller to use word transfers
[27:26]	src_inc	b10	Configures the controller to use word increments for the address
[25:24]	src_size	b10	Configures the controller to use word transfers
[17:14]	R_power	b0010	Configures the controller to perform four DMA transfers
[2:0]	cycle_ctrl	b110	Configures the controller to perform a peripheral scatter-gather DMA cycle
User defined values:			
[23:21]	dst_prot_ctrl	-	Configures the state of HPROT when the controller writes the destination data

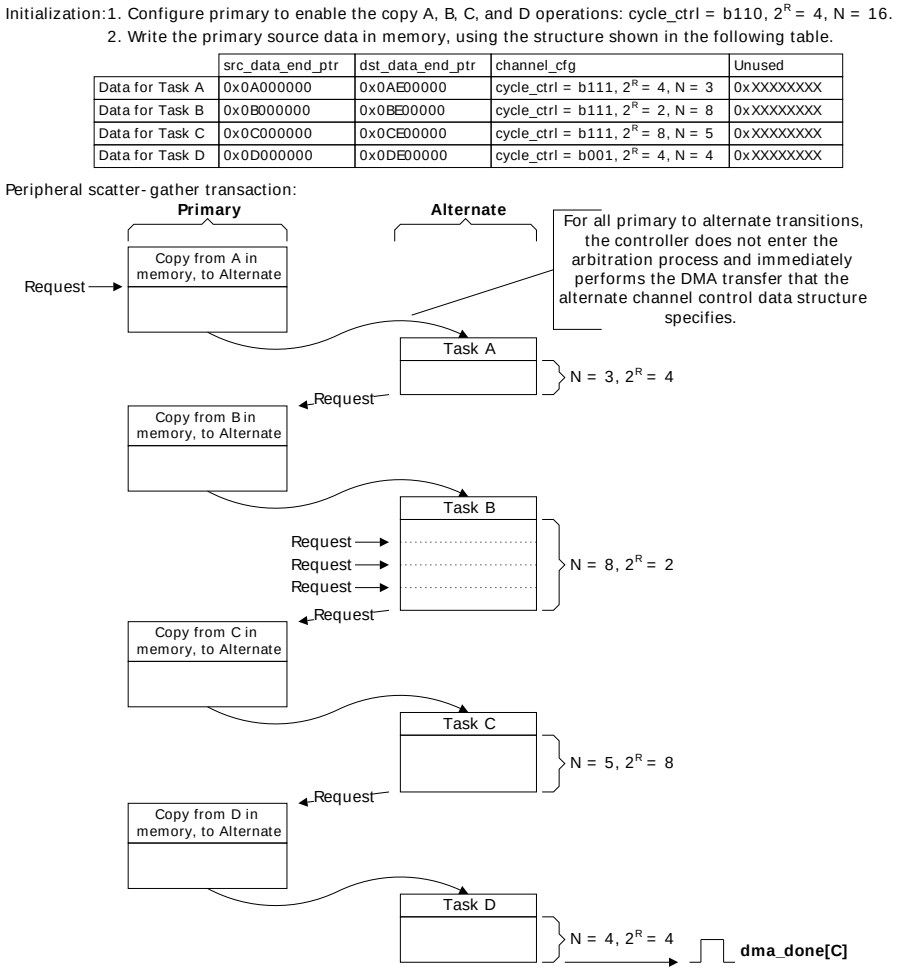
Bit	Field	Value	Description
[20:18]	src_prot_ctrl	-	Configures the state of HPROT when the controller reads the source data
[13:4]	n_minus_1	N [1]	Configures the controller to perform N DMA transfers, where N is a multiple of four
[3]	next_useburst	-	When set to 1, the controller sets the chnl_useburst_set [C] bit to 1 after the alternate transfer completes

[1]Because the R_power field is set to four, you must set N to be a multiple of four. The value given by N/4 is the number of times that you must configure the alternate data structure.

See Section 9.4.3.3 (p. 103) for more information.

Figure 9.5 (p. 99) shows a peripheral scatter-gather example.

Figure 9.5. Peripheral scatter-gather example



In Figure 9.5 (p. 99) :

Initialization

1. The host processor configures the primary data structure to operate in peripheral scatter-gather mode by setting cycle_ctrl to b110. Because a data structure for a single channel consists of four words then you must set 2^R to 4. In this example, there are four tasks and therefore N is set to 16.
2. The host processor writes the data structure for tasks A, B, C, and D to the memory locations that the primary src_data_end_ptr specifies.
3. The host processor enables the channel.

The peripheral scatter-gather transaction commences when the controller receives a request on `dma_req[]`. The transaction continues as follows:

Primary, copy A	1. After receiving a request, the controller performs four DMA transfers. These transfers write the alternate data structure for task A.
Task A	2. The controller performs task A. 3. After the controller completes the task it enters the arbitration process.

After the peripheral issues a new request and it has the highest priority then the process continues with:

Primary, copy B	4. The controller performs four DMA transfers. These transfers write the alternate data structure for task B.
Task B	5. The controller performs task B. To enable the controller to complete the task, the peripheral must issue a further three requests. 6. After the controller completes the task it enters the arbitration process.

After the peripheral issues a new request and it has the highest priority then the process continues with:

Primary, copy C	7. The controller performs four DMA transfers. These transfers write the alternate data structure for task C.
Task C	8. The controller performs task C. 9. After the controller completes the task it enters the arbitration process.

After the peripheral issues a new request and it has the highest priority then the process continues with:

Primary, copy D	10. The controller performs four DMA transfers. These transfers write the alternate data structure for task D. 11. The controller sets the cycle_ctrl bits of the primary data structure to b000, to indicate that this data structure is now invalid.
Task D	12. The controller performs task D using a basic cycle. 13. The controller sets `dma_done[C]` HIGH for one HFCORECLK cycle and enters the arbitration process.

9.4.2.4 Error signaling

If the controller detects an ERROR response on the AHB-Lite master interface, it:

- disables the channel that corresponds to the ERROR
- sets `dma_err` HIGH.

After the host processor detects that `dma_err` is HIGH, it must check which channel was active when the ERROR occurred. It can do this by:

1. Reading the DMA_CHENS register to create a list of disabled channels.

When a channel asserts `dma_done[ ]` then the controller disables the channel. The program running on the host processor must always keep a record of which channels have recently asserted their `dma_done[ ]` outputs.

2. It must compare the disabled channels list from step 1 (p. 100), with the record of the channels that have recently set their `dma_done[ ]` outputs. The channel with no record of `dma_done[C]` being set is the channel that the ERROR occurred on.

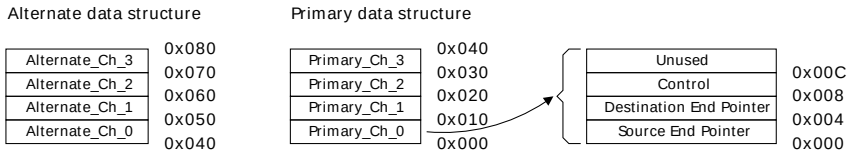
9.4.3 Channel control data structure

You must provide an area of system memory to contain the channel control data structure. This system memory must:

- provide a contiguous area of system memory that the controller and host processor can access
- have a base address that is an integer multiple of the total size of the channel control data structure.

Figure 9.6 (p. 101) shows the memory that the controller requires for the channel control data structure, when all 4 channels and the optional alternate data structure are in use.

Figure 9.6. Memory map for 4 channels, including the alternate data structure



This structure in Figure 9.6 (p. 101) uses bytes of system memory. The controller uses the lower 8 address bits to enable it to access all of the elements in the structure and therefore the base address must be at `0xXXXXXX00`.

You can configure the base address for the primary data structure by writing the appropriate value in the DMA_CTRLBASE register.

You do not need to set aside the full bytes if all dma channels are not used or if all alternate descriptors are not used. If, for example, only 4 channels are used and they only need the primary descriptors, then only 64 bytes need to be set aside.

Table 9.6 (p. 101) lists the address bits that the controller uses when it accesses the elements of the channel control data structure.

Table 9.6. Address bit settings for the channel control data structure

Address bits					
	[7]	[6]	[5]	[4]	[3:0]
	A	C[2]	C[1]	C[0]	0x0, 0x4, or 0x8

Where:

A — Selects one of the channel control data structures:
- A = 0 Selects the primary data structure.
- A = 1 Selects the alternate data structure.

C[2:0] — Selects the DMA channel.

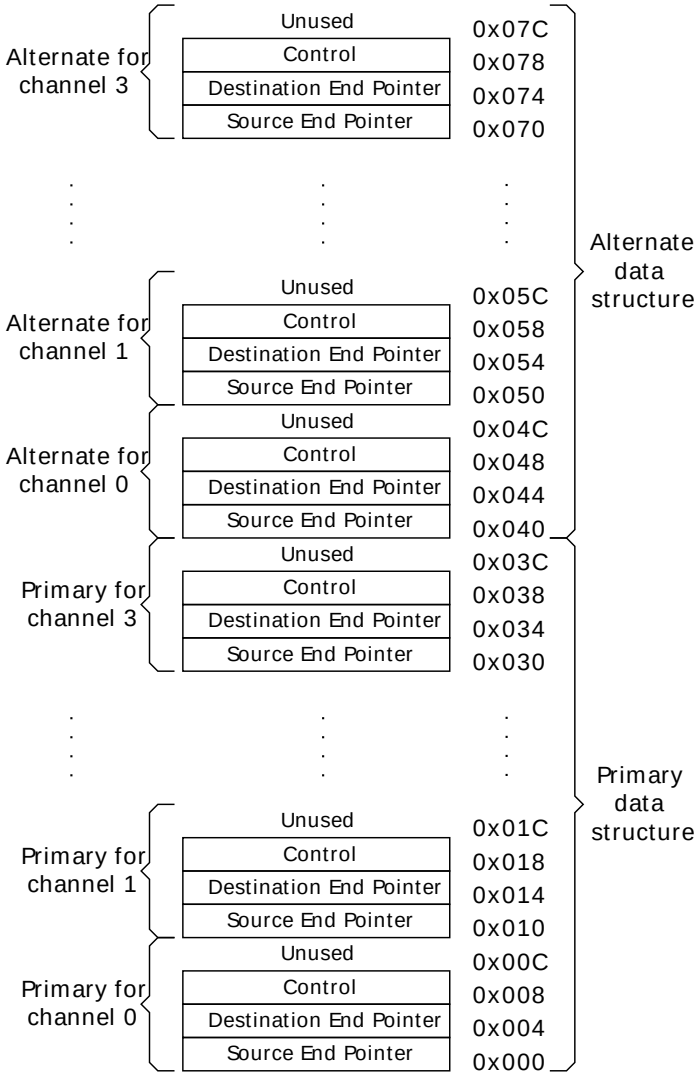
Address[3:0] — Selects one of the control elements:
- `0x0` Selects the source data end pointer.
- `0x4` Selects the destination data end pointer.
- `0x8` Selects the control data configuration.
- `0xC` The controller does not access this address location. If required, you can enable the host processor to use this memory location as system memory.

Note

It is not necessary for you to calculate the base address of the alternate data structure because the DMA_ALTCTRLBASE register provides this information.

Figure 9.7 (p. 102) shows a detailed memory map of the descriptor structure.

Figure 9.7. Detailed memory map for the 4 channels, including the alternate data structure



The controller uses the system memory to enable it to access two pointers and the control information that it requires for each channel. The following subsections will describe these 32-bit memory locations and how the controller calculates the DMA transfer address.

9.4.3.1 Source data end pointer

The src_data_end_ptr memory location contains a pointer to the end address of the source data. Figure 9.7 (p. 102) lists the bit assignments for this memory location.

Table 9.7. src_data_end_ptr bit assignments

Bit	Name	Description
[31:0]	src_data_end_ptr	Pointer to the end address of the source data

Before the controller can perform a DMA transfer, you must program this memory location with the end address of the source data. The controller reads this memory location when it starts a 2^R DMA transfer.

Note

The controller does not write to this memory location.

9.4.3.2 Destination data end pointer

The dst_data_end_ptr memory location contains a pointer to the end address of the destination data. Table 9.8 (p. 103) lists the bit assignments for this memory location.

Table 9.8. dst_data_end_ptr bit assignments

Bit	Name	Description
[31:0]	dst_data_end_ptr	Pointer to the end address of the destination data

Before the controller can perform a DMA transfer, you must program this memory location with the end address of the destination data. The controller reads this memory location when it starts a 2^R DMA transfer.

Note

The controller does not write to this memory location.

9.4.3.3 Control data configuration

For each DMA transfer, the channel_cfg memory location provides the control information for the controller. Figure 9.8 (p. 103) shows the bit assignments for this memory location.

Figure 9.8. channel_cfg bit assignments

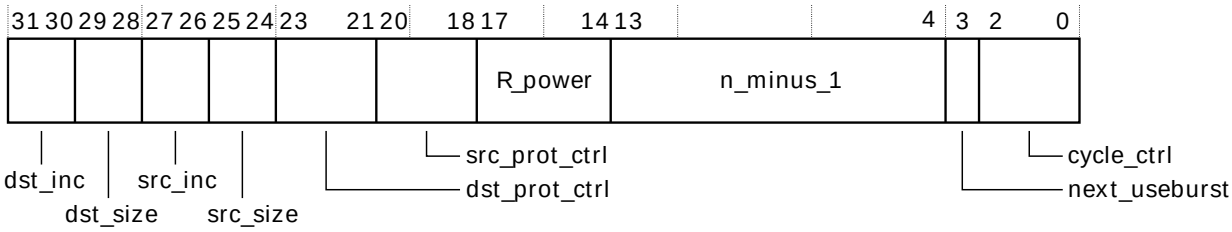


Table 9.9 (p. 103) lists the bit assignments for this memory location.

Table 9.9. channel_cfg bit assignments

Bit	Name	Description	
[31:30]	dst_inc	Destination address increment.	
		The address increment depends on the source data width as follows:	
		Source data width = byte	b00 = byte.
			b01 = halfword.
			b10 = word.
			b11 = no increment. Address remains set to the value that the dst_data_end_ptr memory location contains.
		Source data width = halfword	b00 = reserved.
			b01 = halfword.
			b10 = word.
			b11 = no increment. Address remains set to the value that the dst_data_end_ptr memory location contains.
		Source data width = word	b00 = reserved.
			b01 = reserved.
			b10 = word.

Bit	Name	Description
		b11 = no increment. Address remains set to the value that the dst_data_end_ptr memory location contains.
[29:28]	dst_size	Destination data size. ***Note*** You must set dst_size to contain the same value that src_size contains.
[27:26]	src_inc	Set the bits to control the source address increment. The address increment depends on the source data width as follows: Source data width = byte: b00 = byte. b01 = halfword. b10 = word. b11 = no increment. Address remains set to the value that the src_data_end_ptr memory location contains. Source data width = halfword: b00 = reserved. b01 = halfword. b10 = word. b11 = no increment. Address remains set to the value that the src_data_end_ptr memory location contains. Source data width = word: b00 = reserved. b01 = reserved. b10 = word. b11 = no increment. Address remains set to the value that the src_data_end_ptr memory location contains.
[25:24]	src_size	Set the bits to match the size of the source data: b00 = byte b01 = halfword b10 = word b11 = reserved.
[23:21]	dst_prot_ctrl	Set the bits to control the state of HPROT when the controller writes the destination data. Bit [23] This bit has no effect on the DMA. Bit [22] This bit has no effect on the DMA. Bit [21] Controls the state of HPROT as follows: 0 = HPROT is LOW and the access is non-privileged. 1 = HPROT is HIGH and the access is privileged.
[20:18]	src_prot_ctrl	Set the bits to control the state of HPROT when the controller reads the source data. Bit [20] This bit has no effect on the DMA. Bit [19] This bit has no effect on the DMA. Bit [18] Controls the state of HPROT as follows: 0 = HPROT is LOW and the access is non-privileged. 1 = HPROT is HIGH and the access is privileged.
[17:14]	R_power	Set these bits to control how many DMA transfers can occur before the controller re-arbitrates. The possible arbitration rate settings are: b0000 Arbitrates after each DMA transfer. b0001 Arbitrates after 2 DMA transfers. b0010 Arbitrates after 4 DMA transfers. b0011 Arbitrates after 8 DMA transfers. b0100 Arbitrates after 16 DMA transfers. b0101 Arbitrates after 32 DMA transfers. b0110 Arbitrates after 64 DMA transfers. b0111 Arbitrates after 128 DMA transfers.

Bit	Name	Description
		b1000 Arbitrates after 256 DMA transfers. b1001 Arbitrates after 512 DMA transfers. b1010 - b1111 Arbitrates after 1024 DMA transfers. This means that no arbitration occurs during the DMA transfer because the maximum transfer size is 1024.
[13:4]	n_minus_1	Prior to the DMA cycle commencing, these bits represent the total number of DMA transfers that the DMA cycle contains. You must set these bits according to the size of DMA cycle that you require. The 10-bit value indicates the number of DMA transfers, minus one. The possible values are: b000000000 = 1 DMA transfer b000000001 = 2 DMA transfers b000000010 = 3 DMA transfers b000000011 = 4 DMA transfers b000000100 = 5 DMA transfers . . . b111111111 = 1024 DMA transfers. The controller updates this field immediately prior to it entering the arbitration process. This enables the controller to store the number of outstanding DMA transfers that are necessary to complete the DMA cycle.
[3]	next_useburst	Controls if the chnl_useburst_set [C] bit is set to a 1, when the controller is performing a peripheral scatter-gather and is completing a DMA cycle that uses the alternate data structure. ***Note*** Immediately prior to completion of the DMA cycle that the alternate data structure specifies, the controller sets the chnl_useburst_set [C] bit to 0 if the number of remaining transfers is less than 2^R. The setting of the next_useburst bit controls if the controller performs an additional modification of the chnl_useburst_set [C] bit. In peripheral scatter-gather DMA cycle then after the DMA cycle that uses the alternate data structure completes , either: 0 = the controller does not change the value of the chnl_useburst_set [C] bit. If the chnl_useburst_set [C] bit is 0 then for all the remaining DMA cycles in the peripheral scatter-gather transaction, the controller responds to requests on `dma_req[ ]` and `dma_sreq[ ]`, when it performs a DMA cycle that uses an alternate data structure. 1 = the controller sets the chnl_useburst_set [C] bit to a 1. Therefore, for the remaining DMA cycles in the peripheral scatter-gather transaction, the controller only responds to requests on `dma_req[ ]`, when it performs a DMA cycle that uses an alternate data structure.
[2:0]	cycle_ctrl	The operating mode of the DMA cycle. The modes are: b000 Stop. Indicates that the data structure is invalid. b001 Basic. The controller must receive a new request, prior to it entering the arbitration process, to enable the DMA cycle to complete. b010 Auto-request. The controller automatically inserts a request for the appropriate channel during the arbitration process. This means that the initial request is sufficient to enable the DMA cycle to complete. b011 Ping-pong. The controller performs a DMA cycle using one of the data structures. After the DMA cycle completes, it performs a DMA cycle using the other data structure. After the DMA cycle completes and provided that the host processor has updated the original data structure, it performs a DMA cycle using the original data structure. The controller continues to perform DMA cycles until it either reads an invalid data structure or the host processor changes the cycle_ctrl bits to b001 or b010. See Section 9.4.2.3.4 (p. 93) . b100 Memory scatter/gather. See Section 9.4.2.3.5 (p. 95) . When the controller operates in memory scatter-gather mode, you must only use this value in the primary data structure. b101 Memory scatter/gather. See Section 9.4.2.3.5 (p. 95) . When the controller operates in memory scatter-gather mode, you must only use this value in the alternate data structure. b110 Peripheral scatter/gather. See Section 9.4.2.3.6 (p. 98) .

Bit	Name	Description
		When the controller operates in peripheral scatter-gather mode, you must only use this value in the primary data structure.
		b111 Peripheral scatter/gather. See Section 9.4.2.3.6 (p. 98) .
		When the controller operates in peripheral scatter-gather mode, you must only use this value in the alternate data structure.

At the start of a DMA cycle, or 2^R DMA transfer, the controller fetches the channel_cfg from system memory. After it performs 2^R, or N, transfers it stores the updated channel_cfg in system memory.

The controller does not support a dst_size value that is different to the src_size value. If it detects a mismatch in these values, it uses the src_size value for source and destination and when it next updates the n_minus_1 field, it also sets the dst_size field to the same as the src_size field.

After the controller completes the N transfers it sets the cycle_ctrl field to b000, to indicate that the channel_cfg data is invalid. This prevents it from repeating the same DMA transfer.

9.4.3.4 Address calculation

To calculate the source address of a DMA transfer, the controller performs a left shift operation on the n_minus_1 value by a shift amount that src_inc specifies, and then subtracts the resulting value from the source data end pointer. Similarly, to calculate the destination address of a DMA transfer, it performs a left shift operation on the n_minus_1 value by a shift amount that dst_inc specifies, and then subtracts the resulting value from the destination end pointer.

Depending on the value of src_inc and dst_inc, the source address and destination address can be calculated using the equations:

src_inc = b00 and dst_inc = b00	• source address = src_data_end_ptr - n_minus_1 • destination address = dst_data_end_ptr - n_minus_1.
src_inc = b01 and dst_inc = b01	• source address = src_data_end_ptr - (n_minus_1 << 1) • destination address = dst_data_end_ptr - (n_minus_1 << 1).
src_inc = b10 and dst_inc = b10	• source address = src_data_end_ptr - (n_minus_1 << 2) • destination address = dst_data_end_ptr - (n_minus_1 << 2).
src_inc = b11 and dst_inc = b11	• source address = src_data_end_ptr • destination address = dst_data_end_ptr.

Table 9.10 (p. 106) lists the destination addresses for a DMA cycle of six words.

Table 9.10. DMA cycle of six words using a word increment

Initial values of channel_cfg, prior to the DMA cycle				
src_size = b10, dst_inc = b10, n_minus_1 = b101, cycle_ctrl = 1				
DMA transfers	End Pointer	Count	Difference [1]	Address
	0x2AC	5	0x14	0x298
	0x2AC	4	0x10	0x29C
	0x2AC	3	0xC	0x2A0
	0x2AC	2	0x8	0x2A4
	0x2AC	1	0x4	0x2A8
	0x2AC	0	0x0	0x2AC
Final values of channel_cfg, after the DMA cycle				
src_size = b10, dst_inc = b10, n_minus_1 = 0, cycle_ctrl = 0				

[1] This value is the result of count being shifted left by the value of dst_inc.

Table 9.11 (p. 107) lists the destination addresses for a DMA transfer of 12 bytes using a halfword increment.

Table 9.11. DMA cycle of 12 bytes using a halfword increment

Initial values of channel_cfg, prior to the DMA cycle				
src_size = b00, dst_inc = b01, n_minus_1 = b1011, cycle_ctrl = 1, R_power = b11				
DMA transfers	End Pointer	Count	Difference [1]	Address
	0x5E7	11	0x16	0x5D1
	0x5E7	10	0x14	0x5D3
	0x5E7	9	0x12	0x5D5
	0x5E7	8	0x10	0x5D7
	0x5E7	7	0xE	0x5D9
	0x5E7	6	0xC	0x5DB
	0x5E7	5	0xA	0x5DD
	0x5E7	4	0x8	0x5DF
Values of channel_cfg after 2^R DMA transfers				
src_size = b00, dst_inc = b01, n_minus_1 = b011, cycle_ctrl = 1, R_power = b11				
DMA transfers	End Pointer	Count	Difference	Address
	0x5E7	3	0x6	0x5E1
	0x5E7	2	0x4	0x5E3
	0x5E7	1	0x2	0x5E5
	0x5E7	0	0x0	0x5E7
Final values of channel_cfg, after the DMA cycle				
src_size = b00, dst_inc = b01, n_minus_1 = 0, cycle_ctrl = 0 [2], R_power = b11				

[1] This value is the result of count being shifted left by the value of dst_inc.

[2] After the controller completes the DMA cycle it invalidates the channel_cfg memory location by clearing the cycle_ctrl field.

9.4.4 Interaction with the EMU

The DMA interacts with the Energy Management Unit (EMU) to allow transfers from , e.g., the LEUART to occur in EM2. The EMU can wake up the DMA sufficiently long to allow data transfers to occur. See section "DMA Support" in the LEUART documentation.

9.4.5 Interrupts

The PL230 dma_done[n:0] signals (one for each channel) as well as the dma_err signal, are available as interrupts to the Cortex-M0+ core. They are combined into one interrupt vector, DMA_INT. If the interrupt for the DMA is enabled in the ARM Cortex-M0+ core, an interrupt will be made if one or more of the interrupt flags in DMA_IF and their corresponding bits in DMA_IEN are set.

9.5 Examples

A basic example of how to program the DMA for transferring 42 bytes from the USART1 to memory location 0x20003420. Assumes that the channel 0 is currently disabled, and that the DMA_ALTCTRLBASE register has already been configured.

Example 9.1. DMA Transfer

1. Configure the channel select for using USART1 with DMA channel 0
   a. Write SOURCESEL=0b001101 and SIGSEL=XX to DMA_CHCTRL0
2. Configure the primary channel descriptor for DMA channel 0
   a. Write XX (read address of USART1) to src_data_end_ptr
   b. Write 0x20003420 + 40 to dst_data_end_ptr c
   c. Write these values to channel_cfg for channel 0:
      i. dst_inc=b01 (destination halfword address increment)
      ii. dst_size=b01 (halfword transfer size)
      iii. src_inc=b11 (no address increment for source)
      iv. src_size=01 (halfword transfer size)
      v. dst_prot_ctrl=000 (no cache/buffer/privilege)
      vi. src_prot_ctrl=000 (no cache/buffer/privilege)
      vii.R_power=b0000 (arbitrate after each DMA transfer)
      viiin_minus_1=d20 (transfer 21 halfwords)
      ix. next_useburst=b0 (not applicable)
      x. cycle_ctrl=b001 (basic operating mode)
3. Enable the DMA
   a. Write EN=1 to DMA_CONFIG
4. Disable the single requests for channel 0 (i.e., do not react to data available, wait for buffer full)
   a. Write DMA_CHUSEBURSTS[0]=1
5. Enable buffer-full requests for channel 0
   a. Write DMA_CHREQMASKC[0]=1
6. Use the primary data structure for channel 0
   a. Write DMA_CHALTC[0]=1
7. Enable channel 0
   a. Write DMA_CHENS[0]=1

9.6 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	DMA_STATUS	R	DMA Status Registers
0x004	DMA_CONFIG	W	DMA Configuration Register
0x008	DMA_CTRLBASE	RW	Channel Control Data Base Pointer Register
0x00C	DMA_ALTCTRLBASE	R	Channel Alternate Control Data Base Pointer Register
0x010	DMA_CHWAITSTATUS	R	Channel Wait on Request Status Register
0x014	DMA_CHSWREQ	W1	Channel Software Request Register
0x018	DMA_CHUSEBURSTS	RW1H	Channel Useburst Set Register
0x01C	DMA_CHUSEBURSTC	W1	Channel Useburst Clear Register
0x020	DMA_CHREQMASKS	RW1	Channel Request Mask Set Register
0x024	DMA_CHREQMASKC	W1	Channel Request Mask Clear Register
0x028	DMA_CHENS	RW1	Channel Enable Set Register
0x02C	DMA_CHENC	W1	Channel Enable Clear Register
0x030	DMA_CHALTS	RW1	Channel Alternate Set Register
0x034	DMA_CHALTC	W1	Channel Alternate Clear Register
0x038	DMA_CHPRIS	RW1	Channel Priority Set Register
0x03C	DMA_CHPRIC	W1	Channel Priority Clear Register
0x04C	DMA_ERRORC	RW	Bus Error Clear Register
0xE10	DMA_CHREQSTATUS	R	Channel Request Status
0xE18	DMA_CHSREQSTATUS	R	Channel Single Request Status
0x1000	DMA_IF	R	Interrupt Flag Register
0x1004	DMA_IFS	W1	Interrupt Flag Set Register
0x1008	DMA_IFC	W1	Interrupt Flag Clear Register
0x100C	DMA_IEN	RW	Interrupt Enable register
0x1100	DMA_CH0_CTRL	RW	Channel Control Register
0x1104	DMA_CH1_CTRL	RW	Channel Control Register
0x1108	DMA_CH2_CTRL	RW	Channel Control Register
0x110C	DMA_CH3_CTRL	RW	Channel Control Register
0x1110	DMA_CH4_CTRL	RW	Channel Control Register
0x1114	DMA_CH5_CTRL	RW	Channel Control Register

9.7 Register Description

9.7.1 DMA_STATUS - DMA Status Registers

Offset	Bit Position																															
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset												0x05														0x0						0
Access												R														R						R
Name												CHNUM														STATE						EN

Bit	Name	Reset	Access	Description
31:21	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
20:16	CHNUM	0x05	R	**Channel Number**
	Number of available DMA channels minus one.			
15:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:4	STATE	0x0	R	**Control Current State**
	State can be one of the following. Higher values (11-15) are undefined.			

Value	Mode	Description
0	IDLE	Idle
1	RDCHCTRLDATA	Reading channel controller data
2	RDSRCENDPTR	Reading source data end pointer
3	RDDSTENDPTR	Reading destination data end pointer
4	RDSRCDATA	Reading source data
5	WRDSTDATA	Writing destination data
6	WAITREQCLR	Waiting for DMA request to clear
7	WRCHCTRLDATA	Writing channel controller data
8	STALLED	Stalled
9	DONE	Done
10	PERSCATTRANS	Peripheral scatter-gather transition

3:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	EN	0	R	**DMA Enable Status**
	When this bit is 1, the DMA is enabled.			

9.7.2 DMA_CONFIG - DMA Configuration Register

Offset	Bit Position																															
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0					0
Access																											W					W
Name																											CHPROT					EN

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
5	CHPROT	0	W	**Channel Protection Control**
	Control whether accesses done by the DMA controller are privileged or not. When CHPROT = 1 then HPROT is HIGH and the access is privileged. When CHPROT = 0 then HPROT is LOW and the access is non-privileged.			
4:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	EN	0	W	**Enable DMA**
	Set this bit to enable the DMA controller.			

9.7.3 DMA_CTRLBASE - Channel Control Data Base Pointer Register

Offset	Bit Position																														
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	RW																															
Name	CTRLBASE																															

Bit	Name	Reset	Access	Description
31:0	CTRLBASE	0x00000000	RW	**Channel Control Data Base Pointer**
	The base pointer for a location in system memory that holds the channel control data structure. This register must be written to point to a location in system memory with the channel control data structure before the DMA can be used. Note that ctrl_base_ptr[8:0] must be 0.			

9.7.4 DMA_ALTCTRLBASE - Channel Alternate Control Data Base Pointer Register

Offset	Bit Position																														
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000080																															
Access	R																															
Name	ALTCTRLBASE																															

Bit	Name	Reset	Access	Description
31:0	ALTCTRLBASE	0x00000080	R	**Channel Alternate Control Data Base Pointer**

Bit	Name	Reset	Access	Description
	The base address of the alternate data structure. This register will read as DMA_CTRLBASE + 0x80.			

9.7.5 DMA_CHWAITSTATUS - Channel Wait on Request Status Register

Offset	Bit Position																														
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											1	1	1	1	1	1
Access																											R	R	R	R	R	R
Name																											CH5WAITSTATUS	CH4WAITSTATUS	CH3WAITSTATUS	CH2WAITSTATUS	CH1WAITSTATUS	CH0WAITSTATUS

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5WAITSTATUS	1	R	**Channel 5 Wait on Request Status**
	Status for wait on request for channel 5.			
4	CH4WAITSTATUS	1	R	**Channel 4 Wait on Request Status**
	Status for wait on request for channel 4.			
3	CH3WAITSTATUS	1	R	**Channel 3 Wait on Request Status**
	Status for wait on request for channel 3.			
2	CH2WAITSTATUS	1	R	**Channel 2 Wait on Request Status**
	Status for wait on request for channel 2.			
1	CH1WAITSTATUS	1	R	**Channel 1 Wait on Request Status**
	Status for wait on request for channel 1.			
0	CH0WAITSTATUS	1	R	**Channel 0 Wait on Request Status**
	Status for wait on request for channel 0.			

9.7.6 DMA_CHSWREQ - Channel Software Request Register

Offset	Bit Position																														
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											W1	W1	W1	W1	W1	W1
Name																											CH5SWREQ	CH4SWREQ	CH3SWREQ	CH2SWREQ	CH1SWREQ	CH0SWREQ

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5SWREQ	0	W1	**Channel 5 Software Request**
	Write 1 to this bit to generate a DMA request for this channel.			

Bit	Name	Reset	Access	Description
4	CH4SWREQ	0	W1	**Channel 4 Software Request**
	Write 1 to this bit to generate a DMA request for this channel.			
3	CH3SWREQ	0	W1	**Channel 3 Software Request**
	Write 1 to this bit to generate a DMA request for this channel.			
2	CH2SWREQ	0	W1	**Channel 2 Software Request**
	Write 1 to this bit to generate a DMA request for this channel.			
1	CH1SWREQ	0	W1	**Channel 1 Software Request**
	Write 1 to this bit to generate a DMA request for this channel.			
0	CH0SWREQ	0	W1	**Channel 0 Software Request**
	Write 1 to this bit to generate a DMA request for this channel.			

9.7.7 DMA_CHUSEBURSTS - Channel Useburst Set Register

Offset	Bit Position																											
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											RW1H	RW1H	RW1H	RW1H	RW1H	RW1H
Name																											CH5USEBURSTS	CH4USEBURSTS	CH3USEBURSTS	CH2USEBURSTS	CH1USEBURSTS	CH0USEBURSTS

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5USEBURSTS	0	RW1H	**Channel 5 Useburst Set**
	See description for channel 0.			
4	CH4USEBURSTS	0	RW1H	**Channel 4 Useburst Set**
	See description for channel 0.			
3	CH3USEBURSTS	0	RW1H	**Channel 3 Useburst Set**
	See description for channel 0.			
2	CH2USEBURSTS	0	RW1H	**Channel 2 Useburst Set**
	See description for channel 0.			
1	CH1USEBURSTS	0	RW1H	**Channel 1 Useburst Set**
	See description for channel 0.			
0	CH0USEBURSTS	0	RW1H	**Channel 0 Useburst Set**

Write to 1 to enable the useburst setting for this channel. Reading returns the useburst status. After the penultimate 2^R transfer completes, if the number of remaining transfers, N, is less than 2^R then the controller resets the chnl_useburst_set bit to 0. This enables you to complete the remaining transfers using dma_req[] or dma_sreq[]. In peripheral scatter-gather mode, if the next_useburst bit is set in channel_cfg then the controller sets the chnl_useburst_set[C] bit to a 1, when it completes the DMA cycle that uses the alternate data structure.

Value	Mode	Description
0	SINGLEANDBURST	Channel responds to both single and burst requests
1	BURSTONLY	Channel responds to burst requests only

9.7.8 DMA_CHUSEBURSTC - Channel Useburst Clear Register

Offset	Bit Position																													
0x01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											W1	W1	W1	W1	W1	W1
Name																											CH5USEBURSTC	CH4USEBURSTC	CH3USEBURSTC	CH2USEBURSTC	CH1USEBURSTC	CH0USEBURSTC

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5USEBURSTC	0	W1	**Channel 5 Useburst Clear**
	Write to 1 to disable useburst setting for this channel.			
4	CH4USEBURSTC	0	W1	**Channel 4 Useburst Clear**
	Write to 1 to disable useburst setting for this channel.			
3	CH3USEBURSTC	0	W1	**Channel 3 Useburst Clear**
	Write to 1 to disable useburst setting for this channel.			
2	CH2USEBURSTC	0	W1	**Channel 2 Useburst Clear**
	Write to 1 to disable useburst setting for this channel.			
1	CH1USEBURSTC	0	W1	**Channel 1 Useburst Clear**
	Write to 1 to disable useburst setting for this channel.			
0	CH0USEBURSTC	0	W1	**Channel 0 Useburst Clear**
	Write to 1 to disable useburst setting for this channel.			

9.7.9 DMA_CHREQMASKS - Channel Request Mask Set Register

Offset	Bit Position																													
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											RW1	RW1	RW1	RW1	RW1	RW1
Name																											CH5REQMASKS	CH4REQMASKS	CH3REQMASKS	CH2REQMASKS	CH1REQMASKS	CH0REQMASKS

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5REQMASKS	0	RW1	**Channel 5 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
4	CH4REQMASKS	0	RW1	**Channel 4 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
3	CH3REQMASKS	0	RW1	**Channel 3 Request Mask Set**

Bit	Name	Reset	Access	Description
	Write to 1 to disable peripheral requests for this channel.			
2	CH2REQMASKS	0	RW1	**Channel 2 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
1	CH1REQMASKS	0	RW1	**Channel 1 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			
0	CH0REQMASKS	0	RW1	**Channel 0 Request Mask Set**
	Write to 1 to disable peripheral requests for this channel.			

9.7.10 DMA_CHREQMASKC - Channel Request Mask Clear Register

Offset	Bit Position																													
0x024	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											W1	W1	W1	W1	W1	W1
Name																											CH5REQMASKC	CH4REQMASKC	CH3REQMASKC	CH2REQMASKC	CH1REQMASKC	CH0REQMASKC

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5REQMASKC	0	W1	**Channel 5 Request Mask Clear**
	Write to 1 to enable peripheral requests for this channel.			
4	CH4REQMASKC	0	W1	**Channel 4 Request Mask Clear**
	Write to 1 to enable peripheral requests for this channel.			
3	CH3REQMASKC	0	W1	**Channel 3 Request Mask Clear**
	Write to 1 to enable peripheral requests for this channel.			
2	CH2REQMASKC	0	W1	**Channel 2 Request Mask Clear**
	Write to 1 to enable peripheral requests for this channel.			
1	CH1REQMASKC	0	W1	**Channel 1 Request Mask Clear**
	Write to 1 to enable peripheral requests for this channel.			
0	CH0REQMASKC	0	W1	**Channel 0 Request Mask Clear**
	Write to 1 to enable peripheral requests for this channel.			

9.7.11 DMA_CHENS - Channel Enable Set Register

Offset	Bit Position																													
0x028	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											RW1	RW1	RW1	RW1	RW1	RW1
Name																											CH5ENS	CH4ENS	CH3ENS	CH2ENS	CH1ENS	CH0ENS

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5ENS	0	RW1	**Channel 5 Enable Set**
	Write to 1 to enable this channel. Reading returns the enable status of the channel.			
4	CH4ENS	0	RW1	**Channel 4 Enable Set**
	Write to 1 to enable this channel. Reading returns the enable status of the channel.			
3	CH3ENS	0	RW1	**Channel 3 Enable Set**
	Write to 1 to enable this channel. Reading returns the enable status of the channel.			
2	CH2ENS	0	RW1	**Channel 2 Enable Set**
	Write to 1 to enable this channel. Reading returns the enable status of the channel.			
1	CH1ENS	0	RW1	**Channel 1 Enable Set**
	Write to 1 to enable this channel. Reading returns the enable status of the channel.			
0	CH0ENS	0	RW1	**Channel 0 Enable Set**
	Write to 1 to enable this channel. Reading returns the enable status of the channel.			

9.7.12 DMA_CHENC - Channel Enable Clear Register

Offset	Bit Position																													
0x02C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											W1	W1	W1	W1	W1	W1
Name																											CH5ENC	CH4ENC	CH3ENC	CH2ENC	CH1ENC	CH0ENC

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5ENC	0	W1	**Channel 5 Enable Clear**
	Write to 1 to disable this channel. See also description for channel 0.			
4	CH4ENC	0	W1	**Channel 4 Enable Clear**
	Write to 1 to disable this channel. See also description for channel 0.			
3	CH3ENC	0	W1	**Channel 3 Enable Clear**
	Write to 1 to disable this channel. See also description for channel 0.			
2	CH2ENC	0	W1	**Channel 2 Enable Clear**
	Write to 1 to disable this channel. See also description for channel 0.			
1	CH1ENC	0	W1	**Channel 1 Enable Clear**
	Write to 1 to disable this channel. See also description for channel 0.			
0	CH0ENC	0	W1	**Channel 0 Enable Clear**
	Write to 1 to disable this channel. Note that the controller disables a channel, by setting the appropriate bit, when either it completes the DMA cycle, or it reads a channel_cfg memory location which has cycle_ctrl = b000, or an ERROR occurs on the AHB-Lite bus. A read from this field returns the value of CH0ENS from the DMA_CHENS register.			

9.7.13 DMA_CHALTS - Channel Alternate Set Register

Offset	Bit Position																													
0x030	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											RW1	RW1	RW1	RW1	RW1	RW1
Name																											CH5ALTS	CH4ALTS	CH3ALTS	CH2ALTS	CH1ALTS	CH0ALTS

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5ALTS	0	RW1	**Channel 5 Alternate Structure Set**
	Write to 1 to select the alternate structure for this channel.			
4	CH4ALTS	0	RW1	**Channel 4 Alternate Structure Set**
	Write to 1 to select the alternate structure for this channel.			
3	CH3ALTS	0	RW1	**Channel 3 Alternate Structure Set**
	Write to 1 to select the alternate structure for this channel.			
2	CH2ALTS	0	RW1	**Channel 2 Alternate Structure Set**
	Write to 1 to select the alternate structure for this channel.			
1	CH1ALTS	0	RW1	**Channel 1 Alternate Structure Set**
	Write to 1 to select the alternate structure for this channel.			
0	CH0ALTS	0	RW1	**Channel 0 Alternate Structure Set**
	Write to 1 to select the alternate structure for this channel.			

9.7.14 DMA_CHALTC - Channel Alternate Clear Register

Offset	Bit Position																													
0x034	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											W1	W1	W1	W1	W1	W1
Name																											CH5ALTC	CH4ALTC	CH3ALTC	CH2ALTC	CH1ALTC	CH0ALTC

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5ALTC	0	W1	**Channel 5 Alternate Clear**
	Write to 1 to select the primary structure for this channel.			
4	CH4ALTC	0	W1	**Channel 4 Alternate Clear**
	Write to 1 to select the primary structure for this channel.			
3	CH3ALTC	0	W1	**Channel 3 Alternate Clear**
	Write to 1 to select the primary structure for this channel.			
2	CH2ALTC	0	W1	**Channel 2 Alternate Clear**

Bit	Name	Reset	Access	Description
	Write to 1 to select the primary structure for this channel.			
1	CH1ALTC	0	W1	**Channel 1 Alternate Clear**
	Write to 1 to select the primary structure for this channel.			
0	CH0ALTC	0	W1	**Channel 0 Alternate Clear**
	Write to 1 to select the primary structure for this channel.			

9.7.15 DMA_CHPRIS - Channel Priority Set Register

Offset	Bit Position																													
0x038	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											RW1	RW1	RW1	RW1	RW1	RW1
Name																											CH5PRIS	CH4PRIS	CH3PRIS	CH2PRIS	CH1PRIS	CH0PRIS

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5PRIS	0	RW1	**Channel 5 High Priority Set**
	Write to 1 to obtain high priority for this channel. Reading returns the channel priority status.			
4	CH4PRIS	0	RW1	**Channel 4 High Priority Set**
	Write to 1 to obtain high priority for this channel. Reading returns the channel priority status.			
3	CH3PRIS	0	RW1	**Channel 3 High Priority Set**
	Write to 1 to obtain high priority for this channel. Reading returns the channel priority status.			
2	CH2PRIS	0	RW1	**Channel 2 High Priority Set**
	Write to 1 to obtain high priority for this channel. Reading returns the channel priority status.			
1	CH1PRIS	0	RW1	**Channel 1 High Priority Set**
	Write to 1 to obtain high priority for this channel. Reading returns the channel priority status.			
0	CH0PRIS	0	RW1	**Channel 0 High Priority Set**
	Write to 1 to obtain high priority for this channel. Reading returns the channel priority status.			

9.7.16 DMA_CHPRIC - Channel Priority Clear Register

Offset	Bit Position																													
0x03C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											W1	W1	W1	W1	W1	W1
Name																											CH5PRIC	CH4PRIC	CH3PRIC	CH2PRIC	CH1PRIC	CH0PRIC

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
5	CH5PRIC	0	W1	**Channel 5 High Priority Clear**
	Write to 1 to clear high priority for this channel.			
4	CH4PRIC	0	W1	**Channel 4 High Priority Clear**
	Write to 1 to clear high priority for this channel.			
3	CH3PRIC	0	W1	**Channel 3 High Priority Clear**
	Write to 1 to clear high priority for this channel.			
2	CH2PRIC	0	W1	**Channel 2 High Priority Clear**
	Write to 1 to clear high priority for this channel.			
1	CH1PRIC	0	W1	**Channel 1 High Priority Clear**
	Write to 1 to clear high priority for this channel.			
0	CH0PRIC	0	W1	**Channel 0 High Priority Clear**
	Write to 1 to clear high priority for this channel.			

9.7.17 DMA_ERRORC - Bus Error Clear Register

Offset	Bit Position																													
0x04C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																RW
Name																																ERRORC

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	ERRORC	0	RW	**Bus Error Clear**
	This bit is set high if an AHB bus error has occurred. Writing a 1 to this bit will clear the bit. If the error is deasserted at the same time as an error occurs on the bus, the error condition takes precedence and ERRORC remains asserted.			

9.7.18 DMA_CHREQSTATUS - Channel Request Status

Offset	Bit Position																													
0xE10	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											R	R	R	R	R	R
Name																											CH5REQSTATUS	CH4REQSTATUS	CH3REQSTATUS	CH2REQSTATUS	CH1REQSTATUS	CH0REQSTATUS

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5REQSTATUS	0	R	**Channel 5 Request Status**

Bit	Name	Reset	Access	Description
	When this bit is 1, it indicates that the peripheral connected as the input to this DMA channel is requesting the controller to service the DMA channel. The controller services the request by performing the DMA cycle using 2^R DMA transfers.			
4	CH4REQSTATUS	0	R	**Channel 4 Request Status**
	When this bit is 1, it indicates that the peripheral connected as the input to this DMA channel is requesting the controller to service the DMA channel. The controller services the request by performing the DMA cycle using 2^R DMA transfers.			
3	CH3REQSTATUS	0	R	**Channel 3 Request Status**
	When this bit is 1, it indicates that the peripheral connected as the input to this DMA channel is requesting the controller to service the DMA channel. The controller services the request by performing the DMA cycle using 2^R DMA transfers.			
2	CH2REQSTATUS	0	R	**Channel 2 Request Status**
	When this bit is 1, it indicates that the peripheral connected as the input to this DMA channel is requesting the controller to service the DMA channel. The controller services the request by performing the DMA cycle using 2^R DMA transfers.			
1	CH1REQSTATUS	0	R	**Channel 1 Request Status**
	When this bit is 1, it indicates that the peripheral connected as the input to this DMA channel is requesting the controller to service the DMA channel. The controller services the request by performing the DMA cycle using 2^R DMA transfers.			
0	CH0REQSTATUS	0	R	**Channel 0 Request Status**
	When this bit is 1, it indicates that the peripheral connected as the input to this DMA channel is requesting the controller to service the DMA channel. The controller services the request by performing the DMA cycle using 2^R DMA transfers.			

9.7.19 DMA_CHSREQSTATUS - Channel Single Request Status

Offset	Bit Position																													
0xE18	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											R	R	R	R	R	R
Name																											CH5SREQSTATUS	CH4SREQSTATUS	CH3SREQSTATUS	CH2SREQSTATUS	CH1SREQSTATUS	CH0SREQSTATUS

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5SREQSTATUS	0	R	**Channel 5 Single Request Status**
	When this bit is 1, it indicates that the peripheral connected as the input to this DMA channel is requesting the controller to service the DMA channel. The controller services the request by performing the DMA cycle using single DMA transfers.			
4	CH4SREQSTATUS	0	R	**Channel 4 Single Request Status**
	When this bit is 1, it indicates that the peripheral connected as the input to this DMA channel is requesting the controller to service the DMA channel. The controller services the request by performing the DMA cycle using single DMA transfers.			
3	CH3SREQSTATUS	0	R	**Channel 3 Single Request Status**
	When this bit is 1, it indicates that the peripheral connected as the input to this DMA channel is requesting the controller to service the DMA channel. The controller services the request by performing the DMA cycle using single DMA transfers.			
2	CH2SREQSTATUS	0	R	**Channel 2 Single Request Status**
	When this bit is 1, it indicates that the peripheral connected as the input to this DMA channel is requesting the controller to service the DMA channel. The controller services the request by performing the DMA cycle using single DMA transfers.			
1	CH1SREQSTATUS	0	R	**Channel 1 Single Request Status**
	When this bit is 1, it indicates that the peripheral connected as the input to this DMA channel is requesting the controller to service the DMA channel. The controller services the request by performing the DMA cycle using single DMA transfers.			
0	CH0SREQSTATUS	0	R	**Channel 0 Single Request Status**
	When this bit is 1, it indicates that the peripheral connected as the input to this DMA channel is requesting the controller to service the DMA channel. The controller services the request by performing the DMA cycle using single DMA transfers.			

9.7.20 DMA_IF - Interrupt Flag Register

Offset	Bit Position																													
0x1000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0																										0	0	0	0	0	0
Access	R																										R	R	R	R	R	R
Name	ERR																										CH5DONE	CH4DONE	CH3DONE	CH2DONE	CH1DONE	CH0DONE

Bit	Name	Reset	Access	Description
31	ERR	0	R	**DMA Error Interrupt Flag**
	This flag is set when an error has occurred on the AHB bus.			
30:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5DONE	0	R	**DMA Channel 5 Complete Interrupt Flag**
	Set when the DMA channel has completed its transfer. If the channel is disabled, the flag is set when there is a request for the channel.			
4	CH4DONE	0	R	**DMA Channel 4 Complete Interrupt Flag**
	Set when the DMA channel has completed its transfer. If the channel is disabled, the flag is set when there is a request for the channel.			
3	CH3DONE	0	R	**DMA Channel 3 Complete Interrupt Flag**
	Set when the DMA channel has completed its transfer. If the channel is disabled, the flag is set when there is a request for the channel.			
2	CH2DONE	0	R	**DMA Channel 2 Complete Interrupt Flag**
	Set when the DMA channel has completed its transfer. If the channel is disabled, the flag is set when there is a request for the channel.			
1	CH1DONE	0	R	**DMA Channel 1 Complete Interrupt Flag**
	Set when the DMA channel has completed its transfer. If the channel is disabled, the flag is set when there is a request for the channel.			
0	CH0DONE	0	R	**DMA Channel 0 Complete Interrupt Flag**
	Set when the DMA channel has completed its transfer. If the channel is disabled, the flag is set when there is a request for the channel.			

9.7.21 DMA_IFS - Interrupt Flag Set Register

Offset	Bit Position																													
0x1004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0																										0	0	0	0	0	0
Access	W1																										W1	W1	W1	W1	W1	W1
Name	ERR																										CH5DONE	CH4DONE	CH3DONE	CH2DONE	CH1DONE	CH0DONE

Bit	Name	Reset	Access	Description
31	ERR	0	W1	**DMA Error Interrupt Flag Set**
	Set to 1 to set DMA error interrupt flag.			
30:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5DONE	0	W1	**DMA Channel 5 Complete Interrupt Flag Set**
	Write to 1 to set the corresponding DMA channel complete interrupt flag.			
4	CH4DONE	0	W1	**DMA Channel 4 Complete Interrupt Flag Set**
	Write to 1 to set the corresponding DMA channel complete interrupt flag.			
3	CH3DONE	0	W1	**DMA Channel 3 Complete Interrupt Flag Set**

Bit	Name	Reset	Access	Description
	Write to 1 to set the corresponding DMA channel complete interrupt flag.			
2	CH2DONE	0	W1	**DMA Channel 2 Complete Interrupt Flag Set**
	Write to 1 to set the corresponding DMA channel complete interrupt flag.			
1	CH1DONE	0	W1	**DMA Channel 1 Complete Interrupt Flag Set**
	Write to 1 to set the corresponding DMA channel complete interrupt flag.			
0	CH0DONE	0	W1	**DMA Channel 0 Complete Interrupt Flag Set**
	Write to 1 to set the corresponding DMA channel complete interrupt flag.			

9.7.22 DMA_IFC - Interrupt Flag Clear Register

Offset	Bit Position																													
0x1008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0																										0	0	0	0	0	0
Access	W1																										W1	W1	W1	W1	W1	W1
Name	ERR																										CH5DONE	CH4DONE	CH3DONE	CH2DONE	CH1DONE	CH0DONE

Bit	Name	Reset	Access	Description
31	ERR	0	W1	**DMA Error Interrupt Flag Clear**
	Set to 1 to clear DMA error interrupt flag. Note that if an error happened, the Bus Error Clear Register must be used to clear the DMA.			
30:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5DONE	0	W1	**DMA Channel 5 Complete Interrupt Flag Clear**
	Write to 1 to clear the corresponding DMA channel complete interrupt flag.			
4	CH4DONE	0	W1	**DMA Channel 4 Complete Interrupt Flag Clear**
	Write to 1 to clear the corresponding DMA channel complete interrupt flag.			
3	CH3DONE	0	W1	**DMA Channel 3 Complete Interrupt Flag Clear**
	Write to 1 to clear the corresponding DMA channel complete interrupt flag.			
2	CH2DONE	0	W1	**DMA Channel 2 Complete Interrupt Flag Clear**
	Write to 1 to clear the corresponding DMA channel complete interrupt flag.			
1	CH1DONE	0	W1	**DMA Channel 1 Complete Interrupt Flag Clear**
	Write to 1 to clear the corresponding DMA channel complete interrupt flag.			
0	CH0DONE	0	W1	**DMA Channel 0 Complete Interrupt Flag Clear**
	Write to 1 to clear the corresponding DMA channel complete interrupt flag.			

9.7.23 DMA_IEN - Interrupt Enable register

Offset	Bit Position																													
0x100C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0																										0	0	0	0	0	0
Access	RW																										RW	RW	RW	RW	RW	RW
Name	ERR																										CH5DONE	CH4DONE	CH3DONE	CH2DONE	CH1DONE	CH0DONE

Bit	Name	Reset	Access	Description
31	ERR	0	RW	**DMA Error Interrupt Flag Enable**
	Set this bit to enable interrupt on AHB bus error.			
30:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5DONE	0	RW	**DMA Channel 5 Complete Interrupt Enable**
	Write to 1 to enable complete interrupt on this DMA channel. Clear to disable the interrupt.			
4	CH4DONE	0	RW	**DMA Channel 4 Complete Interrupt Enable**
	Write to 1 to enable complete interrupt on this DMA channel. Clear to disable the interrupt.			
3	CH3DONE	0	RW	**DMA Channel 3 Complete Interrupt Enable**
	Write to 1 to enable complete interrupt on this DMA channel. Clear to disable the interrupt.			
2	CH2DONE	0	RW	**DMA Channel 2 Complete Interrupt Enable**
	Write to 1 to enable complete interrupt on this DMA channel. Clear to disable the interrupt.			
1	CH1DONE	0	RW	**DMA Channel 1 Complete Interrupt Enable**
	Write to 1 to enable complete interrupt on this DMA channel. Clear to disable the interrupt.			
0	CH0DONE	0	RW	**DMA Channel 0 Complete Interrupt Enable**
	Write to 1 to enable complete interrupt on this DMA channel. Clear to disable the interrupt.			

9.7.24 DMA_CHx_CTRL - Channel Control Register

Offset	Bit Position																													
0x1100	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset											0x00																		0x0		
Access											RW																		RW		
Name											SOURCESEL																		SIGSEL		

Bit	Name	Reset	Access	Description
31:22	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
21:16	SOURCESEL	0x00	RW	**Source Select**
	Select input source to DMA channel.			

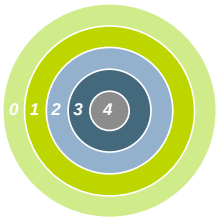
Value	Mode	Description
0b000000	NONE	No source selected
0b001000	ADC0	Analog to Digital Converter 0
0b001100	USART0	Universal Synchronous/Asynchronous Receiver/Transmitter 0
0b001101	USARTRF1	Universal Synchronous/Asynchronous Receiver/Transmitter 1
0b010000	LEUART0	Low Energy UART 0
0b010100	I2C0	I2C 0
0b011000	TIMER0	Timer 0
0b011001	TIMER1	Timer 1
0b011010	TIMER2	Timer 2
0b110000	MSC	
0b110001	AES	Advanced Encryption Standard Accelerator

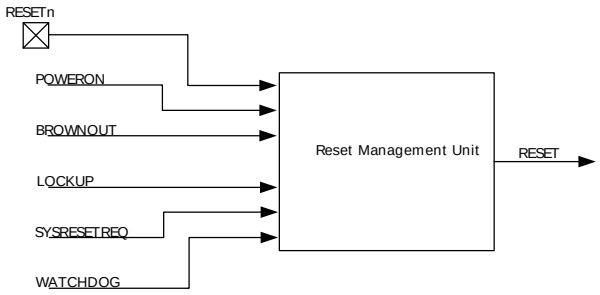
15:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3:0	SIGSEL	0x0	RW	**Signal Select**
	Select input signal to DMA channel.			

Value	Mode	Description
SOURCESEL = 0b000000 (NONE)		

Bit	Name	Reset	Access	Description
	Value	Mode		Description
	0bxxxx	OFF		Channel input selection is turned off
	SOURCESEL = 0b001000 (ADC0)			
	0b0000	ADC0SINGLE		ADC0SINGLE
	0b0001	ADC0SCAN		ADC0SCAN
	SOURCESEL = 0b001100 (USART0)			
	0b0000	USART0RXDATAV		USART0RXDATAV REQ/SREQ
	0b0001	USART0TXBL		USART0TXBL REQ/SREQ
	0b0010	USART0TXEMPTY		USART0TXEMPTY
	SOURCESEL = 0b001101 (USARTRF1)			
	0b0000	USARTRF1RXDATAV		USARTRF1RXDATAV REQ/SREQ
	0b0001	USARTRF1TXBL		USARTRF1TXBL REQ/SREQ
	0b0010	USARTRF1TXEMPTY		USARTRF1TXEMPTY
	0b0011	USARTRF1RXDATAVRIGHT		USARTRF1RXDATAVRIGHT REQ/SREQ
	0b0100	USARTRF1TXBLRIGHT		USARTRF1TXBLRIGHT REQ/SREQ
	SOURCESEL = 0b010000 (LEUART0)			
	0b0000	LEUART0RXDATAV		LEUART0RXDATAV
	0b0001	LEUART0TXBL		LEUART0TXBL
	0b0010	LEUART0TXEMPTY		LEUART0TXEMPTY
	SOURCESEL = 0b010100 (I2C0)			
	0b0000	I2C0RXDATAV		I2C0RXDATAV
	0b0001	I2C0TXBL		I2C0TXBL
	SOURCESEL = 0b011000 (TIMER0)			
	0b0000	TIMER0UFOF		TIMER0UFOF
	0b0001	TIMER0CC0		TIMER0CC0
	0b0010	TIMER0CC1		TIMER0CC1
	0b0011	TIMER0CC2		TIMER0CC2
	SOURCESEL = 0b011001 (TIMER1)			
	0b0000	TIMER1UFOF		TIMER1UFOF
	0b0001	TIMER1CC0		TIMER1CC0
	0b0010	TIMER1CC1		TIMER1CC1
	0b0011	TIMER1CC2		TIMER1CC2
	SOURCESEL = 0b011010 (TIMER2)			
	0b0000	TIMER2UFOF		TIMER2UFOF
	0b0001	TIMER2CC0		TIMER2CC0
	0b0010	TIMER2CC1		TIMER2CC1
	0b0011	TIMER2CC2		TIMER2CC2
	SOURCESEL = 0b110000 (MSC)			
	0b0000	MSCWDATA		MSCWDATA
	SOURCESEL = 0b110001 (AES)			
	0b0000	AESDATAWR		AESDATAWR
	0b0001	AESXORDATAWR		AESXORDATAWR
	0b0010	AESDATARD		AESDATARD
	0b0011	AESKEYWR		AESKEYWR

10 RMU - Reset Management Unit





Quick Facts

What?

The RMU ensures correct reset operation. It is responsible for connecting the different reset sources to the reset lines of the EZR32HG.

Why?

A correct reset sequence is needed to ensure safe and synchronous startup of the EZR32HG. In the case of error situations such as power supply glitches or software crash, the RMU provides proper reset and startup of the EZR32HG.

How?

The Power-on Reset and Brown-out Detector of the EZR32HG provides power line monitoring with exceptionally low power consumption. The cause of the reset may be read from a register, thus providing software with information about the cause of the reset.

10.1 Introduction

The RMU is responsible for handling the reset functionality of the EZR32HG.

10.2 Features

- Reset sources
  - Power-on Reset (POR)
  - Brown-out Detection (BOD) on the following power domains:
    - Regulated domain
    - Unregulated domain
    - Analog Power Domain 0 (AVDD0)
    - Analog Power Domain 1 (AVDD1)
  - RESETn pin reset
  - Watchdog reset
  - EM4 wakeup reset from pin
  - Software triggered reset (SYSRESETREQ)
  - Core LOCKUP condition
- EM4 Detection
- A software readable register indicates the cause of the last reset

10.3 Functional Description

The RMU monitors each of the reset sources of the EZR32HG. If one or more reset sources go active, the RMU applies reset to the EZR32HG. When the reset sources go inactive the EZR32HG starts up. At startup the EZR32HG loads the stack pointer and program entry point from memory, and starts execution.

As seen in Figure 10.1 (p. 126) the Power-on Reset, Brown-out Detectors, Watchdog timeout and RESETn pin all reset the whole system including the Debug Interface. A Core Lockup condition or a System reset request from software resets the whole system except the Debug Interface.

Whenever a reset source is active, the corresponding bit in the RMU_RSTCAUSE register is set. At startup the program code may investigate this register in order to determine the cause of the reset. The register must be cleared by software.

Figure 10.1. RMU Reset Input Sources and Connections.

10.3.1 RMU_RSTCAUSE Register

The RMU_RSTCAUSE register indicates the reason for the last reset. The register should be cleared after the value has been read at startup. Otherwise the register may indicate multiple causes for the reset at next startup.

The following procedure must be done to clear RMU_RSTCAUSE:

1. Write a 1 to RCCLR in RMU_CMD
2. Write a 1 to bit 0 in EMU_AUXCTRL
3. Write a 0 to bit 0 in EMU_AUXCTRL

RMU_RSTCAUSE should be interpreted according to Table 10.1 (p. 126). X bits are don't care. Notice that it is possible to have multiple reset causes. For example, an external reset and a watchdog reset may happen simultaneously.

Table 10.1. RMU Reset Cause Register Interpretation

Register Value	Cause
0bXXX XXXX XXX1	A Power-on Reset has been performed. X bits are don't care.
0bXXX 0XXX XX10	A Brown-out has been detected on the unregulated power.
0bXXX XXX0 0100	A Brown-out has been detected on the regulated power.
0bXXX XXXX 1X00	An external reset has been applied.
0bXXX XXX1 XX00	A watchdog reset has occurred.
0bXXX XX10 0000	A lockup reset has occurred.
0b000 01X0 0000	A system request reset has occurred.
0b000 1XX0 0XX0	The system has woken up from EM4.
0b001 1XX0 0XX0	The system has woken up from EM4 on an EM4 wakeup reset request from pin.
0b010 0000 0000	A Brown-out has been detected on Analog Power Domain 0 (AVDD0).
0b100 0000 0000	A Brown-out has been detected on Analog Power Domain 1 (AVDD1).

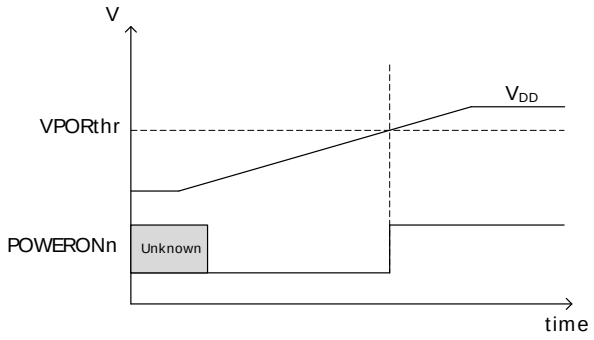
Note

When exiting EM4 with external reset, both the BODREGRST and BODUNREGRST in RSTCAUSE might be set (i.e. are invalid)

10.3.2 Power-On Reset (POR)

The POR ensures that the EZR32HG does not start up before the supply voltage V_{DD} has reached the threshold voltage VPORthr (see Device Datasheet Electrical Characteristics for details). Before the threshold voltage is reached, the EZR32HG is kept in reset state. The operation of the POR is illustrated in Figure 10.2 (p. 127), with the active low POWERONn reset signal. The reason for the “unknown” region is that the corresponding supply voltage is too low for any reliable operation.

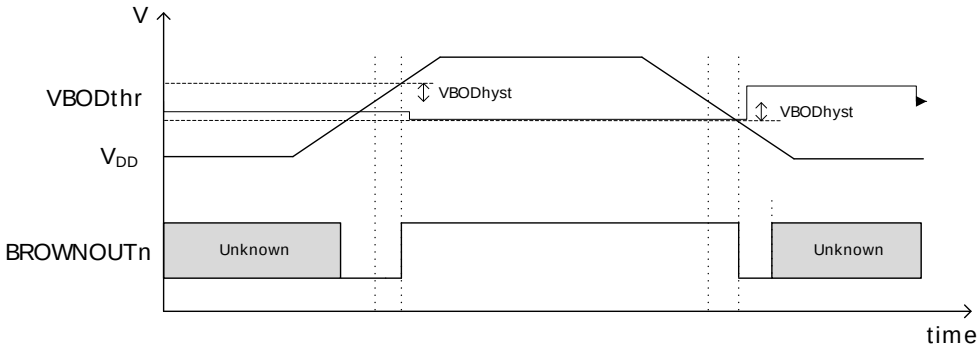
Figure 10.2. RMU Power-on Reset Operation



10.3.3 Brown-Out Detector Reset (BOD)

The EZR32HG has 4 brownout detectors, one for the unregulated 3.0 V power, one for the regulated internal power, one for Analog Power Domain 0 (AVDD0), and one for Analog Power Domain 1 (AVDD1). The BODs are constantly monitoring the voltages. Whenever the unregulated or regulated power drops below the VBODthr value (see Electrical Characteristics for details), or if the AVDD0 or AVDD1 drops below the voltage at the decouple pin (DEC), the corresponding active low BROWNOUTn line is held low. The BODs also include hysteresis, which prevents instability in the corresponding BROWNOUTn line when the supply is crossing the VBODthr limit or the AVDD bods drops below decouple pin (DEC). The operation of the BOD is illustrated in Figure 10.3 (p. 127) . The “unknown” regions are handled by the POR module.

Figure 10.3. RMU Brown-out Detector Operation



10.3.4 RESETn pin Reset

Forcing the RESETn pin low generates a reset of the EZR32HG. The RESETn pin includes an on-chip pull-up resistor, and can therefore be left unconnected if no external reset source is needed. Also connected to the RESETn line is a filter which prevents glitches from resetting the EZR32HG.

10.3.5 Watchdog Reset

The Watchdog circuit is a timer which (when enabled) must be cleared by software regularly. If software does not clear it, a Watchdog reset is activated. This functionality provides recovery from a software stalemate. Refer to the Watchdog section for specifications and description.

10.3.6 Lockup Reset

A Cortex-M0+ lockup is the result of the core being locked up because of an unrecoverable exception following the activation of the processor’s built-in system state protection hardware.

For more information about the Cortex-M0+ lockup conditions see the ARMv7-M Architecture Reference Manual. The Lockup reset does not reset the Debug Interface. Set the LOCKUPRDIS bit in the RMU_CTRL register in order to disable this reset source.

10.3.7 System Reset Request

Software may initiate a reset (e.g. if it finds itself in a non-recoverable state). By writing to the SYSRESETREQ bit in the Application Interrupt and Reset Control Register (see the Cortex-M0+ reference manual), a reset is issued. The SYSRESETREQ does not reset the Debug Interface.

10.3.8 EM4 Reset

Whenever EM4 is entered, the EM4RST bit is set. This bit enables the user to identify that the device has been in EM4. Upon wake-up this bit should be cleared by software.

10.3.9 EM4 Wakeup Reset

Whenever the system is woken up from EM4 on a pin wake-up request, the EM4WURST bit is set. This bit enables the user to identify that the device was woken up from EM4 using a pin wake-up request. Upon wake-up this bit should be cleared by software.

10.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	RMU_CTRL	RW	Control Register
0x004	RMU_RSTCAUSE	R	Reset Cause Register
0x008	RMU_CMD	W1	Command Register

10.5 Register Description

10.5.1 RMU_CTRL - Control Register

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																RW
Name																																LOCKUPRDIS

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	LOCKUPRDIS	0	RW	**Lockup Reset Disable**
	Set this bit to disable the LOCKUP signal (from the Cortex) from resetting the device.			

10.5.2 RMU_RSTCAUSE - Reset Cause Register

Offset	Bit Position																													
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																						0	0	0	0	0	0	0	0	0	0	0
Access																						R	R	R	R	R	R	R	R	R	R	R
Name																						BODAVDD1	BODAVDD0	EM4WURST	EM4RST	SYSREQRST	LOCKUPRST	WDOGRST	EXTRST	BODREGRST	BODUNREGRST	PORST

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	BODAVDD1	0	R	**AVDD1 Bod Reset**
	Set if analog power domain 1 brown out detector reset has been performed. Must be cleared by software. Please see Table 10.1 (p. 126) for details on how to interpret this bit.			
9	BODAVDD0	0	R	**AVDD0 Bod Reset**
	Set if analog power domain 0 brown out detector reset has been performed. Must be cleared by software. Please see Table 10.1 (p. 126) for details on how to interpret this bit.			
8	EM4WURST	0	R	**EM4 Wake-up Reset**
	Set if the system has been woken up from EM4 from a reset request from pin. Must be cleared by software. Please see Table 10.1 (p. 126) for details on how to interpret this bit.			

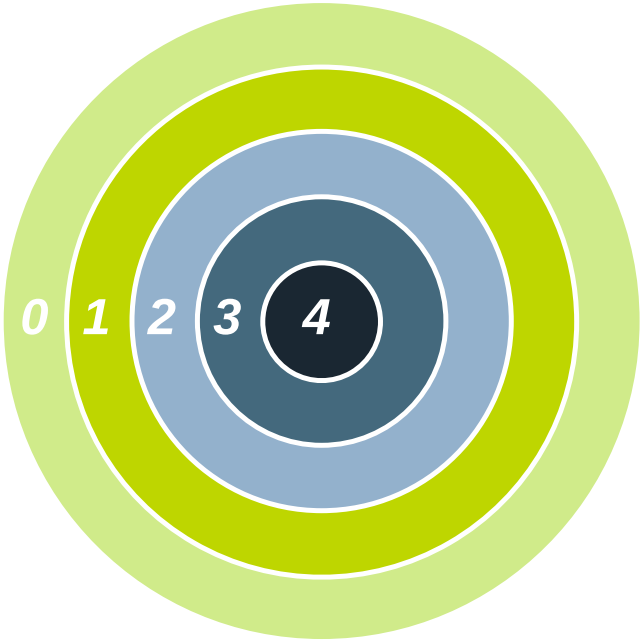
Bit	Name	Reset	Access	Description
7	EM4RST	0	R	**EM4 Reset**
	Set if the system has been in EM4. Must be cleared by software. Please see Table 10.1 (p. 126) for details on how to interpret this bit.			
6	SYSREQRST	0	R	**System Request Reset**
	Set if a system request reset has been performed. Must be cleared by software. Please see Table 10.1 (p. 126) for details on how to interpret this bit.			
5	LOCKUPRST	0	R	**LOCKUP Reset**
	Set if a LOCKUP reset has been requested. Must be cleared by software. Please see Table 10.1 (p. 126) for details on how to interpret this bit.			
4	WDOGRST	0	R	**Watchdog Reset**
	Set if a watchdog reset has been performed. Must be cleared by software. Please see Table 10.1 (p. 126) for details on how to interpret this bit.			
3	EXTRST	0	R	**External Pin Reset**
	Set if an external pin reset has been performed. Must be cleared by software. Please see Table 10.1 (p. 126) for details on how to interpret this bit.			
2	BODREGRST	0	R	**Brown Out Detector Regulated Domain Reset**
	Set if a regulated domain brown out detector reset has been performed. Must be cleared by software. Please see Table 10.1 (p. 126) for details on how to interpret this bit.			
1	BODUNREGRST	0	R	**Brown Out Detector Unregulated Domain Reset**
	Set if a unregulated domain brown out detector reset has been performed. Must be cleared by software. Please see Table 10.1 (p. 126) for details on how to interpret this bit.			
0	PORST	0	R	**Power On Reset**
	Set if a power on reset has been performed. Must be cleared by software. Please see Table 10.1 (p. 126) for details on how to interpret this bit.			

10.5.3 RMU_CMD - Command Register

Offset	Bit Position																														
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																W1
Name																																RCCLR

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	RCCLR	0	W1	**Reset Cause Clear**
	Set this bit to clear the LOCKUPRST and SYSREQRST bits in the RMU_RSTCAUSE register. Use the HRCCLR bit in the EMU_AUXCTRL register to clear the remaining bits.			

11 EMU - Energy Management Unit



Quick Facts

What?

The EMU (Energy Management Unit) handles the different low energy modes in the EZR32HG microcontrollers.

Why?

The need for performance and peripheral functions varies over time in most applications. By efficiently scaling the available resources in real-time to match the demands of the application, the energy consumption can be kept at a minimum.

How?

With a broad selection of energy modes, a high number of low-energy peripherals available even in EM2, and short wake-up time (2 µs from EM2 and EM3), applications can dynamically minimize energy consumption during program execution.

11.1 Introduction

The Energy Management Unit (EMU) manages all the low energy modes (EM) in EZR32HG microcontrollers. Each energy mode manages if the CPU and the various peripherals are available. The energy modes range from EM0 to EM4, where EM0, also called run mode, enables the CPU and all peripherals. The lowest recoverable energy mode, EM3, disables the CPU and most peripherals while maintaining wake-up and RAM functionality. EM4 disables everything except the POR, pin reset and optionally GPIO state retention and EM4 reset wakeup request.

The various energy modes differ in:

- Energy consumption
- CPU activity
- Reaction time
- Wake-up triggers
- Active peripherals
- Available clock sources

Low energy modes EM1 to EM4 are enabled through the application software. In EM1-EM3, a range of wake-up triggers return the microcontroller back to EM0. EM4 can only return to EM0 by power on reset, external pin reset or EM4 GPIO wakeup request.

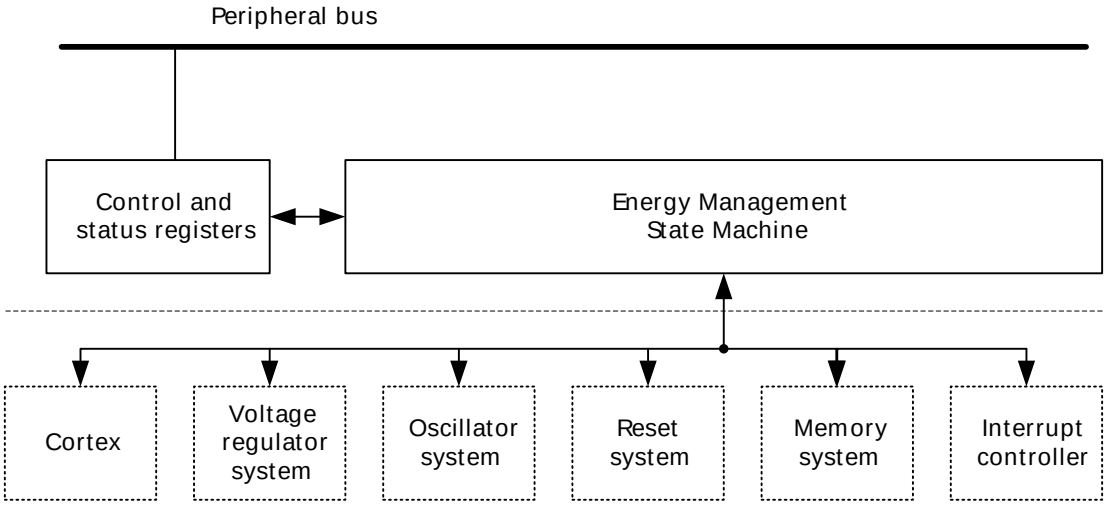
11.2 Features

- Energy Mode control from software
- Flexible wakeup from low energy modes
- Low wakeup time

11.3 Functional Description

The Energy Management Unit (EMU) is responsible for managing the wide range of energy modes available in EZR32HG. An overview of the EMU module is shown in Figure 11.1 (p. 132) .

Figure 11.1. EMU Overview



The EMU is available as a peripheral on the peripheral bus. The energy management state machine is triggered from the Cortex-M0+ and controls the internal voltage regulators, oscillators, memories and interrupt systems in the low energy modes. Events from the interrupt or reset systems can in turn cause the energy management state machine to return to its active state. This is further described in the following sections.

11.3.1 Energy Modes

There are five main energy modes available in EZR32HG, called Energy Mode 0 (EM0) through Energy Mode 4 (EM4). EM0, also called the active mode, is the energy mode in which any peripheral function can be enabled and the Cortex-M0+ core is executing instructions. EM1 through EM4, also called low energy modes, provide a selection of reduced peripheral functionality that also lead to reduced energy consumption, as described below.

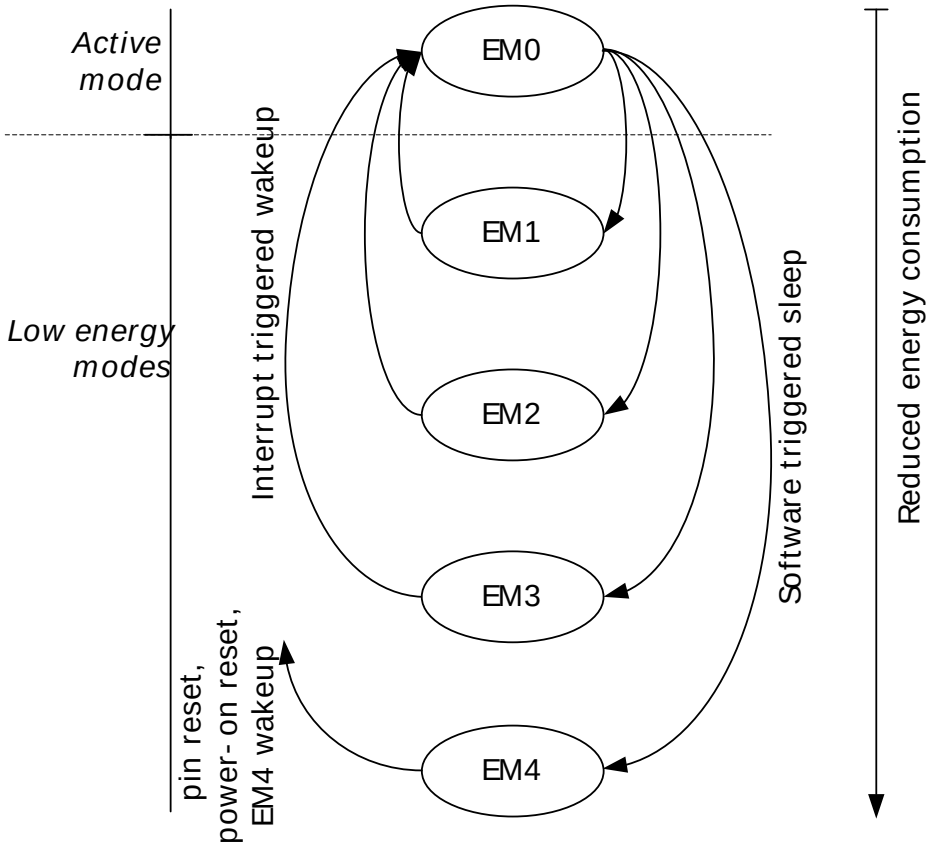
Figure 11.2 (p. 133) shows the transitions between different energy modes. After reset the EMU will always start in EM0. A transition from EM0 to another energy mode is always initiated by software. EM0 is the highest activity mode, in which all functionality is available. EM0 is therefore also the mode with highest energy consumption.

The low energy modes EM1 through EM4 result in less functionality being available, and therefore also reduced energy consumption. The Cortex-M0+ is not executing instructions in any low energy mode. Each low energy mode provides different energy consumptions associated with it, for example because a different set of peripherals are enabled or because these peripherals are configured differently.

A transition from EM0 to a low energy mode can only be triggered by software.

A transition from EM1 – EM3 to EM0 can be triggered by an enabled interrupt or event. In addition, a chip reset will return the device to EM0. A transition from EM4 can only be triggered by a pin reset, power-on reset, or EM4 GPIO wakeup request.

Figure 11.2. EMU Energy Mode Transitions



No direct transitions between EM1, EM2 or EM3 are available, as can also be seen from Figure 11.2 (p. 133) . Instead, a wakeup will transition back to EM0, in which software can enter any other low energy mode. An overview of the supported energy modes and the functionality available in each mode is shown in Table 11.1 (p. 134) . Most peripheral functionality indicated as "On" in a particular energy mode can also be turned off from software in order to save further energy.

Table 11.1. EMU Energy Mode Overview

	EM0[1]	EM1[2]	EM2[2]	EM3[2]	EM4[2]
Wakeup time to EM0	-	-	2 µs	2 µs	160 µs
MCU clock tree	On	-	-	-	-
High frequency peripheral clock trees	On	On	-	-	-
Core voltage regulator	On	On	-	-	-
High frequency oscillator	On	On	-	-	-
I^2C full functionality	On	On	-	-	-
Low frequency peripheral clock trees	On	On	On	-	-
Low frequency oscillator	On	On	On	-	-
Real Time Counter	On	On	On	On[3]	-
LEUART	On	On	On	-	-
PCNT	On	On	On	On	-
ACMP	On	On	On	On	-
I^2C receive address recognition	On	On	On	On	-
IDAC	On	On	On	On	-
Watchdog	On	On	On	On[3]	-
Pin interrupts	On	On	On	On	-
RAM voltage regulator/RAM retention	On	On	On	On	-
Brown Out Reset	On	On	On	On	-
Power On Reset	On	On	On	On	On
Pin Reset	On	On	On	On	On
GPIO state retention	On	On	On	On	On
EM4 Reset Wakeup Request	-	-	-	-	On

[1]Energy Mode 0/Active Mode
[2]Energy Mode 1/2/3/4
[3]When the 1 kHz ULFRCO is selected

The different Energy Modes are summarized in the following sections.

11.3.1.1 EM0

- The high frequency oscillator is active
- High frequency clock trees are active
- All peripheral functionality is available

11.3.1.2 EM1

- The high frequency oscillator is active
- MCU clock tree is inactive
- High frequency peripheral clock trees are active
- All peripheral functionality is available

11.3.1.3 EM2

- The high frequency oscillator is inactive

- The high frequency peripheral and MCU clock trees are inactive
- The low frequency oscillator and clock trees are active
- Low frequency peripheral functionality is available
- Wakeup through peripheral interrupt or asynchronous pin interrupt
- RAM and register values are preserved

11.3.1.4 EM3

- Both high and low frequency oscillators and clock trees are inactive
- Wakeup through asynchronous pin interrupts, I^2C address recognition or ACMP edge interrupt
- Watchdog and some low frequency peripherals available when ULFRCO (1 kHz clock) has been selected
- All other peripheral functionality is disabled
- RAM and register values are preserved

11.3.1.5 EM4

- All oscillators and regulators are inactive
- RAM and register values are not preserved
- Optional GPIO state retention
- Wakeup from external pin reset or pins that support EM4 wakeup

11.3.2 Entering a Low Energy Mode

A low energy mode is entered by first configuring the desired Energy Mode through the EMU_CTRL register and the SLEEPDEEP bit in the Cortex-M0+ System Control Register, see Table 11.2 (p. 135). A Wait For Interrupt (WFI) or Wait For Event (WFE) instruction from the Cortex-M0+ triggers the transition into a low energy mode.

The transition into a low energy mode can optionally be delayed until the lowest priority Interrupt Service Routine (ISR) is exited, if the SLEEPONEXIT bit in the Cortex-M0+ System Control Register is set.

Entering the lowest energy mode, EM4, is done by writing a sequence to the EM4CTRL bitfield in the EMU_CTRL register. Writing a zero to the EM4CTRL bitfield will restart the power sequence. EM2BLOCK prevents the EMU to enter EM2 or lower, and it will instead enter EM1.

EM3 is equal to EM2, except that the LFACLK/LFBCLK are disabled in EM3. The LFACLK/LFBCLK must be disabled by the user before entering low energy mode.

The EMVREG bit in EMU_CTRL can be used to prevent the voltage regulator from being turned off in low energy modes. The device will then essentially stay in EM1 (with HF oscillators disabled) when entering a low energy mode. Note that if a DMA transfer is initiated in this mode, the HF-oscillators will start and remain enabled until the device is woken up from an EM2 interrupt.

Table 11.2. EMU Entering a Low Energy Mode

Low Energy Mode	EM4CTRL	EMVREG	EM2BLOCK	SLEEPDEEP	Cortex-M0+ Instruction
EM1	0	x	x	0	WFI or WFE
EM2	0	0	0	1	WFI or WFE
EM4	Write sequence: 2, 3, 2, 3, 2, 3, 2, 3, 2	x	x	x	x

('x' means don't care)

11.3.3 Leaving a Low Energy Mode

In each low energy mode a selection of peripheral units are available, and software can either enable or disable the functionality. Enabled interrupts that can cause wakeup from a low energy mode are shown in Table 11.3 (p. 136) . The wakeup triggers always return the EZR32 to EM0. Additionally, any reset source will return to EM0.

Table 11.3. EMU Wakeup Triggers from Low Energy Modes

Peripheral	Wakeup Trigger	EM0[1]	EM1[2]	EM2[2]	EM3[2]	EM4[2]
RTC	Any enabled interrupt	-	Yes	Yes	Yes[3]	-
USART	Receive / transmit	-	Yes	-	-	-
LEUART	Receive / transmit	-	Yes	Yes	-	-
I^2C	Any enabled interrupt	-	Yes	-	-	-
I^2C	Receive address recognition	-	Yes	Yes	Yes	-
TIMER	Any enabled interrupt	-	Yes	-	-	-
CMU	Any enabled interrupt	-	Yes	-	-	-
DMA	Any enabled interrupt	-	Yes	-	-	-
MSC	Any enabled interrupt	-	Yes	-	-	-
ADC	Any enabled interrupt	-	Yes	-	-	-
AES	Any enabled interrupt	-	Yes	-	-	-
PCNT	Any enabled interrupt	-	Yes	Yes	Yes[4]	-
ACMP	Any enabled edge interrupt	-	Yes	Yes	Yes	-
VCMP	Any enabled edge interrupt	-	Yes	Yes	Yes	-
Pin interrupts	Asynchronous	-	Yes	Yes	Yes	-
Pin	Reset	-	Yes	Yes	Yes	Yes
EM4 wakeup on supported pins	Asynchronous	-	-	-	-	Yes
Power	Cycle Off/On		Yes	Yes	Yes	Yes

[1]Energy Mode 0/Active Mode
[2]Energy mode 1/2/3/4
[3]When the 1 kHz ULFRCO is selected
[4]When using an external clock

11.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	EMU_CTRL	RW	Control Register
0x008	EMU_LOCK	RW	Configuration Lock Register
0x024	EMU_AUXCTRL	RW	Auxiliary Control Register

11.5 Register Description

11.5.1 EMU_CTRL - Control Register

Offset	Bit Position																															
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																													0x0		0	0
Access																													RW		RW	RW
Name																													EM4CTRL		EM2BLOCK	EMVREG

Bit	Name	Reset	Access	Description
31:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3:2	EM4CTRL	0x0	RW	**Energy Mode 4 Control**
	This register is used to enter Energy Mode 4, in which the device only wakes up from an external pin reset, from a power cycle, or EM4 wakeup reset request. Energy Mode 4 is entered when the EM4 sequence is written to this bitfield.			
1	EM2BLOCK	0	RW	**Energy Mode 2 Block**
	This bit is used to prevent the MCU to enter Energy Mode 2 or lower.			
0	EMVREG	0	RW	**Energy Mode Voltage Regulator Control**
	Control the voltage regulator in low energy modes 2 and 3.			

Value	Mode	Description
0	REDUCED	Reduced voltage regulator drive strength in EM2 and EM3.
1	FULL	Full voltage regulator drive strength in EM2 and EM3.

11.5.2 EMU_LOCK - Configuration Lock Register

Offset	Bit Position																															
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	LOCKKEY															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
15:0	LOCKKEY	0x0000	RW	**Configuration Lock Key**

Write any other value than the unlock code to lock all EMU registers, except the interrupt registers, from editing. Write the unlock code to unlock. When reading the register, bit 0 is set when the lock is enabled.

Mode	Value	Description
Read Operation		
UNLOCKED	0	EMU registers are unlocked.
LOCKED	1	EMU registers are locked.
Write Operation		
LOCK	0	Lock EMU registers.
UNLOCK	0xADE8	Unlock EMU registers.

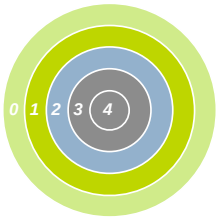
11.5.3 EMU_AUXCTRL - Auxiliary Control Register

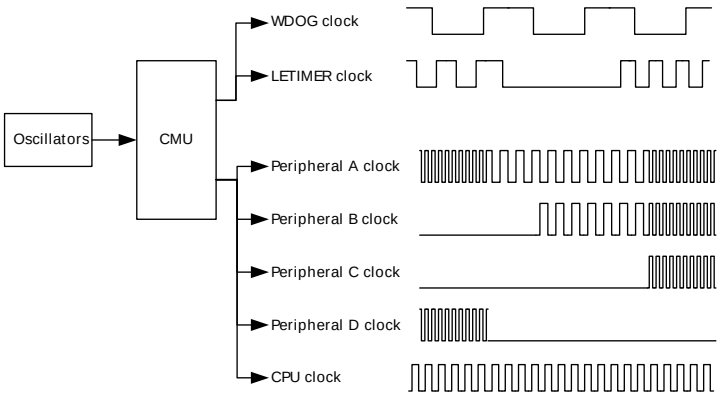
Offset	Bit Position																													
0x024	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																RW
Name																																HRCCLR

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	HRCCLR	0	RW	**Hard Reset Cause Clear**

Write to 1 and then 0 to clear the POR, BOD and WDOG reset cause register bits. See also the Reset Management Unit (RMU).

12 CMU - Clock Management Unit





Quick Facts

What?

The CMU controls oscillators and clocks. EZR32HG supports several different oscillators with minimized power consumption and short start-up time. An additional separate RC oscillator is used for flash programming. The CMU also has HW support for calibration of RC oscillators.

Why?

Oscillators and clocks contribute significantly to the power consumption of the MCU. With the low power oscillators combined with the flexible clock control scheme, it is possible to minimize the energy consumption in any given application.

How?

The CMU can configure different clock sources, enable/disable clocks to peripherals on an individual basis and set the prescaler for the different clocks. The short oscillator start-up times makes duty-cycling between active mode and the different low energy modes (EM2-EM4) very efficient. The calibration feature ensures high accuracy RC oscillators. Several interrupts are available to avoid CPU polling of flags.

12.1 Introduction

The Clock Management Unit (CMU) is responsible for controlling the oscillators and clocks on-board the EZR32HG. The CMU provides the capability to turn on and off the clock on an individual basis to all peripheral modules in addition to enable/disable and configure the available oscillators. The high degree of flexibility enables software to minimize energy consumption in any specific application by not wasting power on peripherals and oscillators that are inactive.

12.2 Features

- Multiple clock sources available:
  - 1-21 MHz High Frequency RC Oscillator (HFRCO)
  - 4-25 MHz High Frequency Crystal Oscillator (HFXO)
  - 32768 Hz Low Frequency RC Oscillator (LFRCO)
  - 32768 Hz Low Frequency Crystal Oscillator (LFXO)
  - 1000 Hz Ultra Low Frequency RC Oscillator (ULFRCO)
  - 48/24 MHz Universal Serial High Frequency RC Oscillator (USHFRCO)
- Low power oscillators
- Low start-up times
- Separate prescaler for High Frequency Core Clocks (HFCORECLK) and Peripheral Clocks (HFPERCLK)

- Individual clock prescaler selection for each Low Energy Peripheral
- Clock Gating on an individual basis to core modules and all peripherals
- Selectable clocks can be output on two pins for use externally.
- Auxiliary 1-21 MHz RC oscillator (AUXHFRCO) for flash programming.

12.3 Functional Description

An overview of the CMU is shown in Figure 12.1 (p. 140) . The number of peripheral modules that are connected to the different clocks varies from device to device.

Figure 12.1. CMU Overview

12.3.1 System Clocks

12.3.1.1 HFCLK - High Frequency Clock

HFCLK is the selected High Frequency Clock. This clock is used by the CMU and drives the two prescalers that generate HFCORECLK and HFPERCLK. The HFCLK can be driven by a high-frequency oscillator (HFRCO, USHFRCO or HFXO) or one of the low-frequency oscillators (LFRCO or LFXO). By default the HFRCO is selected. In most applications, one of the high frequency oscillators will be the preferred choice. To change the selected HFCLK write to HFCLKSEL in CMU_CMD. The HFCLK is running in EM0 and EM1.

HFCLK can optionally be divided down by setting HFCLKDIV in CMU_CTRL to a nonzero value. This divides down HFCLK to all high frequency components, and combined with the HFCORECLK and HFPERCLK prescalers the HFCLK divider allows for more flexible clock division.

12.3.1.2 HFCORECLK - High Frequency Core Clock

HFCORECLK is a prescaled version of HFCLK. This clock drives the Core Modules, which consists of the CPU and modules that are tightly coupled to the CPU, e.g. MSC, DMA etc. This also includes the interface to the Low Energy Peripherals. Some of the modules that are driven by this clock can be clock gated completely when not in use. This is done by clearing the clock enable bit for the specific module in CMU_HFCORECLKEN0. The frequency of HFCORECLK is set using the CMU_HFCORECLKDIV register. The setting can be changed dynamically and the new setting takes effect immediately.

The USB Core runs on $HFCORECLK_{USBC}$. Selectable clock sources are LFXO, LFRCO and USHFRCO. When the USB Core is active this clock must be switched to a 32 kHz clock (LFRCO or LFXO) when entering EM2. The USB Core uses this clock for monitoring the USB bus. The switch is done by writing USBCCLKSEL in CMU_CMD. The currently active clock can be checked by reading CMU_STATUS. The clock switch can take up to 1.5 32 kHz cycle (45 us). To avoid polling the clock selection status when switching from 32 kHz to HFCLK when coming up from EM2 the USBCHFCLKSEL interrupt can be used. EM3 is not supported when the USB is active.

Note

Note that if HFPERCLK runs faster than HFCORECLK, the number of clock cycles for each bus-access to peripheral modules will increase with the ratio between the clocks. Please refer to Section 6.2.2.2 (p. 60) for more details.

12.3.1.3 HFPERCLK - High Frequency Peripheral Clock

Like HFCORECLK, HFPERCLK can also be a prescaled version of HFCLK. This clock drives the High-Frequency Peripherals. All the peripherals that are driven by this clock can be clock gated completely when not in use. This is done by clearing the clock enable bit for the specific peripheral in

CMU_HFPERCLKEN0. The frequency of HFPERCLK is set using the CMU_HFPERCLKDIV register. The setting can be changed dynamically and the new setting takes effect immediately.

Note

Note that if HFPERCLK runs faster than HFCORECLK, the number of clock cycles for each bus-access to peripheral modules will increase with the ratio between the clocks. E.g. if a bus-access normally takes three cycles, it will take 9 cycles if HFPERCLK runs three times as fast as the HFCORECLK.

12.3.1.4 LFACLK - Low Frequency A Clock

LFACLK is the selected clock for the Low Energy A Peripherals. There are four selectable sources for LFACLK: LFRCO, LFXO, HFCORECLK/2 and ULFRCO. In addition, the LFACLK can be disabled. From reset, the LFACLK source is set to LFRCO. However, note that the LFRCO is disabled from reset. The selection is configured using the LFA field in CMU_LFCLKSEL. The HFCORECLK/2 setting allows the Low Energy A Peripherals to be used as high-frequency peripherals.

Note

If HFCORECLK/2 is selected as LFACLK, the clock will stop in EM2/3.

Each Low Energy Peripheral that is clocked by LFACLK has its own prescaler setting and enable bit. The prescaler settings are configured using CMU_LFAPRESC0 and the clock enable bits can be found in CMU_LFACLKEN0. When operating in oversampling mode, the pulse counters are clocked by LFACLK. This is configured for each pulse counter (n) individually by setting PCNTnCLKSEL in CMU_PCNTCTRL.

12.3.1.5 LFBCLK - Low Frequency B Clock

LFBCLK is the selected clock for the Low Energy B Peripherals. There are four selectable sources for LFBCLK: LFRCO, LFXO, HFCORECLK/2 and ULFRCO. In addition, the LFBCLK can be disabled. From reset, the LFBCLK source is set to LFRCO. However, note that the LFRCO is disabled from reset. The selection is configured using the LFB field in CMU_LFCLKSEL. The HFCORECLK/2 setting allows the Low Energy B Peripherals to be used as high-frequency peripherals.

Note

If HFCORECLK/2 is selected as LFBCLK, the clock will stop in EM2/3.

Each Low Energy Peripheral that is clocked by LFBCLK has its own prescaler setting and enable bit. The prescaler settings are configured using CMU_LFBPRESC0 and the clock enable bits can be found in CMU_LFBCLKEN0.

12.3.1.6 LFCCLK - Low Frequency C Clock

LFCCLK is the selected clock for the Low Energy C Peripherals. There are two selectable sources for LFCCLK: LFRCO and LFXO. In addition, the LFCCLK can be disabled. From reset, the LFCCLK source is set to LFRCO. However, note that the LFRCO is disabled from reset. The selection is configured using the LFC field in CMU_LFCLKSEL.

12.3.1.7 PCNTnCLK - Pulse Counter n Clock

Each available pulse counter is driven by its own clock, PCNTnCLK where n is the pulse counter instance number. Each pulse counter can be configured to use an external pin (PCNTn_S0) or LFACLK as PCNTnCLK.

12.3.1.8 WDOGCLK - Watchdog Timer Clock

The Watchdog Timer (WDOG) can be configured to use one of three different clock sources: LFRCO, LFXO or ULFRCO. ULFRCO (Ultra Low Frequency RC Oscillator) is a separate 1 kHz RC oscillator that also runs in EM3.

12.3.1.9 AUXCLK - Auxiliary Clock

AUXCLK is a 1-21 MHz clock driven by a separate RC oscillator, AUXHFRCO. This clock is used for flash programming operation. During flash programming this clock will be active. If the AUXHFRCO has not been enabled explicitly by software, the MSC module will automatically start and stop it. The AUXHFRCO is enabled by writing a 1 to AUXHFRCOEN in CMU_OSCENCMD.

12.3.2 Oscillator Selection

12.3.2.1 Start-up Time

The different oscillators have different start-up times. For the RC oscillators, the start-up time is fixed, but both the LFXO and the HFXO have configurable start-up time. At the end of the start-up time a ready flag is set to indicated that the start-up time has exceeded and that the clock is available. The low start-up time values can be used for an external clock source of already high quality, while the higher start-up times should be used when the clock signal is coming directly from a crystal. The startup time for HFXO and LFXO can be set by configuring the HFXOTIMEOUT and LFXOTIMEOUT bitfields, respectively. Both bitfields are located in CMU_CTRL. For HFXO it is also possible to enable a glitch detection filter by setting HFXOGLITCHDETEN in CMU_CTRL. The glitch detector will reset the start-up counter if a glitch is detected, making the start-up process start over again.

There are individual bits for each oscillator indicating the status of the oscillator:

- ENABLED - Indicates that the oscillator is enabled
- READY - Start-up time is exceeded
- SELECTED - Start-up time is exceeded and oscillator is chosen as clock source

These status bits are located in the CMU_STATUS register.

12.3.2.2 Switching Clock Source

The HFRCO oscillator is a low energy oscillator with extremely short wake-up time. Therefore, this oscillator is always chosen by hardware as the clock source for HFCLK when the device starts up (e.g. after reset and after waking up from EM2 and EM3). After reset, the HFRCO frequency is 14 MHz.

Software can switch between the different clock sources at run-time. E.g., when the HFRCO is the clock source, software can switch to HFXO by writing the field HFCLKSEL in the CMU_CMD command register. See Figure 12.2 (p. 143) for a description of the sequence of events for this specific operation.

Note

It is important first to enable the HFXO since switching to a disabled oscillator will effectively stop HFCLK and only a reset can recover the system.

During the start-up period HFCLK will stop since the oscillator driving it is not ready. This effectively stalls the Core Modules and the High-Frequency Peripherals. It is possible to avoid this by first enabling the HFXO and then wait for the oscillator to become ready before switching the clock source. This way, the system continues to run on the HFRCO until the HFXO has timed out and provides a reliable clock. This sequence of events is shown in Figure 12.3 (p. 143) .

A separate flag is set when the oscillator is ready. This flag can also be configured to generate an interrupt.

Figure 12.2. CMU Switching from HFRCO to HFXO before HFXO is ready

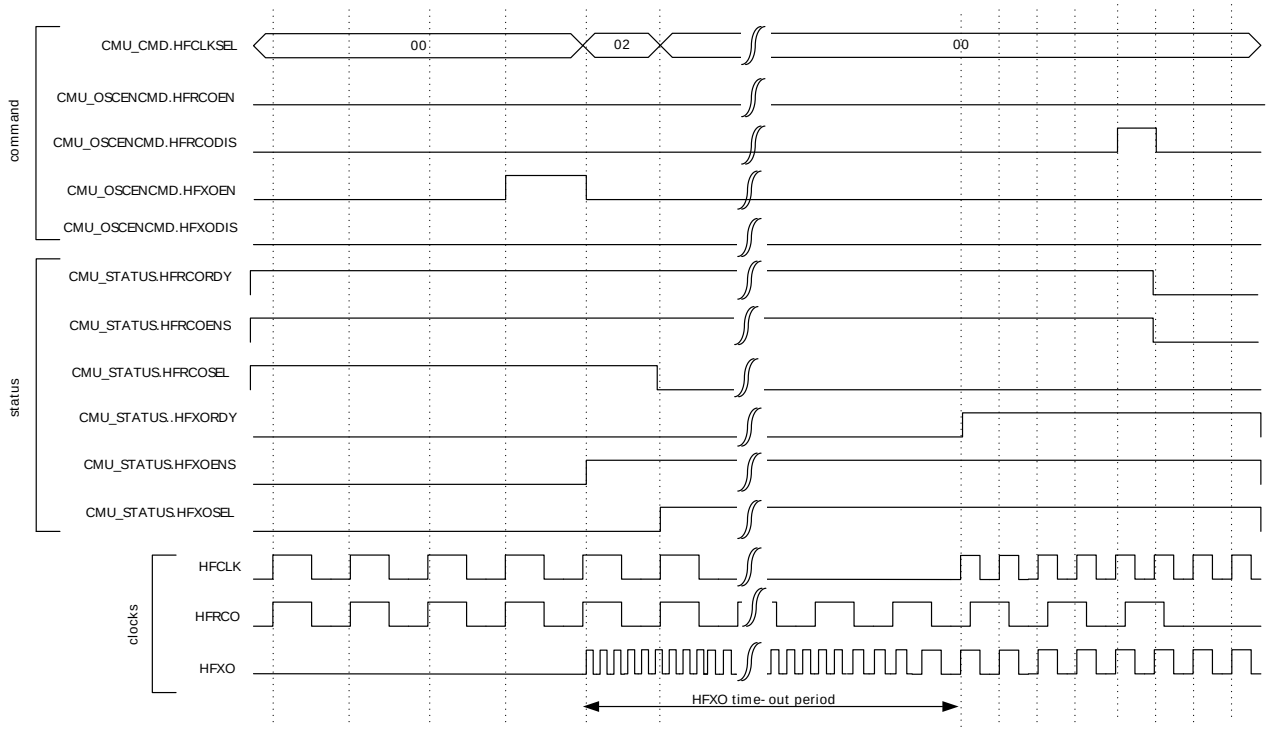
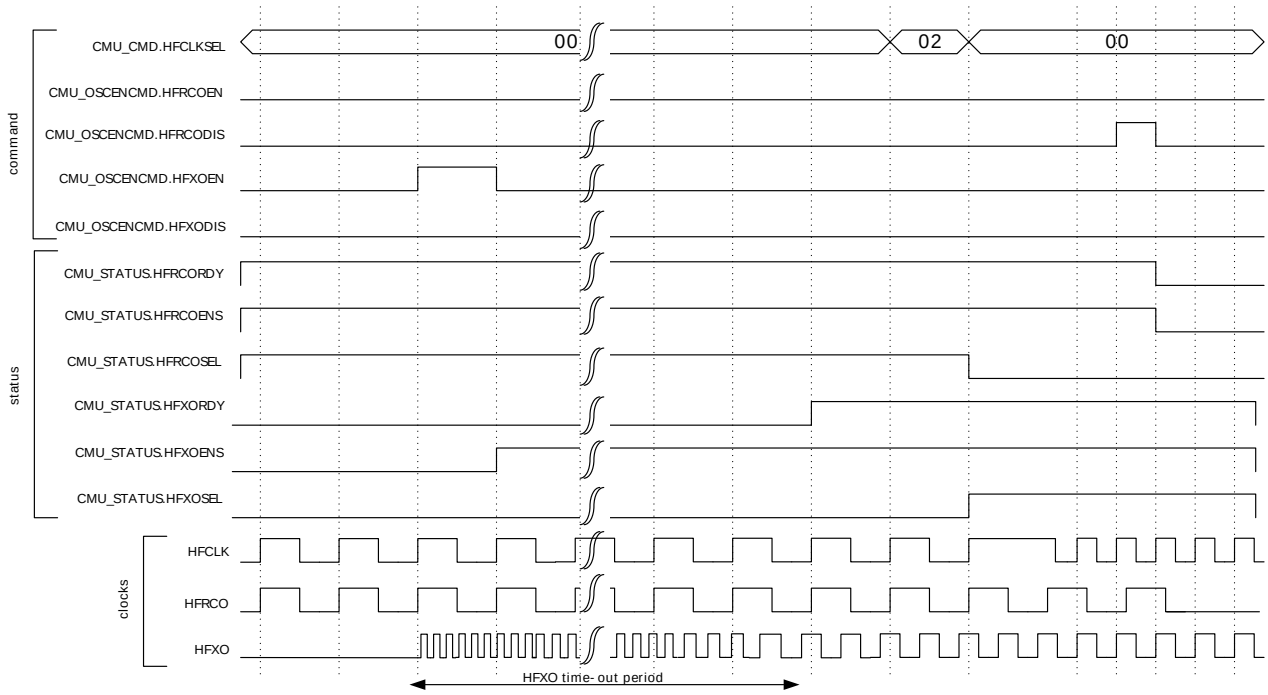


Figure 12.3. CMU Switching from HFRCO to HFXO after HFXO is ready



Switching clock source for LFACLK and LFBCLK is done by setting the LFA and LFB fields in CMU_LFCLKSEL. To ensure no stalls in the Low Energy Peripherals, the clock source should be ready before switching to it.

Note

To save energy, remember to turn off all oscillators not in use.

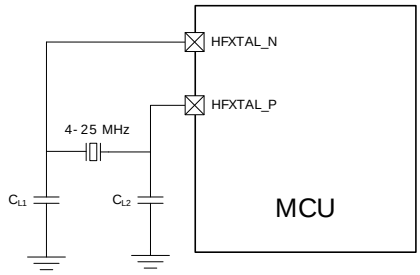
12.3.3 Oscillator Configuration

12.3.3.1 HFXO and LFXO

The crystal oscillators are by default configured to ensure safe startup and operation of the most common crystals. In order to optimize startup margin, startup time and power consumption for a given crystal, it is possible to adjust the gain in the oscillator. HFXO gain can be increased by setting HFXOBOOST field in CMU_CTRL, LFXO gain can be increased by setting LFXOBOOST field in CMU_CTRL. It is important that the boost settings, along with the crystal load capacitors are matched to the crystals in use. Correct values for these parameters can be found using the energyAware Designer.

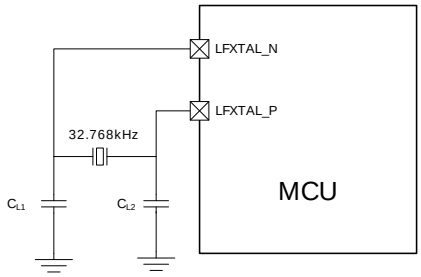
The HFXO crystal is connected to the HFXTAL_N/HFXTAL_P pins as shown in Figure 12.4 (p. 144)

Figure 12.4. HFXO Pin Connection



Similarly, the LFXO crystal is connected to the LFXTAL_N/LFXTAL_P pins as shown in Figure 12.5 (p. 144)

Figure 12.5. LFXO Pin Connection



It is possible to connect an external clock source to HFXTAL_N/LFXTAL_N pin of the HFXO or LFXO oscillator. By configuring the HFXOMODE/LFXOMODE fields in CMU_CTRL, the HFXO/LFXO can be bypassed.

12.3.3.2 USHFRCO

The USHFRCO has a startup time of 6 microseconds. This timeout needs to be configured in the TIMEOUT bit field of CMU_USHFRCOCTRL before starting the oscillator. The USHFCRO can be suspended by setting the SUSPEND bit in CMU_USHFRCOCTRL. From suspended state, the startup time is 200 nanoseconds. The USHFRCO has two frequency bands, 48MHz and 24MHz, configured in the BAND bit field in CMU_USHFRCOCONF. The frequency can be tuned by configuring the TUNING bit field in CMU_USHFRCOCTRL. For finer grained calibration, FINETUNING in CMU_USHFRCOTUNE can be used.

Note

If the USHFRCO is running at 48 MHz, the clock divider controlled by USHFRCODIV2DIS in CMU_USHFRCOCONF needs to be enabled before the USHFRCO is selected as HFCLK. When switching frequency band or enabling/disabling the USHFRCO clock divider, the USHFRCO should not be selected as clock source for HFCLK or USBC.

The USHFRCO can be automatically calibrated during USB communication to achieve sufficient accuracy. This feature is enabled by setting EN in CMU_USBCRCTRL. When operating USB in Low Speed mode, the LSMODE bit in CMU_USBCRCTRL also needs to be set. USB clock recovery will automatically tune the FINETUNING bit field in CMU_USHFRCOTUNE.

12.3.3.3 HFRCO, LFRCO and AUXHFRCO

The HFRCO and AUXHFRCO can be set to one of several different frequency bands from 1 MHz to 28 MHz by setting the BAND field in CMU_HFRCOCTRL and CMU_AUXHFRCOCTRL.The HFRCO and AUXHFRCO frequency bands are calibrated during production test, and the production tested calibration values can be read from the Device Information (DI) page. The DI page contains a separate tuning value for each frequency band. During reset, HFRCO and AUXHFRCO tuning values are set to the production calibrated values for the 14 MHz band, which is the default frequency band. When changing to a different HFRCO or AUXHFRCO band, make sure to also update the tuning value.

The LFRCO and is also calibrated in production and its TUNING value is set to the correct value during reset.

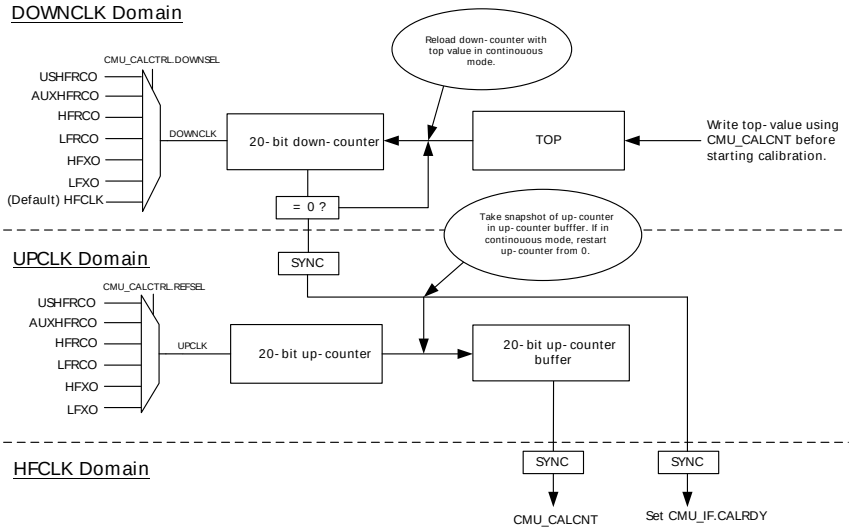
12.3.3.4 RC oscillator calibration

It is possible to calibrate the HFRCO, AUXHFRCO, USHFRCO and LFRCO to achieve higher accuracy (see the device datasheets for details on accuracy). The frequency is adjusted by changing the TUNING fields in CMU_HFRCOCTRL/CMU_AUXHFRCOCTRL/CMU_LFRCOCTRL. Changing to a higher value will result in a higher frequency. Please refer to the datasheet for stepsize details.

For the USHFRCO, the frequency can be tuned using the TUNING field in CMU_USHFRCOCTRL. The USHFRCO also employs a second set of FINETUNING registers in CMU_USHFRCOTUNE with smaller step-size allowing for much finer tuning. The FINETUNING registers are inteded for temperature/ voltage calibration, and is what the clock recovery hardware is using to keep the frequency constant over temperature. Note that for the USHFRCO both the TUNING and FINTUNING bit-fields are inverted, meaning that a higher value gives a lower frequency.

The CMU has built-in HW support to efficiently calibrate the RC oscillators at run-time, see Figure 12.6 (p. 146) The concept is to select a reference and compare the RC frequency with the reference frequency. When the calibration circuit is started, one down-counter running on a selectable clock (DOWNSEL in CMU_CALCTRL) and one up-counter running on a selectable clock (UPSEL in CMU_CALCTRL) are started simultaneously. The top value for the down-counter must be written to CMU_CALCNT before calibration is started. The smallest value that can be written to the CMU_CALCNT is 1. The down-counter counts for CMU_CALCNT+1 cycles. When the down-counter has reached 0, the up-counter is sampled and the CALRDY interrupt flag is set. If CONT in CMU_CALCTRL is cleared, the counters are stopped at this point. If continuous mode is selected by setting CONT in CMU_CALCTRL the down-counter reloads the top value and continues counting and the up-counter restarts from 0. Software can then read out the sampled up-counter value from CMU_CALCNT. Then it is easy to find the ratio between the reference and the oscillator subject to the calibration. Overflows of the up-counter will not occur. If the up-counter reaches its top value before the down counter reaches 0, the top counter stays at its top value. Calibration can be stopped by writing CALSTOP in CMU_CMD. With this HW support, it is simple to write efficient calibration algorithms in software.

Figure 12.6. HW-support for RC Oscillator Calibration



The counter operation for single and continuous mode are shown in Figure 12.7 (p. 146) and Figure 12.8 (p. 146) respectively.

Figure 12.7. Single Calibration (CONT=0)

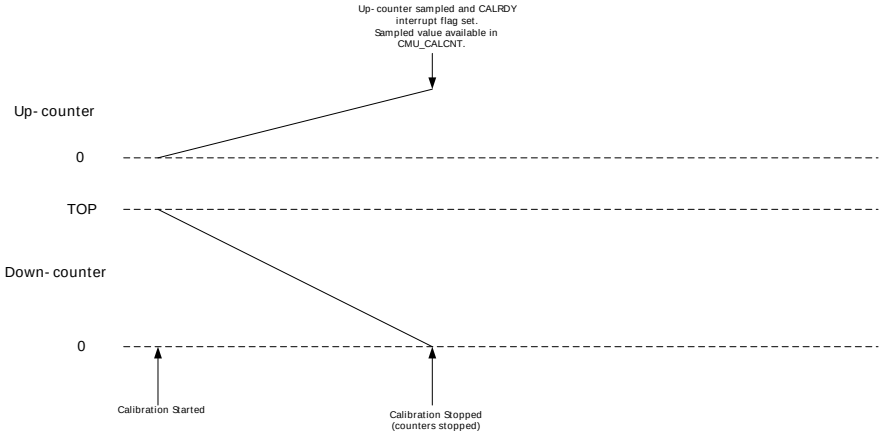
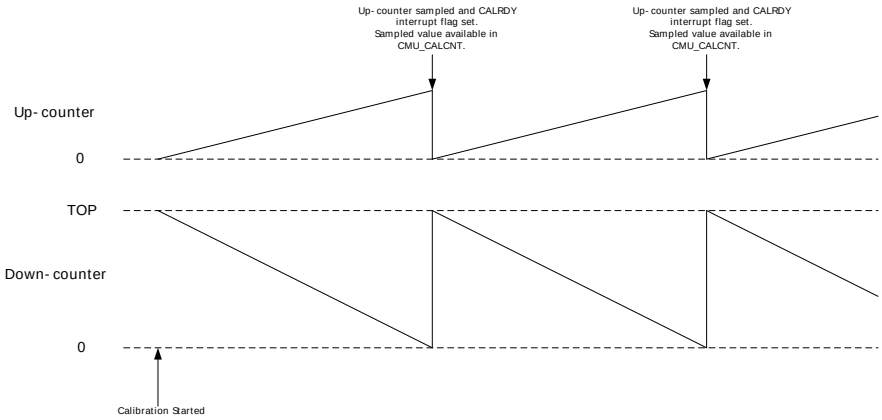


Figure 12.8. Continuous Calibration (CONT=1)



12.3.4 Output Clock on a Pin

It is possible to configure the CMU to output clocks on two pins. This clock selection is done using CLKOUTSEL0 and CLKOUTSEL1 fields in CMU_CTRL. The output pins must be configured in the CMU_ROUTE register.

- LFRCO, LFXO, HFCLK or the qualified clock from any of the oscillators can be output on one pin (CMU_OUT1). A qualified clock will not have any glitches or skewed duty-cycle during startup. For LFXO and HFXO you need to configure LFXOTIMEOUT and HFXOTIMEOUT in CMU_CTRL correctly to guarantee a qualified clock.
- HFRCO, HFXO, HFCLK/2, HFCLK/4, HFCLK/8, HFCLK/16, ULFRCO or AUXHFRCO can be output on another pin (CMU_OUT0)

Note that HFXO and HFRCO clock outputs to pin can be unstable after startup and should not be output on a pin before HFXORDY/HFRCORDY is set high in CMU_STATUS.

12.3.5 Protection

It is possible to lock the control- and command registers to prevent unintended software writes to critical clock settings. This is controlled by the CMU_LOCK register.

12.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	CMU_CTRL	RW	CMU Control Register
0x004	CMU_HFCORECLKDIV	RW	High Frequency Core Clock Division Register
0x008	CMU_HFPERCLKDIV	RW	High Frequency Peripheral Clock Division Register
0x00C	CMU_HFRCOCTRL	RW	HFRCO Control Register
0x010	CMU_LFRCOCTRL	RW	LFRCO Control Register
0x014	CMU_AUXHFRCOCTRL	RW	AUXHFRCO Control Register
0x018	CMU_CALCTRL	RW	Calibration Control Register
0x01C	CMU_CALCNT	RWH	Calibration Counter Register
0x020	CMU_OSCENCMD	W1	Oscillator Enable/Disable Command Register
0x024	CMU_CMD	W1	Command Register
0x028	CMU_LFCLKSEL	RW	Low Frequency Clock Select Register
0x02C	CMU_STATUS	R	Status Register
0x030	CMU_IF	R	Interrupt Flag Register
0x034	CMU_IFS	W1	Interrupt Flag Set Register
0x038	CMU_IFC	W1	Interrupt Flag Clear Register
0x03C	CMU_IEN	RW	Interrupt Enable Register
0x040	CMU_HFCORECLKEN0	RW	High Frequency Core Clock Enable Register 0
0x044	CMU_HFPERCLKEN0	RW	High Frequency Peripheral Clock Enable Register 0
0x050	CMU_SYNCBUSY	R	Synchronization Busy Register
0x054	CMU_FREEZE	RW	Freeze Register
0x058	CMU_LFACLKEN0	RW	Low Frequency A Clock Enable Register 0 (Async Reg)
0x060	CMU_LFBCLKEN0	RW	Low Frequency B Clock Enable Register 0 (Async Reg)
0x064	CMU_LFCCLKEN0	RW	Low Frequency C Clock Enable Register 0 (Async Reg)
0x068	CMU_LFAPRESC0	RW	Low Frequency A Prescaler Register 0 (Async Reg)
0x070	CMU_LFBPRESC0	RW	Low Frequency B Prescaler Register 0 (Async Reg)
0x078	CMU_PCNTCTRL	RW	PCNT Control Register
0x080	CMU_ROUTE	RW	I/O Routing Register
0x084	CMU_LOCK	RW	Configuration Lock Register
0x0D0	CMU_USBCRCTRL	RW	USB Clock Recovery Control
0x0D4	CMU_USHFRCOCTRL	RW	USHFRCO Control
0x0D8	CMU_USHFRCOTUNE	RWH	USHFRCO Frequency Tune
0x0DC	CMU_USHFRCOCONF	RW	USHFRCO Configuration

12.5 Register Description

12.5.1 CMU_CTRL - CMU Control Register

Offset	Bit Position																															
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset						0x0				0x0			0x3		0	0x0			1	0x0		0x3			0	0x1			0x3		0x0	
Access						RW				RW			RW		RW	RW			RW	RW		RW			RW	RW			RW		RW	
Name						CLKOUTSEL1				CLKOUTSEL0			LFXOTIMEOUT		LFXOBUFCUR	HFCLKDIV			LFXOBOOST	LFXOMODE		HFXOTIMEOUT			HFXOGLITCHDETEN	HFXOBUFCUR			HFXOBOOST		HFXOMODE	

Bit	Name	Reset	Access	Description
31:27	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
26:23	CLKOUTSEL1	0x0	RW	**Clock Output Select 1**

Controls the clock output multiplexer. To actually output on the pin, set CLKOUT1PEN in CMU_ROUTE.

Value	Mode	Description
0	LFRCO	LFRCO (directly from oscillator).
1	LFXO	LFXO (directly from oscillator).
2	HFCLK	HFCLK (undivided).
3	LFXOQ	LFXO (qualified).
4	HFXOQ	HFXO (qualified).
5	LFRCOQ	LFRCO (qualified).
6	HFRCOQ	HFRCO (qualified).
7	AUXHFRCOQ	AUXHFRCO (qualified).
8	USHFRCO	USHFRCO

22:20	CLKOUTSEL0	0x0	RW	**Clock Output Select 0**

Controls the clock output multiplexer. To actually output on the pin, set CLKOUT0PEN in CMU_ROUTE.

Value	Mode	Description
0	HFRCO	HFRCO (directly from oscillator).
1	HFXO	HFXO (directly from oscillator).
2	HFCLK2	HFCLK/2.
3	HFCLK4	HFCLK/4.
4	HFCLK8	HFCLK/8.
5	HFCLK16	HFCLK/16.
6	ULFRCO	ULFRCO (directly from oscillator).
7	AUXHFRCO	AUXHFRCO (directly from oscillator).

19:18	LFXOTIMEOUT	0x3	RW	**LFXO Timeout**

Configures the start-up delay for LFXO.

Value	Mode	Description
0	8CYCLES	Timeout period of 8 cycles.
1	1KCYCLES	Timeout period of 1024 cycles.
2	16KCYCLES	Timeout period of 16384 cycles.
3	32KCYCLES	Timeout period of 32768 cycles.

17	LFXOBUFCUR	0	RW	**LFXO Boost Buffer Current**

This value has been updated to the correct level during calibration and should not be changed.

16:14	HFCLKDIV	0x0	RW	**HFCLK Division**

Bit	Name	Reset	Access	Description
	Use to divide HFCLK frequency by (HFCLKDIV + 1).			
13	LFXOBOOST	1	RW	**LFXO Start-up Boost Current**
	Adjusts start-up boost current for LFXO.			

Value	Mode	Description
0	70PCENT	70 %.
1	100PCENT	100 %.

12:11	LFXOMODE	0x0	RW	**LFXO Mode**
	Set this to configure the external source for the LFXO. The oscillator setting takes effect when 1 is written to LFXOEN in CMU_OSCENCMD. The oscillator setting is reset to default when 1 is written to LFXODIS in CMU_OSCENCMD.			

Value	Mode	Description
0	XTAL	32.768 kHz crystal oscillator.
1	BUFEXTCLK	An AC coupled buffer is coupled in series with LFXTAL_N pin, suitable for external sinus wave (32.768 kHz).
2	DIGEXTCLK	Digital external clock on LFXTAL_N pin. Oscillator is effectively bypassed.

10:9	HFXOTIMEOUT	0x3	RW	**HFXO Timeout**
	Configures the start-up delay for HFXO.			

Value	Mode	Description
0	8CYCLES	Timeout period of 8 cycles.
1	256CYCLES	Timeout period of 256 cycles.
2	1KCYCLES	Timeout period of 1024 cycles.
3	16KCYCLES	Timeout period of 16384 cycles.

8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7	HFXOGLITCHDETEN	0	RW	**HFXO Glitch Detector Enable**
	This bit enables the glitch detector which is active as long as the start-up ripple-counter is counting. A detected glitch will reset the ripple-counter effectively increasing the start-up time. Once the ripple-counter has timed-out, glitches will not be detected.			
6:5	HFXOBUFCUR	0x1	RW	**HFXO Boost Buffer Current**
	This value has been set during calibration and should not be changed.			
4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3:2	HFXOBOOST	0x3	RW	**HFXO Start-up Boost Current**
	Used to adjust start-up boost current for HFXO.			

Value	Mode	Description
0	50PCENT	50 %.
1	70PCENT	70 %.
2	80PCENT	80 %.
3	100PCENT	100 % (default).

1:0	HFXOMODE	0x0	RW	**HFXO Mode**
	Set this to configure the external source for the HFXO. The oscillator setting takes effect when 1 is written to HFXOEN in CMU_OSCENCMD. The oscillator setting is reset to default when 1 is written to HFXODIS in CMU_OSCENCMD.			

Value	Mode	Description
0	XTAL	4-25 MHz crystal oscillator.
1	BUFEXTCLK	An AC coupled buffer is coupled in series with HFXTAL_N, suitable for external sine wave (4-25 MHz). The sine wave should have a minimum of 200 mV peak to peak.
2	DIGEXTCLK	Digital external clock on HFXTAL_N pin. Oscillator is effectively bypassed.

12.5.2 CMU_HFCORECLKDIV - High Frequency Core Clock Division Register

Offset	Bit Position																															
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																								0					0x0			
Access																								RW					RW			
Name																								HFCORECLKLEDIV					HFCORECLKDIV			

Bit	Name	Reset	Access	Description
31:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8	HFCORECLKLEDIV	0	RW	**Additional Division Factor For HFCORECLKLE**

Additional division factor for HFCORECLKLE.

Value	Mode	Description
0	DIV2	HFCORECLK divided by 2.
1	DIV4	HFCORECLK divided by 4.

7:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3:0	HFCORECLKDIV	0x0	RW	**HFCORECLK Divider**

Specifies the clock divider for HFCORECLK.

Value	Mode	Description
0	HFCLK	HFCORECLK = HFCLK.
1	HFCLK2	HFCORECLK = HFCLK/2.
2	HFCLK4	HFCORECLK = HFCLK/4.
3	HFCLK8	HFCORECLK = HFCLK/8.
4	HFCLK16	HFCORECLK = HFCLK/16.
5	HFCLK32	HFCORECLK = HFCLK/32.
6	HFCLK64	HFCORECLK = HFCLK/64.
7	HFCLK128	HFCORECLK = HFCLK/128.
8	HFCLK256	HFCORECLK = HFCLK/256.
9	HFCLK512	HFCORECLK = HFCLK/512.

12.5.3 CMU_HFPERCLKDIV - High Frequency Peripheral Clock Division Register

Offset	Bit Position																															
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																								1					0x0			
Access																								RW					RW			
Name																								HFPERCLKEN					HFPERCLKDIV			

Bit	Name	Reset	Access	Description
31:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
8	HFPERCLKEN	1	RW	**HFPERCLK Enable**
	Set to enable the HFPERCLK.			
7:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3:0	HFPERCLKDIV	0x0	RW	**HFPERCLK Divider**
	Specifies the clock divider for the HFPERCLK.			

Value	Mode	Description
0	HFCLK	HFPERCLK = HFCLK.
1	HFCLK2	HFPERCLK = HFCLK/2.
2	HFCLK4	HFPERCLK = HFCLK/4.
3	HFCLK8	HFPERCLK = HFCLK/8.
4	HFCLK16	HFPERCLK = HFCLK/16.
5	HFCLK32	HFPERCLK = HFCLK/32.
6	HFCLK64	HFPERCLK = HFCLK/64.
7	HFCLK128	HFPERCLK = HFCLK/128.
8	HFCLK256	HFPERCLK = HFCLK/256.
9	HFCLK512	HFPERCLK = HFCLK/512.

12.5.4 CMU_HFRCOCTRL - HFRCO Control Register

Offset	Bit Position																														
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x00						0x3		0x80							
Access																	RW						RW		RW							
Name																	SUDELAY						BAND		TUNING							

Bit	Name	Reset	Access	Description
31:17	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
16:12	SUDELAY	0x00	RW	**HFRCO Start-up Delay**
	Always write this field to 0.			
11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	BAND	0x3	RW	**HFRCO Band Select**
	Write this field to set the frequency band in which the HFRCO is to operate. When changing this setting there will be no glitches on the HFRCO output, hence it is safe to change this setting even while the system is running on the HFRCO. To ensure an accurate frequency, the HFTUNING value should also be written when changing the frequency band. The calibrated tuning value for the different bands can be read from the Device Information page.			

Value	Mode	Description
0	1MHZ	1 MHz band. NOTE: Also set the TUNING value (bits 7:0) when changing band.
1	7MHZ	7 MHz band. NOTE: Also set the TUNING value (bits 7:0) when changing band.
2	11MHZ	11 MHz band. NOTE: Also set the TUNING value (bits 7:0) when changing band.
3	14MHZ	14 MHz band. NOTE: Also set the TUNING value (bits 7:0) when changing band.
4	21MHZ	21 MHz band. NOTE: Also set the TUNING value (bits 7:0) when changing band.

7:0	TUNING	0x80	RW	**HFRCO Tuning Value**
	Writing this field adjusts the HFRCO frequency (the higher value, the higher frequency). This field is updated with the production calibrated value for the 14 MHz band during reset, and the reset value might therefore vary between devices.			

12.5.5 CMU_LFRCOCTRL - LFRCO Control Register

Offset	Bit Position																															
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																										0x40						
Access																										RW						
Name																										TUNING						

Bit	Name	Reset	Access	Description
31:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:0	TUNING	0x40	RW	**LFRCO Tuning Value**
	Writing this field adjusts the LFRCO frequency (the higher value, the higher frequency). This field is updated with the production calibrated value during reset, and the reset value might therefore vary between devices.			

12.5.6 CMU_AUXHFRCOCTRL - AUXHFRCO Control Register

Offset	Bit Position																															
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0x0						0x80				
Access																							RW						RW				
Name																							BAND						TUNING				

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	BAND	0x0	RW	**AUXHFRCO Band Select**
	Write this field to set the frequency band in which the AUXHFRCO is to operate. When changing this setting there will be no glitches on the AUXHFRCO output, hence it is safe to change this setting even while the system is using the AUXHFRCO. To ensure an accurate frequency, the AUXTUNING value should also be written when changing the frequency band. The calibrated tuning value for the different bands can be read from the Device Information page. Flash erase and write use this clock. If it is changed to another value than the default, MSC_TIMEBASE must also be configured to ensure correct flash erase and write operation.			

Value	Mode	Description
0	14MHZ	14 MHz band. NOTE: Also set the TUNING value (bits 7:0) when changing band.
1	11MHZ	11 MHz band. NOTE: Also set the TUNING value (bits 7:0) when changing band.
2	7MHZ	7 MHz band. NOTE: Also set the TUNING value (bits 7:0) when changing band.
3	1MHZ	1 MHz band. NOTE: Also set the TUNING value (bits 7:0) when changing band.
7	21MHZ	21 MHz band. NOTE: Also set the TUNING value (bits 7:0) when changing band.

7:0	TUNING	0x80	RW	**AUXHFRCO Tuning Value**
	Writing this field adjusts the AUXHFRCO frequency (the higher value, the higher frequency).This field is updated with the production calibrated value during reset, and the reset value might therefore vary between devices.			

12.5.7 CMU_CALCTRL - Calibration Control Register

Offset	Bit Position																														
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																										0	0x0			0x0		
Access																										RW	RW			RW		
Name																										CONT	DOWNSEL			UPSEL		

Bit	Name	Reset	Access	Description
31:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	CONT	0	RW	**Continuous Calibration**
	Set this bit to enable continuous calibration.			
5:3	DOWNSEL	0x0	RW	**Calibration Down-counter Select**
	Selects clock source for the calibration down-counter.			

Value	Mode	Description
0	HFCLK	Select HFCLK for down-counter.
1	HFXO	Select HFXO for down-counter.
2	LFXO	Select LFXO for down-counter.
3	HFRCO	Select HFRCO for down-counter.
4	LFRCO	Select LFRCO for down-counter.
5	AUXHFRCO	Select AUXHFRCO for down-counter.
6	USHFRCO	Select USHFRCO for down-counter.

Bit	Name	Reset	Access	Description
2:0	UPSEL	0x0	RW	**Calibration Up-counter Select**
	Selects clock source for the calibration up-counter.			

Value	Mode	Description
0	HFXO	Select HFXO as up-counter.
1	LFXO	Select LFXO as up-counter.
2	HFRCO	Select HFRCO as up-counter.
3	LFRCO	Select LFRCO as up-counter.
4	AUXHFRCO	Select AUXHFRCO as up-counter.
5	USHFRCO	Select USHFRCO as up-counter.

12.5.8 CMU_CALCNT - Calibration Counter Register

Offset	Bit Position																														
0x01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset													0x00000																			
Access													RWH																			
Name													CALCNT																			

Bit	Name	Reset	Access	Description
31:20	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
19:0	CALCNT	0x00000	RWH	**Calibration Counter**
	Write top value before calibration. Read calibration result from this register when Calibration Ready flag has been set.			

12.5.9 CMU_OSCENCMD - Oscillator Enable/Disable Command Register

Offset	Bit Position																													
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																					0	0	0	0	0	0	0	0	0	0	0	0
Access																					W1	W1	W1	W1	W1	W1	W1	W1	W1	W1	W1	W1
Name																					USHFRCODIS	USHFRCOEN	LFXODIS	LFXOEN	LFRCODIS	LFRCOEN	AUXHFRCODIS	AUXHFRCOEN	HFXODIS	HFXOEN	HFRCODIS	HFRCOEN

Bit	Name	Reset	Access	Description
31:12	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
11	USHFRCODIS	0	W1	**USHFRCO Disable**
	Disables the USHFRCO. USHFRCOEN has higher priority if written simultaneously.			
10	USHFRCOEN	0	W1	**USHFRCO Enable**
	Enables the USHFRCO.			
9	LFXODIS	0	W1	**LFXO Disable**
	Disables the LFXO. LFXOEN has higher priority if written simultaneously.			
8	LFXOEN	0	W1	**LFXO Enable**
	Enables the LFXO.			
7	LFRCODIS	0	W1	**LFRCO Disable**
	Disables the LFRCO. LFRCOEN has higher priority if written simultaneously.			
6	LFRCOEN	0	W1	**LFRCO Enable**
	Enables the LFRCO.			
5	AUXHFRCODIS	0	W1	**AUXHFRCO Disable**
	Disables the AUXHFRCO. AUXHFRCOEN has higher priority if written simultaneously. WARNING: Do not disable this clock during a flash erase/write operation.			
4	AUXHFRCOEN	0	W1	**AUXHFRCO Enable**
	Enables the AUXHFRCO.			
3	HFXODIS	0	W1	**HFXO Disable**
	Disables the HFXO. HFXOEN has higher priority if written simultaneously. WARNING: Do not disable the HFRXO if this oscillator is selected as the source for HFCLK.			
2	HFXOEN	0	W1	**HFXO Enable**
	Enables the HFXO.			
1	HFRCODIS	0	W1	**HFRCO Disable**
	Disables the HFRCO. HFRCOEN has higher priority if written simultaneously. WARNING: Do not disable the HFRCO if this oscillator is selected as the source for HFCLK.			
0	HFRCOEN	0	W1	**HFRCO Enable**
	Enables the HFRCO.			

12.5.10 CMU_CMD - Command Register

Offset	Bit Position																															
0x024	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																										0x0		0	0		0x0	
Access																										W1		W1	W1		W1	
Name																										USBCCLKSEL		CALSTOP	CALSTART		HFCLKSEL	

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:5	USBCCLKSEL	0x0	W1	**USB Core Clock Select**

Selects the clock for $HFCORECLK_{USBC}$. The status register is updated when the clock switch has taken effect.

Value	Mode	Description
2	LFXO	Select LFXO as $HFCORECLK_{USBC}$.
3	LFRCO	Select LFRCO as $HFCORECLK_{USBC}$.
4	USHFRCO	Select USHFRCO as $HFCORECLK_{USBC}$.

4	CALSTOP	0	W1	**Calibration Stop**

Stops the calibration counters.

3	CALSTART	0	W1	**Calibration Start**

Starts the calibration, effectively loading the CMU_CALCNT into the down-counter and start decrementing.

2:0	HFCLKSEL	0x0	W1	**HFCLK Select**

Selects the clock source for HFCLK. Note that selecting an oscillator that is disabled will cause the system clock to stop. Check the status register and confirm that oscillator is ready before switching.

Value	Mode	Description
1	HFRCO	Select HFRCO as HFCLK.
2	HFXO	Select HFXO as HFCLK.
3	LFRCO	Select LFRCO as HFCLK.
4	LFXO	Select LFXO as HFCLK.
5	USHFRCODIV2	Select USHFRCO divided by two as HFCLK.

12.5.11 CMU_LFCLKSEL - Low Frequency Clock Select Register

Offset	Bit Position																															
0x028	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset												0				0											0x1		0x1		0x1	
Access												RW				RW											RW		RW		RW	
Name												LFBE				LFAE											LFC		LFB		LFA	

Bit	Name	Reset	Access	Description
31:21	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
20	LFBE	0	RW	**Clock Select for LFB Extended**

This bit redefines the meaning of the LFB field.

Value	Mode	Description
0	DISABLED	LFBCLK is disabled (when LFB = DISABLED).

Bit	Name	Reset	Access	Description

Value	Mode	Description
1	ULFRCO	ULFRCO selected as LFBCLK (when LFB = DISABLED).

19:17	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
16	LFAE	0	RW	**Clock Select for LFA Extended**

This bit redefines the meaning of the LFA field.

Value	Mode	Description
0	DISABLED	LFACLK is disabled (when LFA = DISABLED).
1	ULFRCO	ULFRCO selected as LFACLK (when LFA = DISABLED).

15:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5:4	LFC	0x1	RW	**Clock Select for LFC**

Selects the clock source for LFCCLK.

Value	Mode	Description
0	DISABLED	LFCCLK clock disabled.
1	LFRCO	LFRCO selected as LFCCLK clock
2	LFXO	LFXO selected as LFCCLK clock

3:2	LFB	0x1	RW	**Clock Select for LFB**

Selects the clock source for LFBCLK.

LFB	LFBE	Mode	Description
0	0	Disabled	LFBCLK is disabled
1	0	LFRCO	LFRCO selected as LFBCLK
2	0	LFXO	LFXO selected as LFBCLK
3	0	HFCORECLKLEDIV2	$HFCORECLK_{LE}$ divided by two is selected as LFBCLK
0	1	ULFRCO	ULFRCO selected as LFBCLK

1:0	LFA	0x1	RW	**Clock Select for LFA**

Selects the clock source for LFACLK.

LFA	LFAE	Mode	Description
0	0	Disabled	LFACLK is disabled
1	0	LFRCO	LFRCO selected as LFACLK
2	0	LFXO	LFXO selected as LFACLK
3	0	HFCORECLKLEDIV2	$HFCORECLK_{LE}$ divided by two is selected as LFACLK
0	1	ULFRCO	ULFRCO selected as LFACLK

12.5.12 CMU_STATUS - Status Register

Offset	Bit Position																															
0x02C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset						0			0	0	0	0		0	0	0		0	0	0	0	1	0	0	0	0	0	0	0	0	1	1
Access						R			R	R	R	R		R	R	R		R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Name						USHFRCODIV2SEL			USHFRCOSUSPEND	USHFRCORDY	USHFRCOENS	USBCHFCLKSYNC		USBCUSHFRCOSEL	USBCLFRCOSEL	USBCLFXOSEL		CALBSY	LFXOSEL	LFRCOSEL	HFXOSEL	HFRCOSEL	LFXORDY	LFXOENS	LFRCORDY	LFRCOENS	AUXHFRCORDY	AUXHFRCOENS	HFXORDY	HFXOENS	HFRCORDY	HFRCOENS

Bit	Name	Reset	Access	Description
31:27	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
26	USHFRCODIV2SEL	0	R	**USHFRCODIV2 Selected**
	USHFRCO divided by two is selected a HFCLK clock source.			
25:24	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
23	USHFRCOSUSPEND	0	R	**USHFRCO is suspended**
	Set when the USHFRCO is suspended, either by CMU or USB.			
22	USHFRCORDY	0	R	**USHFRCO Ready**
	USHFRCO is enabled and start-up time has exceeded.			
21	USHFRCOENS	0	R	**USHFRCO Enable Status**
	USHFRCO is enabled.			
20	USBCHFCLKSYNC	0	R	**USBC is synchronous to HFCLK**
	Set when USBC is synchronous to HFCLK.			
19	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
18	USBCUSHFRCOSEL	0	R	**USBC USHFRCO Selected**
	USHFRCO is selected (and active) as $HFCORECLK_{USBC}$.			
17	USBCLFRCOSEL	0	R	**USBC LFRCO Selected**
	LFRCO is selected (and active) as $HFCORECLK_{USBC}$.			
16	USBCLFXOSEL	0	R	**USBC LFXO Selected**
	LFXO is selected (and active) as $HFCORECLK_{USBC}$.			
15	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
14	CALBSY	0	R	**Calibration Busy**
	Calibration is on-going.			
13	LFXOSEL	0	R	**LFXO Selected**
	LFXO is selected as HFCLK clock source.			
12	LFRCOSEL	0	R	**LFRCO Selected**
	LFRCO is selected as HFCLK clock source.			
11	HFXOSEL	0	R	**HFXO Selected**
	HFXO is selected as HFCLK clock source.			
10	HFRCOSEL	1	R	**HFRCO Selected**
	HFRCO is selected as HFCLK clock source.			
9	LFXORDY	0	R	**LFXO Ready**
	LFXO is enabled and start-up time has exceeded.			
8	LFXOENS	0	R	**LFXO Enable Status**
	LFXO is enabled.			
7	LFRCORDY	0	R	**LFRCO Ready**
	LFRCO is enabled and start-up time has exceeded.			
6	LFRCOENS	0	R	**LFRCO Enable Status**
	LFRCO is enabled.			
5	AUXHFRCORDY	0	R	**AUXHFRCO Ready**
	AUXHFRCO is enabled and start-up time has exceeded.			
4	AUXHFRCOENS	0	R	**AUXHFRCO Enable Status**
	AUXHFRCO is enabled.			
3	HFXORDY	0	R	**HFXO Ready**
	HFXO is enabled and start-up time has exceeded.			
2	HFXOENS	0	R	**HFXO Enable Status**
	HFXO is enabled.			

Bit	Name	Reset	Access	Description
1	HFRCORDY	1	R	**HFRCO Ready**
	HFRCO is enabled and start-up time has exceeded.			
0	HFRCOENS	1	R	**HFRCO Enable Status**
	HFRCO is enabled.			

12.5.13 CMU_IF - Interrupt Flag Register

Offset	Bit Position																													
0x030	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0	0		0	0	0	0	0	0	1
Access																							R	R		R	R	R	R	R	R	R
Name																							USBCHFOSCSEL	USHFRCORDY		CALOF	CALRDY	AUXHFRCORDY	LFXORDY	LFRCORDY	HFXORDY	HFRCORDY

Bit	Name	Reset	Access	Description
31:10	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
9	USBCHFOSCSEL	0	R	**USBC HF-oscillator Selected Interrupt Flag**
	Set when USBC is coming from a High Frequency Oscillator.			
8	USHFRCORDY	0	R	**USHFRCO Ready Interrupt Flag**
	Set when USHFRCO is ready (start-up time exceeded).			
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	CALOF	0	R	**Calibration Overflow Interrupt Flag**
	Set when calibration overflow has occurred			
5	CALRDY	0	R	**Calibration Ready Interrupt Flag**
	Set when calibration is completed.			
4	AUXHFRCORDY	0	R	**AUXHFRCO Ready Interrupt Flag**
	Set when AUXHFRCO is ready (start-up time exceeded).			
3	LFXORDY	0	R	**LFXO Ready Interrupt Flag**
	Set when LFXO is ready (start-up time exceeded).			
2	LFRCORDY	0	R	**LFRCO Ready Interrupt Flag**
	Set when LFRCO is ready (start-up time exceeded).			
1	HFXORDY	0	R	**HFXO Ready Interrupt Flag**
	Set when HFXO is ready (start-up time exceeded).			
0	HFRCORDY	1	R	**HFRCO Ready Interrupt Flag**
	Set when HFRCO is ready (start-up time exceeded).			

12.5.14 CMU_IFS - Interrupt Flag Set Register

Offset	Bit Position																																	
0x034	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0	0		0	0	0	0	0	0	0
Access																							W1	W1		W1	W1	W1	W1	W1	W1	W1
Name																							USBCHFOSCSEL	USHFRCORDY		CALOF	CALRDY	AUXHFRCORDY	LFXORDY	LFRCORDY	HFXORDY	HFRCORDY

Bit	Name	Reset	Access	Description
31:10	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
9	USBCHFOSCSEL	0	W1	**USBC HF-oscillator Selected Interrupt Flag Set**
	Write to 1 to set the USBC HF-oscillator Selected Interrupt Flag.			
8	USHFRCORDY	0	W1	**USHFRCO Ready Interrupt Flag Set**
	Write to 1 to set the USHFRCO Ready Interrupt Flag.			
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	CALOF	0	W1	**Calibration Overflow Interrupt Flag Set**
	Write to 1 to set the Calibration Overflow Interrupt Flag.			
5	CALRDY	0	W1	**Calibration Ready Interrupt Flag Set**
	Write to 1 to set the Calibration Ready(completed) Interrupt Flag.			
4	AUXHFRCORDY	0	W1	**AUXHFRCO Ready Interrupt Flag Set**
	Write to 1 to set the AUXHFRCO Ready Interrupt Flag.			
3	LFXORDY	0	W1	**LFXO Ready Interrupt Flag Set**
	Write to 1 to set the LFXO Ready Interrupt Flag.			
2	LFRCORDY	0	W1	**LFRCO Ready Interrupt Flag Set**
	Write to 1 to set the LFRCO Ready Interrupt Flag.			
1	HFXORDY	0	W1	**HFXO Ready Interrupt Flag Set**
	Write to 1 to set the HFXO Ready Interrupt Flag.			
0	HFRCORDY	0	W1	**HFRCO Ready Interrupt Flag Set**
	Write to 1 to set the HFRCO Ready Interrupt Flag.			

12.5.15 CMU_IFC - Interrupt Flag Clear Register

Offset	Bit Position																																	
0x038	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0	0		0	0	0	0	0	0	0
Access																							W1	W1		W1	W1	W1	W1	W1	W1	W1
Name																							USBCHFOSCSEL	USHFRCORDY		CALOF	CALRDY	AUXHFRCORDY	LFXORDY	LFRCORDY	HFXORDY	HFRCORDY

Bit	Name	Reset	Access	Description
31:10	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
9	USBCHFOSCSEL	0	W1	**USBC HF-oscillator Selected Interrupt Flag Clear**
	Write to 1 to clear the USBC HF-oscillator Selected Interrupt Flag.			
8	USHFRCORDY	0	W1	**USHFRCO Ready Interrupt Flag Clear**
	Write to 1 to clear the USHFRCO Ready Interrupt Flag.			
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	CALOF	0	W1	**Calibration Overflow Interrupt Flag Clear**
	Write to 1 to clear the Calibration Overflow Interrupt Flag.			
5	CALRDY	0	W1	**Calibration Ready Interrupt Flag Clear**
	Write to 1 to clear the Calibration Ready Interrupt Flag.			
4	AUXHFRCORDY	0	W1	**AUXHFRCO Ready Interrupt Flag Clear**
	Write to 1 to clear the AUXHFRCO Ready Interrupt Flag.			
3	LFXORDY	0	W1	**LFXO Ready Interrupt Flag Clear**
	Write to 1 to clear the LFXO Ready Interrupt Flag.			
2	LFRCORDY	0	W1	**LFRCO Ready Interrupt Flag Clear**
	Write to 1 to clear the LFRCO Ready Interrupt Flag.			
1	HFXORDY	0	W1	**HFXO Ready Interrupt Flag Clear**
	Write to 1 to clear the HFXO Ready Interrupt Flag.			
0	HFRCORDY	0	W1	**HFRCO Ready Interrupt Flag Clear**
	Write to 1 to clear the HFRCO Ready Interrupt Flag.			

12.5.16 CMU_IEN - Interrupt Enable Register

Offset	Bit Position																													
0x03C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0	0		0	0	0	0	0	0	0
Access																							RW	RW		RW	RW	RW	RW	RW	RW	RW
Name																							USBCHFOSCSEL	USHFRCORDY		CALOF	CALRDY	AUXHFRCORDY	LFXORDY	LFRCORDY	HFXORDY	HFRCORDY

Bit	Name	Reset	Access	Description
31:10	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
9	USBCHFOSCSEL	0	RW	**USBC HF-oscillator Selected Interrupt Flag Clear**
	Set to enable the USBC HF-oscillator Selected Interrupt Flag.			
8	USHFRCORDY	0	RW	**USHFRCO Ready Interrupt Enable**
	Set to enable the USHFRCO Ready Interrupt.			
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	CALOF	0	RW	**Calibration Overflow Interrupt Enable**
	Set to enable the Calibration Overflow Interrupt.			
5	CALRDY	0	RW	**Calibration Ready Interrupt Enable**
	Set to enable the Calibration Ready Interrupt.			
4	AUXHFRCORDY	0	RW	**AUXHFRCO Ready Interrupt Enable**
	Set to enable the AUXHFRCO Ready Interrupt.			

Bit	Name	Reset	Access	Description
3	LFXORDY	0	RW	**LFXO Ready Interrupt Enable**
	Set to enable the LFXO Ready Interrupt.			
2	LFRCORDY	0	RW	**LFRCO Ready Interrupt Enable**
	Set to enable the LFRCO Ready Interrupt.			
1	HFXORDY	0	RW	**HFXO Ready Interrupt Enable**
	Set to enable the HFXO Ready Interrupt.			
0	HFRCORDY	0	RW	**HFRCO Ready Interrupt Enable**
	Set to enable the HFRCO Ready Interrupt.			

12.5.17 CMU_HFCORECLKEN0 - High Frequency Core Clock Enable Register 0

Offset	Bit Position																													
0x040	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																												0	0	0	0	0
Access																												RW	RW	RW	RW	RW
Name																												USB	USBC	LE	DMA	AES

Bit	Name	Reset	Access	Description
31:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	USB	0	RW	**Universal Serial Bus Interface Clock Enable**
	Set to enable the clock for USB.			
3	USBC	0	RW	**Universal Serial Bus Interface Core Clock Enable**
	Set to enable the clock for USBC.			
2	LE	0	RW	**Low Energy Peripheral Interface Clock Enable**
	Set to enable the clock for LE. Interface used for bus access to Low Energy peripherals.			
1	DMA	0	RW	**Direct Memory Access Controller Clock Enable**
	Set to enable the clock for DMA.			
0	AES	0	RW	**Advanced Encryption Standard Accelerator Clock Enable**
	Set to enable the clock for AES.			

12.5.18 CMU_HFPERCLKEN0 - High Frequency Peripheral Clock Enable Register 0

Offset	Bit Position																													
0x044	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																					0	0	0	0	0	0		0	0	0	0	0
Access																					RW	RW	RW	RW	RW	RW		RW	RW	RW	RW	RW
Name																					I2C0	ADC0	VCMP	GPIO	IDAC0	PRS		USARTRF1	USART0	TIMER2	TIMER1	TIMER0

Bit	Name	Reset	Access	Description
31:12	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
11	I2C0	0	RW	**I2C 0 Clock Enable**
	Set to enable the clock for I2C0.			
10	ADC0	0	RW	**Analog to Digital Converter 0 Clock Enable**
	Set to enable the clock for ADC0.			
9	VCMP	0	RW	**Voltage Comparator Clock Enable**
	Set to enable the clock for VCMP.			
8	GPIO	0	RW	**General purpose Input/Output Clock Enable**
	Set to enable the clock for GPIO.			
7	IDAC0	0	RW	**Current Digital to Analog Converter 0 Clock Enable**
	Set to enable the clock for IDAC0.			
6	PRS	0	RW	**Peripheral Reflex System Clock Enable**
	Set to enable the clock for PRS.			
5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	USARTRF1	0	RW	**Universal Synchronous/Asynchronous Receiver/Transmitter 1 Clock Enable**
	Set to enable the clock for USARTRF1.			
3	USART0	0	RW	**Universal Synchronous/Asynchronous Receiver/Transmitter 0 Clock Enable**
	Set to enable the clock for USART0.			
2	TIMER2	0	RW	**Timer 2 Clock Enable**
	Set to enable the clock for TIMER2.			
1	TIMER1	0	RW	**Timer 1 Clock Enable**
	Set to enable the clock for TIMER1.			
0	TIMER0	0	RW	**Timer 0 Clock Enable**
	Set to enable the clock for TIMER0.			

12.5.19 CMU_SYNCBUSY - Synchronization Busy Register

Offset	Bit Position																													
0x050	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																								0		0		0		0		0
Access																								R		R		R		R		R
Name																								LFCCLKEN0		LFBPRESC0		LFBCLKEN0		LFAPRESC0		LFACLKEN0

Bit	Name	Reset	Access	Description
31:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8	LFCCLKEN0	0	R	**Low Frequency C Clock Enable 0 Busy**
	Used to check the synchronization status of CMU_LFCCLKEN0.			

Value	Description
0	CMU_LFCCLKEN0 is ready for update.
1	CMU_LFCCLKEN0 is busy synchronizing new value.

7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
6	LFBPRESC0	0	R	**Low Frequency B Prescaler 0 Busy**

Used to check the synchronization status of CMU_LFBPRESC0.

Value	Description
1	CMU_LFBPRESC0 is busy synchronizing new value.

5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	LFBCLKEN0	0	R	**Low Frequency B Clock Enable 0 Busy**

Used to check the synchronization status of CMU_LFBCLKEN0.

Value	Description
0	CMU_LFBCLKEN0 is ready for update.
1	CMU_LFBCLKEN0 is busy synchronizing new value.

3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	LFAPRESC0	0	R	**Low Frequency A Prescaler 0 Busy**

Used to check the synchronization status of CMU_LFAPRESC0.

Value	Description
0	CMU_LFAPRESC0 is ready for update.
1	CMU_LFAPRESC0 is busy synchronizing new value.

1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	LFACLKEN0	0	R	**Low Frequency A Clock Enable 0 Busy**

Used to check the synchronization status of CMU_LFACLKEN0.

Value	Description
0	CMU_LFACLKEN0 is ready for update.
1	CMU_LFACLKEN0 is busy synchronizing new value.

12.5.20 CMU_FREEZE - Freeze Register

Offset	Bit Position																															
0x054	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																RW
Name																																REGFREEZE

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	REGFREEZE	0	RW	**Register Update Freeze**

When set, the update of the Low Frequency clock control registers is postponed until this bit is cleared. Use this bit to update several registers simultaneously.

Value	Mode	Description
0	UPDATE	Each write access to a Low Frequency clock control register is updated into the Low Frequency domain as soon as possible.
1	FREEZE	The LE Clock Control registers are not updated with the new written value.

12.5.21 CMU_LFACLKEN0 - Low Frequency A Clock Enable Register 0 (Async Reg)

Offset	Bit Position																															
0x058	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																RW
Name																																RTC

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	RTC	0	RW	**Real-Time Counter Clock Enable**
	Set to enable the clock for RTC.			

12.5.22 CMU_LFBCLKEN0 - Low Frequency B Clock Enable Register 0 (Async Reg)

Offset	Bit Position																															
0x060	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																RW
Name																																LEUART0

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	LEUART0	0	RW	**Low Energy UART 0 Clock Enable**
	Set to enable the clock for LEUART0.			

12.5.23 CMU_LFCCLKEN0 - Low Frequency C Clock Enable Register 0 (Async Reg)

Offset	Bit Position																															
0x064	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																RW
Name																																USBLE

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	USBLE	0	RW	**Universal Serial Bus Low Energy Clock Clock Enable**

Bit	Name	Reset	Access	Description
	Set to enable the clock for USBLE.			

12.5.24 CMU_LFAPRESC0 - Low Frequency A Prescaler Register 0 (Async Reg)

Offset	Bit Position																														
0x068	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																														0x0		
Access																														RW		
Name																														RTC		

Bit	Name	Reset	Access	Description
31:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3:0	RTC	0x0	RW	**Real-Time Counter Prescaler**

Configure Real-Time Counter prescaler

Value	Mode	Description
0	DIV1	$LFACLK_{RTC}$ = LFACLK
1	DIV2	$LFACLK_{RTC}$ = LFACLK/2
2	DIV4	$LFACLK_{RTC}$ = LFACLK/4
3	DIV8	$LFACLK_{RTC}$ = LFACLK/8
4	DIV16	$LFACLK_{RTC}$ = LFACLK/16
5	DIV32	$LFACLK_{RTC}$ = LFACLK/32
6	DIV64	$LFACLK_{RTC}$ = LFACLK/64
7	DIV128	$LFACLK_{RTC}$ = LFACLK/128
8	DIV256	$LFACLK_{RTC}$ = LFACLK/256
9	DIV512	$LFACLK_{RTC}$ = LFACLK/512
10	DIV1024	$LFACLK_{RTC}$ = LFACLK/1024
11	DIV2048	$LFACLK_{RTC}$ = LFACLK/2048
12	DIV4096	$LFACLK_{RTC}$ = LFACLK/4096
13	DIV8192	$LFACLK_{RTC}$ = LFACLK/8192
14	DIV16384	$LFACLK_{RTC}$ = LFACLK/16384
15	DIV32768	$LFACLK_{RTC}$ = LFACLK/32768

12.5.25 CMU_LFBPRESC0 - Low Frequency B Prescaler Register 0 (Async Reg)

Offset	Bit Position																														
0x070	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0x0	
Access																															RW	
Name																															LEUART0	

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
1:0	LEUART0	0x0	RW	**Low Energy UART 0 Prescaler**

Configure Low Energy UART 0 prescaler

Value	Mode	Description
0	DIV1	$LFBCLK_{LEUART0}$ = LFBCLK
1	DIV2	$LFBCLK_{LEUART0}$ = LFBCLK/2
2	DIV4	$LFBCLK_{LEUART0}$ = LFBCLK/4
3	DIV8	$LFBCLK_{LEUART0}$ = LFBCLK/8

12.5.26 CMU_PCNTCTRL - PCNT Control Register

Offset	Bit Position																														
0x078	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															RW	RW
Name																															PCNT0CLKSEL	PCNT0CLKEN

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	PCNT0CLKSEL	0	RW	**PCNT0 Clock Select**

This bit controls which clock that is used for the PCNT.

Value	Mode	Description
0	LFACLK	LFACLK is clocking PCNT0.
1	PCNT0S0	External pin PCNT0_S0 is clocking PCNT0.

Bit	Name	Reset	Access	Description
0	PCNT0CLKEN	0	RW	**PCNT0 Clock Enable**

This bit enables/disables the clock to the PCNT.

Value	Description
0	PCNT0 is disabled.
1	PCNT0 is enabled.

12.5.27 CMU_ROUTE - I/O Routing Register

Offset	Bit Position																														
0x080	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																													0x0		0	0
Access																													RW		RW	RW
Name																													LOCATION		CLKOUT1PEN	CLKOUT0PEN

Bit	Name	Reset	Access	Description
31:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4:2	LOCATION	0x0	RW	**I/O Location**

Bit	Name	Reset	Access	Description
	Decides the location of the CMU I/O pins.			

Value	Mode	Description
0	LOC0	Location 0
1	LOC1	Location 1
2	LOC2	Location 2
3	LOC3	Location 3

1	CLKOUT1PEN	0	RW	**CLKOUT1 Pin Enable**
	When set, the CLKOUT1 pin is enabled.			
0	CLKOUT0PEN	0	RW	**CLKOUT0 Pin Enable**
	When set, the CLKOUT0 pin is enabled.			

12.5.28 CMU_LOCK - Configuration Lock Register

Offset	Bit Position																													
0x084	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	LOCKKEY															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	LOCKKEY	0x0000	RW	**Configuration Lock Key**
	Write any other value than the unlock code to lock CMU_CTRL, CMU_HFCORECLKDIV, CMU_HFPERCLKDIV, CMU_HFRCOCTRL, CMU_LFRCOCTRL, CMU_AUXHFRCOCTRL, CMU_OSCENCMD, CMU_CMD, CMU_LFCLKSEL, CMU_HFCORECLKEN0, CMU_HFPERCLKEN0, CMU_LFACLKEN0, CMU_LFBCLKEN0, CMU_LFAPRESC0, CMU_LFBPRESC0, CMU_USHFRCOCTRL, CMU_USHFRCOFTUNE and CMU_PCNTCTRL from editing. Write the unlock code to unlock. When reading the register, bit 0 is set when the lock is enabled.			

Mode	Value	Description
Read Operation		
UNLOCKED	0	CMU registers are unlocked.
LOCKED	1	CMU registers are locked.
Write Operation		
LOCK	0	Lock CMU registers.
UNLOCK	0x580E	Unlock CMU registers.

12.5.29 CMU_USBCRCTRL - USB Clock Recovery Control

Offset	Bit Position																													
0x0D0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															RW	RW
Name																															LSMODE	EN

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	LSMODE	0	RW	**Low Speed Clock Recovery Mode**
	This bit must be set to 1 if clock recovery is used when operating as a Low Speed USB device.			
0	EN	0	RW	**Clock Recovery Enable**
	This bit enables and disables the USB clock recovery feature.			

12.5.30 CMU_USHFRCOCTRL - USHFRCO Control

Offset	Bit Position																													
0x0D4	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset													0xFF										0	0		0x40						
Access													RW										RW	RW		RW						
Name													TIMEOUT										SUSPEND	DITHEN		TUNING						

Bit	Name	Reset	Access	Description
31:20	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
19:12	TIMEOUT	0xFF	RW	**USHFRCO Timeout**
	Timeout value in HFCLK cycles for USHFRCO startup. The timeout needs to be at least 6 µs.			
11:10	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
9	SUSPEND	0	RW	**USHFRCO suspend**
	Set this bit to suspend the USHFRCO.			
8	DITHEN	0	RW	**USHFRCO dither enable**
	Setting this bit dithers the oscillator control value every four oscillator cycles. In effect this gives an average USHFRCO frequency between FINETUNING and FINETUNING+1.			
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:0	TUNING	0x40	RW	**USHFRCO frequency adjust**
	This field controls the output frequency of the USHFRCO in coarse steps. The reset value is factory calibrated to generate a USB oscillator frequency of 48 MHz. Notice that higher value gives lower frequency.			

12.5.31 CMU_USHFRCOTUNE - USHFRCO Frequency Tune

Offset	Bit Position																													
0x0D8	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																													0x20			
Access																													RWH			
Name																													FINETUNING			

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
5:0	FINETUNING	0x20	RWH	**Oscillator fine frequency adjust**
	This field controls the output frequency of the USHFRCO in fine steps. The reset value is factory calibrated to generate a USHFRCO frequency of 48 MHz. This register is modified by the clock recovery hardware to fine-tune the USHFRCO to meet the requirements for USB clock tolerance. Notice that higher value gives lower frequency			

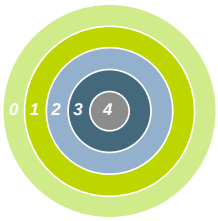
12.5.32 CMU_USHFRCOCONF - USHFRCO Configuration

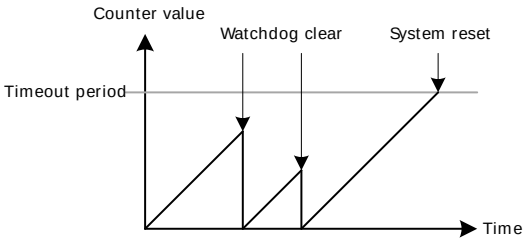
Offset	Bit Position																																
0x0DC	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reset																												0			0x1		
Access																												RW			RW		
Name																												USHFRCODIV2DIS			BAND		

Bit	Name	Reset	Access	Description
31:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	USHFRCODIV2DIS	0	RW	**USHFRCO divider for HFCLK disable**
	Set this bit to bypass the divider for USHFRCO to HFCLK. Must not be changed while USHFRCO is selected as HFCLK or the 48 MHz band is selected.			
3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2:0	BAND	0x1	RW	**USHFRCO Band Select**
	Write this field to set the frequency band in which the USHFRCO is to operate. Note that the switching of band on this oscillator is not glitch-free and it should be selected as neither USBC clock nor HFCLK when changing band.			

Value	Mode	Description
1	48MHZ	48 MHz band. NOTE: Also set the TUNING and FINETUNING value when changing band.
3	24MHZ	24 MHz band. NOTE: Also set the TUNING and FINETUNING value when changing band.

13 WDOG - Watchdog Timer





Quick Facts

What?

The WDOG (Watchdog Timer) resets the system in case of a fault condition, and can be enabled in all energy modes as long as the low frequency clock source is available.

Why?

If a software failure or external event renders the MCU unresponsive, a Watchdog timeout will reset the system to a known, safe state.

How?

An enabled Watchdog Timer implements a configurable timeout period. If the CPU fails to re-start the Watchdog Timer before it times out, a full system reset will be triggered. The Watchdog consumes insignificant power, and allows the device to remain safely in low energy modes for up to 256 seconds at a time.

13.1 Introduction

The purpose of the watchdog timer is to generate a reset in case of a system failure, to increase application reliability. The failure may e.g. be caused by an external event, such as an ESD pulse, or by a software failure.

13.2 Features

- Clock input from selectable oscillators
  - Internal 32.768 Hz RC oscillator
  - Internal 1 kHz RC oscillator
  - External 32.768 Hz XTAL oscillator
- Configurable timeout period from 9 to 256k watchdog clock cycles
- Individual selection to keep running or freeze when entering EM2 or EM3
- Selection to keep running or freeze when entering debug mode
- Selection to block the CPU from entering Energy Mode 4
- Selection to block the CMU from disabling the selected watchdog clock

13.3 Functional Description

The watchdog is enabled by setting the EN bit in WDOG_CTRL. When enabled, the watchdog counts up to the period value configured through the PERSEL field in WDOG_CTRL. If the watchdog timer is not cleared to 0 (by writing a 1 to the CLEAR bit in WDOG_CMD) before the period is reached, the chip is reset. If a timely clear command is issued, the timer starts counting up from 0 again. The watchdog can optionally be locked by writing the LOCK bit in WDOG_CTRL. Once locked, it cannot be disabled or reconfigured by software.

The watchdog counter is reset when EN is reset.

13.3.1 Clock Source

Three clock sources are available for use with the watchdog, through the CLKSEL field in WDOG_CTRL. The corresponding clocks must be enabled in the CMU. The SWOSCBLOCK bit in WDOG_CTRL can be written to prevent accidental disabling of the selected clocks. Also, setting this bit will automatically start the selected oscillator source when the watchdog is enabled. The PERSEL field in WDOG_CTRL is used to divide the selected watchdog clock, and the timeout for the watchdog timer can be calculated like this:

WDOG Timeout Equation

$$T_{TIMEOUT} = (2^{3+PERSEL} + 1)/f, \quad (13.1)$$

where f is the frequency of the selected clock.

It is recommended to clear the watchdog first, if PERSEL is changed while the watchdog is enabled.

To use this module, the LE interface clock must be enabled in CMU_HFCORECLKEN0, in addition to the module clock.

Note

Before changing the clock source for WDOG, the EN bit in WDOG_CTRL should be cleared. In addition to this, the WDOG_SYNCBUSY value should be zero.

13.3.2 Debug Functionality

The watchdog timer can either keep running or be frozen when the device is halted by a debugger. This configuration is done through the DEBUGRUN bit in WDOG_CTRL. When code execution is resumed, the watchdog will continue counting where it left off.

13.3.3 Energy Mode Handling

The watchdog timer can be configured to either keep on running or freeze when entering EM2 or EM3. The configuration is done individually for each energy mode in the EM2RUN and EM3RUN bits in WDOG_CTRL. When the watchdog has been frozen and is re-entering an energy mode where it is running, the watchdog timer will continue counting where it left off. For the watchdog there is no difference between EM0 and EM1. The watchdog does not run in EM4, and if EM4BLOCK in WDOG_CTRL is set, the CPU is prevented from entering EM4.

Note

If the WDOG is clocked by the LFXO or LFRCO, writing the SWOSCBLOCK bit will effectively prevent the CPU from entering EM3. When running from the ULFRCO, writing the SWOSCBLOCK bit will prevent the CPU from entering EM4.

13.3.4 Register access

Since this module is a Low Energy Peripheral, and runs off a clock which is asynchronous to the HFCORECLK, special considerations must be taken when accessing registers. Please refer to Section 6.3 (p. 61) for a description on how to perform register accesses to Low Energy Peripherals. note that clearing the EN bit in WDOG_CTRL will reset the WDOG module, which will halt any ongoing register synchronization.

Note

Never write to the WDOG registers when it is disabled, except to enable it by setting WDOG_CTRL_EN or when changing the clock source using WDOG_CTRL_CLKSEL. Make sure that the enable is registered (i.e. WDOG_SYNCBUSY_CTRL goes low), before writing other registers.

13.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	WDOG_CTRL	RW	Control Register
0x004	WDOG_CMD	W1	Command Register
0x008	WDOG_SYNCBUSY	R	Synchronization Busy Register

13.5 Register Description

13.5.1 WDOG_CTRL - Control Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																			0x0			0xF				0	0	0	0	0	0	0
Access																			RW			RW				RW	RW	RW	RW	RW	RW	RW
Name																			CLKSEL			PERSEL				SWOSCBLOCK	EM4BLOCK	LOCK	EM3RUN	EM2RUN	DEBUGRUN	EN

Bit	Name	Reset	Access	Description
31:14	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
13:12	CLKSEL	0x0	RW	**Watchdog Clock Select**

Selects the WDOG oscillator, i.e. the clock on which the watchdog will run.

Value	Mode	Description
0	ULFRCO	ULFRCO
1	LFRCO	LFRCO
2	LFXO	LFXO

11:8	PERSEL	0xF	RW	**Watchdog Timeout Period Select**

Select watchdog timeout period.

Value	Description
0	Timeout period of 9 watchdog clock cycles.
1	Timeout period of 17 watchdog clock cycles.
2	Timeout period of 33 watchdog clock cycles.
3	Timeout period of 65 watchdog clock cycles.
4	Timeout period of 129 watchdog clock cycles.
5	Timeout period of 257 watchdog clock cycles.
6	Timeout period of 513 watchdog clock cycles.
7	Timeout period of 1k watchdog clock cycles.
8	Timeout period of 2k watchdog clock cycles.
9	Timeout period of 4k watchdog clock cycles.
10	Timeout period of 8k watchdog clock cycles.
11	Timeout period of 16k watchdog clock cycles.
12	Timeout period of 32k watchdog clock cycles.
13	Timeout period of 64k watchdog clock cycles.
14	Timeout period of 128k watchdog clock cycles.
15	Timeout period of 256k watchdog clock cycles.

Bit	Name	Reset	Access	Description
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	SWOSCBLOCK	0	RW	**Software Oscillator Disable Block**

Set to disallow disabling of the selected WDOG oscillator. Writing this bit to 1 will turn on the selected WDOG oscillator if it is not already running.

Value	Description
0	Software is allowed to disable the selected WDOG oscillator. See CMU for detailed description. Note that also CMU registers are lockable.
1	Software is not allowed to disable the selected WDOG oscillator.

5	EM4BLOCK	0	RW	**Energy Mode 4 Block**

Set to prevent the EMU from entering EM4.

Value	Description
0	EM4 can be entered. See EMU for detailed description.
1	EM4 cannot be entered.

4	LOCK	0	RW	**Configuration lock**

Set to lock the watchdog configuration. This bit can only be cleared by reset.

Value	Description
0	Watchdog configuration can be changed.
1	Watchdog configuration cannot be changed.

3	EM3RUN	0	RW	**Energy Mode 3 Run Enable**

Set to keep watchdog running in EM3.

Value	Description
0	Watchdog timer is frozen in EM3.
1	Watchdog timer is running in EM3.

2	EM2RUN	0	RW	**Energy Mode 2 Run Enable**

Set to keep watchdog running in EM2.

Value	Description
0	Watchdog timer is frozen in EM2.
1	Watchdog timer is running in EM2.

1	DEBUGRUN	0	RW	**Debug Mode Run Enable**

Set to keep watchdog running in debug mode.

Value	Description
0	Watchdog timer is frozen in debug mode.
1	Watchdog timer is running in debug mode.

0	EN	0	RW	**Watchdog Timer Enable**

Set to enabled watchdog timer.

13.5.2 WDOG_CMD - Command Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																															
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																W1
Name																																CLEAR

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	CLEAR	0	W1	**Watchdog Timer Clear**

Clear watchdog timer. The bit must be written 4 watchdog cycles before the timeout.

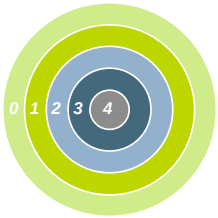
Value	Mode	Description
0	UNCHANGED	Watchdog timer is unchanged.
1	CLEARED	Watchdog timer is cleared to 0.

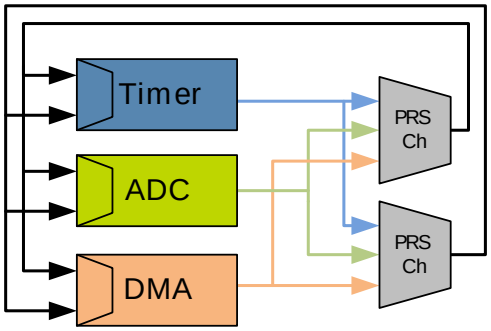
13.5.3 WDOG_SYNCBUSY - Synchronization Busy Register

Offset	Bit Position																														
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															R	R
Name																															CMD	CTRL

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	CMD	0	R	**CMD Register Busy**
	Set when the value written to CMD is being synchronized.			
0	CTRL	0	R	**CTRL Register Busy**
	Set when the value written to CTRL is being synchronized.			

14 PRS - Peripheral Reflex System





Quick Facts

What?

The PRS (Peripheral Reflex System) allows configurable, fast and autonomous communication between the peripherals.

Why?

Events and signals from one peripheral can be used as input signals or triggers by other peripherals and ensure timing-critical operation and reduced software overhead.

How?

Without CPU intervention the peripherals can send reflex signals (both pulses and level) to each other in single- or chained steps. The peripherals can be set up to perform actions based on the incoming reflex signals. This results in improved system performance and reduced energy consumption.

14.1 Introduction

The Peripheral Reflex System (PRS) system is a network which allows the different peripheral modules to communicate directly with each other without involving the CPU. Peripheral modules which send out reflex signals are called producers. The PRS routes these reflex signals to consumer peripherals which apply actions depending on the reflex signals received. The format for the reflex signals is not given, but edge triggers and other functionality can be applied by the PRS.

14.2 Features

- 4 configurable interconnect channels
  - Each channel can be connected to any producing peripheral
  - Consumers can choose which channel to listen to
  - Selectable edge detector (rising, falling and both edges)
- Software controlled channel output
  - Configurable level
  - Triggered pulses

14.3 Functional Description

An overview of the PRS module is shown in Figure 14.1 (p. 177) . The PRS contains 4 interconnect channels, and each of these can select between all the output reflex signals offered by the producers. The consumers can then choose which PRS channel to listen to and perform actions based on the reflex signals routed through that channel. The reflex signals can be both pulse signals and level signals. Synchronous PRS pulses are one HFPERCLK cycle long, and can either be sent out by a producer (e.g., ADC conversion complete) or be generated from the edge detector in the PRS channel. Level signals can have an arbitrary waveform (e.g., Timer PWM output).

14.3.1 Asynchronous Mode

Many reflex signals can operate in two modes, synchronous or asynchronous. A synchronous reflex is clocked on HFPERCLK, and can be used as an input to all reflex consumers, but since they require HFPERCLK, they will not work in EM2/EM3.

Asynchronous reflexes are not clocked on HFPERCLK, and can be used even in EM2/EM3. There is a limitation to reflexes operating in asynchronous mode though: they can only be used by a subset of the reflex consumers, the ones marked with async support in Table 14.2 (p. 179) . Peripherals that can produce asynchronous reflexes are marked with async support in Table 14.1 (p. 178) . To use these reflexes asynchronously, set ASYNC in the CHCTRL register for the PRS channel selecting the reflex signal.

Note

If a peripheral channel with ASYNC set is used in a consumer not supporting asynchronous reflexes, the behaviour is undefined.

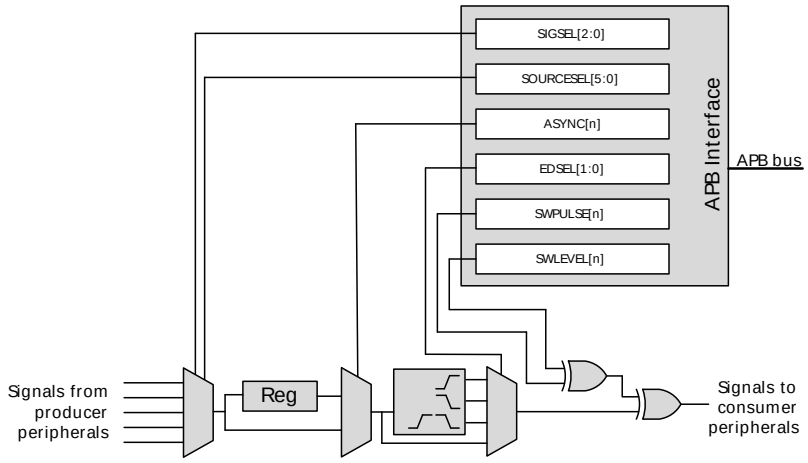
14.3.2 Channel Functions

Different functions can be applied to a reflex signal within the PRS. Each channel includes an edge detector to enable generation of pulse signals from level signals. It is also possible to generate output reflex signals by configuring the SWPULSE and SWLEVEL bits. SWLEVEL is a programmable level for each channel and holds the value it is programmed to. The SWPULSE will give out a one-cycle high pulse if it is written to 1, otherwise a 0 is asserted. The SWLEVEL and SWPULSE signals are then XOR'ed with the selected input from the producers to form the output signal sent to the consumers listening to the channel.

Note

The edge detector controlled by EDSEL should only be used when working with synchronous reflexes, i.e., ASYNC in CHCTRL is cleared.

Figure 14.1. PRS Overview



14.3.3 Producers

Each PRS channel can choose between signals from several producers, which is configured in SOURCESEL in PRS_CHx_CTRL. Each of these producers outputs one or more signals which can be selected by setting the SIGSEL field in PRS_CHx_CTRL. Setting the SOURCESEL bits to 0 (Off) leads to a constant 0 output from the input mux. An overview of the available producers is given in Table 14.1 (p. 178) .

Table 14.1. Reflex Producers

Module	Reflex Output	Output Format	Async Support
ADC	Single Conversion Done	Pulse	
	Scan Conversion Done	Pulse	
GPIO	Pin 0 Input	Level	Yes
	Pin 1 Input	Level	Yes
	Pin 2 Input	Level	Yes
	Pin 3 Input	Level	Yes
	Pin 4 Input	Level	Yes
	Pin 5 Input	Level	Yes
	Pin 6 Input	Level	Yes
	Pin 7 Input	Level	Yes
	Pin 8 Input	Level	Yes
	Pin 9 Input	Level	Yes
	Pin 10 Input	Level	Yes
	Pin 11 Input	Level	Yes
	Pin 12 Input	Level	Yes
	Pin 13 Input	Level	Yes
	Pin 14 Input	Level	Yes
	Pin 15 Input	Level	Yes
RTC	Overflow	Pulse	Yes
	Compare Match 0	Pulse	Yes
	Compare Match 1	Pulse	Yes
TIMER	Underflow	Pulse	
	Overflow	Pulse	
	CC0 Output	Level	
	CC1 Output	Level	
	CC2 Output	Level	
LETIMER	CH0	Level	Yes
	CH1	Level	Yes
USART	TX Complete	Pulse	
	RX Data Received	Pulse	
	IrDA Decoder Output	Level	
VCMP	Comparator Output	Level	Yes
USB	Start of Frame		Yes

Module	Reflex Output	Output Format	Async Support
	Start of Fram Sent/ Received		Yes

14.3.4 Consumers

Consumer peripherals (listed in Table 14.2 (p. 179)) can be set to listen to a PRS channel and perform an action based on the signal received on that channel. Most consumers expect pulse input, while some can handle level inputs as well.

Table 14.2. Reflex Consumers

Module	Reflex Input	Input Format	Async Support
ADC	Single Mode Trigger	Pulse	
	Scan Mode Trigger	Pulse	
IDAC	IDAC Enable	Level	Yes
TIMER	CC0 Input	Pulse/Level	
	CC1 Input	Pulse/Level	
	CC2 Input	Pulse/Level	
	DTI Fault Source 0 (TIMER0 only)	Pulse	
	DTI Fault Source 1 (TIMER0 only)	Pulse	
	DTI Input (TIMER0 only)	Pulse/Level	
USART	TX/RX Enable	Pulse	
	IrDA Encoder Input (USART0 only)	Pulse	
	RX Input	Pulse/Level	Yes
LEUART	RX Input	Pulse/Level	Yes
PCNT	S0 input	Level	Yes
	S1 input	Level	Yes

Note

It is possible to output prs channel 0 - channel 3 onto the GPIO by setting CH0PEN, CH1PEN, CH2PEN, or CH3PEN in the PRS_ROUTE register.

14.3.5 Example

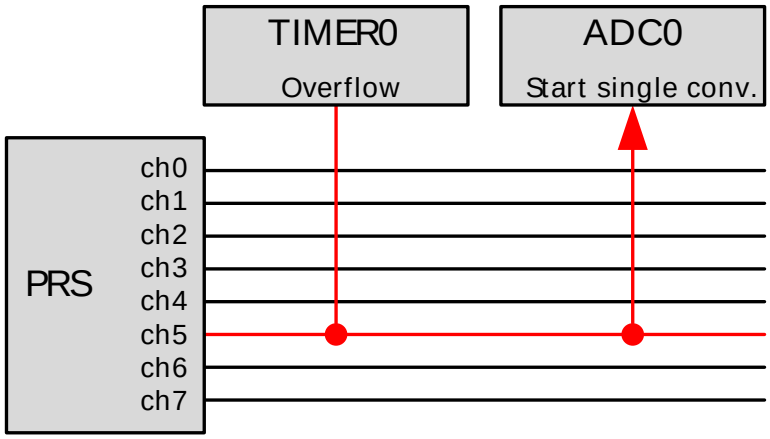
The example below (illustrated in Figure 14.2 (p. 180)) shows how to set up ADC0 to start single conversions every time TIMER0 overflows (one HFPERCLK cycle high pulse), using PRS channel 5:

- Set SOURCESEL in PRS_CH5_CTRL to 0b011100 to select TIMER0 as input to PRS channel 5.
- Set SIGSEL in PRS_CH5_CTRL to 0b001 to select the overflow signal (from TIMER0).
- Configure ADC0 with the desired conversion set-up.
- Set SINGLEPRSEN in ADC0_SINGLECTRL to 1 to enable single conversions to be started by a high PRS input signal.
- Set SINGLEPRSSEL in ADC0_SINGLECTRL to 0x5 to select PRS channel 5 as input to start the single conversion.

- Start TIMER0 with the desired TOP value, an overflow PRS signal is output automatically on overflow.

Note that the ADC results needs to be fetched either by the CPU or DMA.

Figure 14.2. TIMER0 overflow starting ADC0 single conversions through PRS channel 5.



14.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	PRS_SWPULSE	W1	Software Pulse Register
0x004	PRS_SWLEVEL	RW	Software Level Register
0x008	PRS_ROUTE	RW	I/O Routing Register
0x010	PRS_CH0_CTRL	RW	Channel Control Register
0x014	PRS_CH1_CTRL	RW	Channel Control Register
0x018	PRS_CH2_CTRL	RW	Channel Control Register
0x01C	PRS_CH3_CTRL	RW	Channel Control Register
0x020	PRS_CH4_CTRL	RW	Channel Control Register
0x024	PRS_CH5_CTRL	RW	Channel Control Register
0x040	PRS_TRACECTRL	RW	MTB Trace Control Register

14.5 Register Description

14.5.1 PRS_SWPULSE - Software Pulse Register

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											W1	W1	W1	W1	W1	W1
Name																											CH5PULSE	CH4PULSE	CH3PULSE	CH2PULSE	CH1PULSE	CH0PULSE

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5PULSE	0	W1	**Channel 5 Pulse Generation**
	See bit 0.			
4	CH4PULSE	0	W1	**Channel 4 Pulse Generation**
	See bit 0.			
3	CH3PULSE	0	W1	**Channel 3 Pulse Generation**
	See bit 0.			
2	CH2PULSE	0	W1	**Channel 2 Pulse Generation**
	See bit 0.			
1	CH1PULSE	0	W1	**Channel 1 Pulse Generation**
	See bit 0.			
0	CH0PULSE	0	W1	**Channel 0 Pulse Generation**
	Write to 1 to generate one HFPERCLK cycle high pulse. This pulse is XOR'ed with the corresponding bit in the SWLEVEL register and the selected PRS input signal to generate the channel output.			

14.5.2 PRS_SWLEVEL - Software Level Register

Offset	Bit Position																													
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											RW	RW	RW	RW	RW	RW
Name																											CH5LEVEL	CH4LEVEL	CH3LEVEL	CH2LEVEL	CH1LEVEL	CH0LEVEL

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	CH5LEVEL	0	RW	**Channel 5 Software Level**
	See bit 0.			
4	CH4LEVEL	0	RW	**Channel 4 Software Level**
	See bit 0.			
3	CH3LEVEL	0	RW	**Channel 3 Software Level**
	See bit 0.			
2	CH2LEVEL	0	RW	**Channel 2 Software Level**
	See bit 0.			
1	CH1LEVEL	0	RW	**Channel 1 Software Level**
	See bit 0.			
0	CH0LEVEL	0	RW	**Channel 0 Software Level**
	The value in this register is XOR'ed with the corresponding bit in the SWPULSE register and the selected PRS input signal to generate the channel output.			

14.5.3 PRS_ROUTE - I/O Routing Register

Offset	Bit Position																													
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																						0x0							0	0	0	0
Access																						RW							RW	RW	RW	RW
Name																						LOCATION							CH3PEN	CH2PEN	CH1PEN	CH0PEN

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	LOCATION	0x0	RW	**I/O Location**
	Decides the location of the PRS I/O pins.			

Value	Mode	Description
0	LOC0	Location 0
1	LOC1	Location 1
2	LOC2	Location 2
3	LOC3	Location 3

7:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	CH3PEN	0	RW	**CH3 Pin Enable**
	When set, GPIO output from PRS channel 3 is enabled			

Bit	Name	Reset	Access	Description
2	CH2PEN	0	RW	**CH2 Pin Enable**
	When set, GPIO output from PRS channel 2 is enabled			
1	CH1PEN	0	RW	**CH1 Pin Enable**
	When set, GPIO output from PRS channel 1 is enabled			
0	CH0PEN	0	RW	**CH0 Pin Enable**
	When set, GPIO output from PRS channel 0 is enabled			

14.5.4 PRS_CHx_CTRL - Channel Control Register

Offset	Bit Position																													
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset				0			0x0				0x00																			0x0		
Access				RW			RW				RW																			RW		
Name				ASYNC			EDSEL				SOURCESEL																			SIGSEL		

Bit	Name	Reset	Access	Description
31:29	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
28	ASYNC	0	RW	**Asynchronous reflex**
	Set to disable synchronization of this reflex signal			
27:26	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
25:24	EDSEL	0x0	RW	**Edge Detect Select**
	Select edge detection.			

Value	Mode	Description
0	OFF	Signal is left as it is
1	POSEDGE	A one HFPERCLK cycle pulse is generated for every positive edge of the incoming signal
2	NEGEDGE	A one HFPERCLK clock cycle pulse is generated for every negative edge of the incoming signal
3	BOTHEDGES	A one HFPERCLK clock cycle pulse is generated for every edge of the incoming signal

23:22	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
21:16	SOURCESEL	0x00	RW	**Source Select**
	Select input source to PRS channel.			

Value	Mode	Description
0b000000	NONE	No source selected
0b000001	VCMP	Voltage Comparator
0b001000	ADC0	Analog to Digital Converter 0
0b010000	USART0	Universal Synchronous/Asynchronous Receiver/Transmitter 0
0b010001	USARTRF1	Universal Synchronous/Asynchronous Receiver/Transmitter 1
0b011100	TIMER0	Timer 0
0b011101	TIMER1	Timer 1
0b011110	TIMER2	Timer 2
0b100100	USB	Universal Serial Bus Interface
0b101000	RTC	Real-Time Counter
0b110000	GPIOL	General purpose Input/Output
0b110001	GPIOH	General purpose Input/Output
0b110110	PCNT0	Pulse Counter 0

Bit	Name	Reset	Access	Description
15:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2:0	SIGSEL	0x0	RW	**Signal Select**

Select signal input to PRS channel.

Value	Mode	Description
SOURCESEL = 0b000000 (NONE)		
0bxxx	OFF	Channel input selection is turned off
SOURCESEL = 0b000001 (VCMP)		
0b000	VCMPOUT	Voltage comparator output VCMPOUT
SOURCESEL = 0b001000 (ADC0)		
0b000	ADC0SINGLE	ADC single conversion done ADC0SINGLE
0b001	ADC0SCAN	ADC scan conversion done ADC0SCAN
SOURCESEL = 0b010000 (USART0)		
0b000	USART0IRTX	USART 0 IRDA out USART0IRTX
0b001	USART0TXC	USART 0 TX complete USART0TXC
0b010	USART0RXDATAV	USART 0 RX Data Valid USART0RXDATAV
SOURCESEL = 0b010001 (USARTRF1)		
0b000	USARTRF1IRTX	USART 1 IRDA out USARTRF1IRTX
0b001	USARTRF1TXC	USART 1 TX complete USARTRF1TXC
0b010	USARTRF1RXDATAV	USART 1 RX Data Valid USARTRF1RXDATAV
SOURCESEL = 0b011100 (TIMER0)		
0b000	TIMER0UF	Timer 0 Underflow TIMER0UF
0b001	TIMER0OF	Timer 0 Overflow TIMER0OF
0b010	TIMER0CC0	Timer 0 Compare/Capture 0 TIMER0CC0
0b011	TIMER0CC1	Timer 0 Compare/Capture 1 TIMER0CC1
0b100	TIMER0CC2	Timer 0 Compare/Capture 2 TIMER0CC2
SOURCESEL = 0b011101 (TIMER1)		
0b000	TIMER1UF	Timer 1 Underflow TIMER1UF
0b001	TIMER1OF	Timer 1 Overflow TIMER1OF
0b010	TIMER1CC0	Timer 1 Compare/Capture 0 TIMER1CC0
0b011	TIMER1CC1	Timer 1 Compare/Capture 1 TIMER1CC1
0b100	TIMER1CC2	Timer 1 Compare/Capture 2 TIMER1CC2
SOURCESEL = 0b011110 (TIMER2)		
0b000	TIMER2UF	Timer 2 Underflow TIMER2UF
0b001	TIMER2OF	Timer 2 Overflow TIMER2OF
0b010	TIMER2CC0	Timer 2 Compare/Capture 0 TIMER2CC0
0b011	TIMER2CC1	Timer 2 Compare/Capture 1 TIMER2CC1
0b100	TIMER2CC2	Timer 2 Compare/Capture 2 TIMER2CC2
SOURCESEL = 0b100100 (USB)		
0b000	USBSOF	USB Start of Frame USBSOF
0b001	USBSOFSR	USB Start of Frame Sent/Received USBSOFSR
SOURCESEL = 0b101000 (RTC)		
0b000	RTCOF	RTC Overflow RTCOF
0b001	RTCCOMP0	RTC Compare 0 RTCCOMP0
0b010	RTCCOMP1	RTC Compare 1 RTCCOMP1
SOURCESEL = 0b110000 (GPIO)		
0b000	GPIOPIN0	GPIO pin 0 GPIOPIN0
0b001	GPIOPIN1	GPIO pin 1 GPIOPIN1
0b010	GPIOPIN2	GPIO pin 2 GPIOPIN2
0b011	GPIOPIN3	GPIO pin 3 GPIOPIN3
0b100	GPIOPIN4	GPIO pin 4 GPIOPIN4
0b101	GPIOPIN5	GPIO pin 5 GPIOPIN5

Bit	Name	Reset	Access	Description

Value	Mode	Description
0b110	GPIOPIN6	GPIO pin 6 GPIOPIN6
0b111	GPIOPIN7	GPIO pin 7 GPIOPIN7
SOURCESEL = 0b110001 (GPIO)		
0b000	GPIOPIN8	GPIO pin 8 GPIOPIN8
0b001	GPIOPIN9	GPIO pin 9 GPIOPIN9
0b010	GPIOPIN10	GPIO pin 10 GPIOPIN10
0b011	GPIOPIN11	GPIO pin 11 GPIOPIN11
0b100	GPIOPIN12	GPIO pin 12 GPIOPIN12
0b101	GPIOPIN13	GPIO pin 13 GPIOPIN13
0b110	GPIOPIN14	GPIO pin 14 GPIOPIN14
0b111	GPIOPIN15	GPIO pin 15 GPIOPIN15
SOURCESEL = 0b110110 (PCNT0)		
0b000	PCNT0TCC	Triggered compare match PCNT0TCC

14.5.5 PRS_TRACECTRL - MTB Trace Control Register

Offset	Bit Position																													
0x040	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																					0x0			0					0x0			0
Access																					RW			RW					RW			RW
Name																					TSTOP			TSTOPEN					TSTART			TSTARTEN

Bit	Name	Reset	Access	Description
31:12	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
11:9	TSTOP	0x0	RW	**MTB TSTOP PRS select**

Select PRS channel controlling the TSTOP signal to the MTB.

Value	Mode	Description
0	PRSCH0	PRS ch 0 is controlling TSTOP.
1	PRSCH1	PRS ch 1 is controlling TSTOP.
2	PRSCH2	PRS ch 2 is controlling TSTOP.
3	PRSCH3	PRS ch 3 is controlling TSTOP.
4	PRSCH4	PRS ch 4 is controlling TSTOP.
5	PRSCH5	PRS ch 5 is controlling TSTOP.

8	TSTOPEN	0	RW	**PRS TSTOP Enable**

Set PRS control of the TSTOP-signal going to the MTB.

Value	Description
0	TSTOP is not controlled by PRS.
1	TSTOP is controlled by PRS.

7:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3:1	TSTART	0x0	RW	**MTB TSTART PRS select**

Select PRS channel controlling the TSTART signal to the MTB.

Value	Mode	Description
0	PRSCH0	PRS ch 0 is controlling TSTART.
1	PRSCH1	PRS ch 1 is controlling TSTART.
2	PRSCH2	PRS ch 2 is controlling TSTART.
3	PRSCH3	PRS ch 3 is controlling TSTART.

Bit	Name	Reset	Access	Description

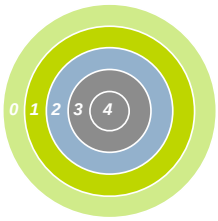
Value	Mode	Description
4	PRSCH4	PRS ch 4 is controlling TSTART.
5	PRSCH5	PRS ch 5 is controlling TSTART.

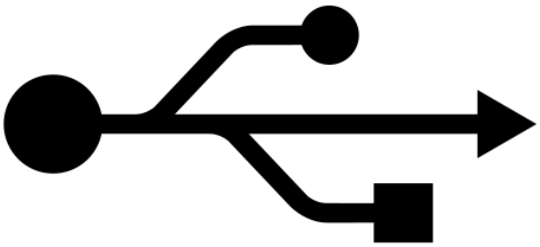
Bit	Name	Reset	Access	Description
0	TSTARTEN	0	RW	**PRS TSTART Enable**

Set PRS control of the TSTART-signal going to the MTB.

Value	Description
0	TSTART is not controlled by PRS.
1	TSTART is controlled by PRS.

15 USB - Universal Serial Bus Controller





Quick Facts

What?

The USB is a full-speed/low-speed USB 2.0 compliant USB Controller that can be used in various Device configurations. The on-chip 3.3V regulator delivers up to 50 mA and can also be used to power external components, eliminating the need for an external LDO. The on-chip regulator allows the system to run from a battery utilizing the full voltage range of the EZR32 still being compliant with the 3.3V +/- 10% USB voltage range.

Why?

USB provides a robust, industry-standard way to interface PCs and other portable devices.

How?

The flexible and highly software-configurable architecture of the USB Controller makes it easy to implement both device- and host-capable solutions. The on-chip OTG PHY with software controllable pull-up and pull-down resistors reduces the number of external components to a minimum. Third-party USB software stacks are also available, reducing the development time substantially. By utilizing the very low energy consumption in EM2, the USB device will be able to wake up and perform tasks several times a second without violating the 2.5 mA maximum average current during suspend.

15.1 Introduction

The USB is a full-speed/low-speed USB 2.0 compliant device controller. The architecture is very flexible and allows the USB to be used in various and Device-only configurations. The on-chip voltage regulator and PHY reduces the number of external components to a minimum.

15.2 Features

- Fully compliant with Universal Serial Bus Specification, Revision 2.0
- Supports full-speed (12 Mbit/s) and low-speed (1.5 Mbit/s) host and device
- Low Energy Mode, reducing the average current consumption with up to 90%.
- Dedicated Internal DMA Controller
- 6 software-configurable endpoints (3 IN, 3 OUT) in addition to endpoint 0
- 1 KB endpoint memory
- Resume/Reset detection in EM2 (during suspend)
- Soft connect/disconnect
- On-chip PHY

  - Internal pull-up and pull-down resistors
- Internal 3.3V Regulator
  - Output voltage: 3.3V
  - Output current: 50 mA
  - Input voltage range: 4.0 - 5.5V
  - Enabled automatically when input voltage applied
  - Low quiescent current: 100 uA
  - Output pin can be used to power the EZR32 itself as well as external components
  - Regulator voltage output sense feature for detecting USB plug/unplug events (also available in EM2/3)

15.3 USB System Description

A block diagram of the USB is shown in Figure 15.1 (p. 188) .

Figure 15.1. USB Block Diagram

The USB consists of a digital logic part, an endpoint RAM, PHY and a voltage regulator with output voltage sensor. The voltage regulator provides a stable 3.3 V supply for the PHY, but can also be used to power the EZR32 itself as well as external components.

The digital logic of the USB is split into two parts: system and core.

The system part is accessed using USB registers from offset 0x000 to 0x018 and controls the voltage regulator, Low Energy Mode and enabling/disabling of the PHY and USB pins. This part is clocked by $HFCORECLK_{USB}$ and is accessed using an APB slave interface. The system part can thus be accessed independently of the core part, without $HFCORECLK_{USBC}$ running.

The core part is clocked by $HFCORECLK_{USBC}$ and is accessed using an AHB slave interface. This interface is used for accessing the FIFO contents and the registers in the core part starting at offset 0x3C000. An additional master interface is used by the internal DMA controller of the core. The core part takes care of all the USB protocol related functionality. The clock to the system part must not be disabled when the core part is active.

The two AHB interfaces also features asynchronous AHB bridges, allowing the system and the USB Core to run at different clocks. Note that these asynchronous AHB bridges will add extra delay when transferring data over these interfaces. The asynchronous bridges will be bypassed when the HFCLK and $HFCORECLK_{USBC}$ are derived from the same clock source, allowing AHB transaction to complete in normal time.

There are several pins associated with the USB. USB_DP and USB_DM are the USB D+ and D- pins. These are the USB data signaling pins. USB_VREGI is the input to the voltage regulator and USB_VREGO is the regulated output. USB_DMPU is used to enable/disable an external D- pull-up resistor. This is needed for low-speed device only. USB_DMPU will be high-impedance until enabled from software. Thus, if a defined level is required during start-up an external pull-up/pull-down can be used.

15.3.1 USB Clocks

The USB requires the device to run a 24 MHz crystal (2500 ppm or better), or the Universal Serial High Frequency Oscillator (USHFRCO). The core part of the USB will always run from $HFCORECLK_{USBC}$, which is 48 MHz. The current consumption for the rest of the device can be reduced by dividing down HFCORECLK using the CMU_HFCORECLKDIV register, or by running the system on a different oscillator, at a lower speed. Bandwidth requirements for the specific USB application must be taken

into account when dividing down HFCORECLK. Bandwidth requirements must also be considered when using different oscillators for the USBC and the rest of the system, as this infers extra delay in the asynchronous AHB bridges.

15.3.2 USB Initialization

Follow these steps to enable the USB:

1. Enable the clock to the system part by setting USB in CMU_HFCORECLKEN0.
2. If the internal USB regulator is bypassed (by applying 3.3V on USB_VREGI and USB_VREGO externally), disable the regulator by setting VREGDIS in USB_CTRL.
3. If the PHY is powered from VBUS using the internal regulator, the VREGO sense circuit should be enabled by setting VREGOSEN in USB_CTRL.
4. Enable the USB PHY pins by setting PHYPEN in USB_ROUTE.
5. If low-speed device, set DMPUAP in USB_CTRL to the desired value and then enable the USB_DMPU pin in USB_ROUTE. Set the MODE for the pin to PUSHPULL.
6. Make sure the oscillator is ready and selected in CMU_CMD_USBCCLKSEL.
7. Enable the clock to the core part by setting USBC in CMU_HFCORECLKEN0.
8. Wait for the core to come out of reset. This is easiest done by polling a core register with non-zero reset value until it reads a non-zero value. This takes approximately 20 48-MHz cycles.
9. Start initializing the USB core as described in USB Core Description.

15.3.3 Configurations

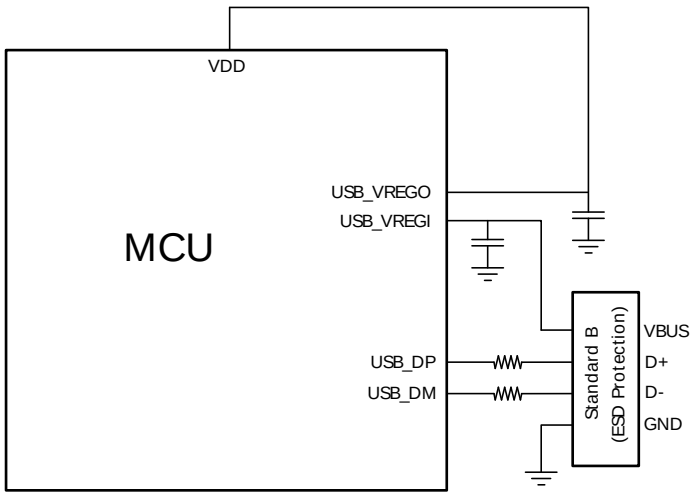
The USB is device-only, but with several power options The sections below describe the different configurations. External ESD protection and series resistors for impedance matching are required. The voltage regulator requires a 4.7 uF external decoupling capacitor on the input and a 1 uF external decoupling capacitor on the output. Decoupling not related to USB is not shown in the figures.

15.3.3.1 Bus-powered Device

A bus-powered device configuration is shown in Figure 15.2 (p. 189) . In this configuration the voltage regulator powers the PHY and the EZR32 at 3.3 V. The voltage regulator output (USB_VREGO) can also be used to power other components of the system.

In this configuration, the VREGO sense circuit should be left disabled.

Figure 15.2. Bus-powered Device

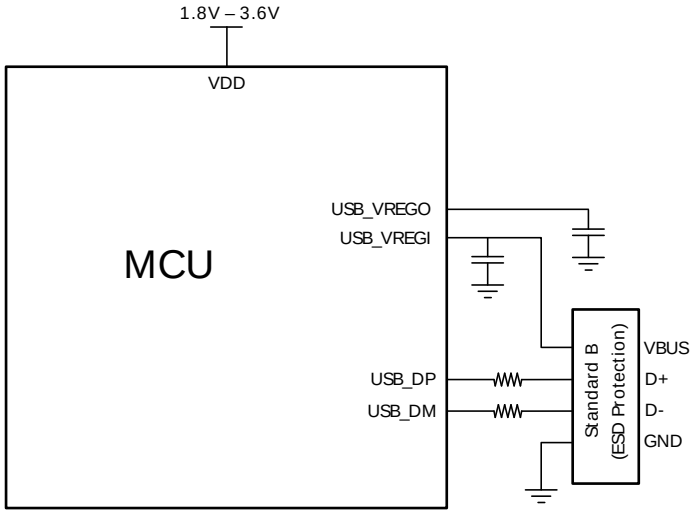


15.3.3.2 Self-powered Device

A self-powered device configuration is shown in Figure 15.3 (p. 190) . When the USB is configured as a self-powered device, the voltage regulator is typically used to power the PHY only, although it may also be used to power other 3.3 V components. When the USB is connected to a host, the voltage regulator is activated. Software can detect this event by enabling the VREGO Sense High (VREGOSH) interrupt. The PHY pins can then be enabled and USB traffic can start. The VREGO Sense Low (VREGOSL) interrupt can be used to detect when VBUS voltage disappears (for example if the USB cable is unplugged).

In this configuration, the VREGO sense circuit must be enabled.

Figure 15.3. Self-powered Device



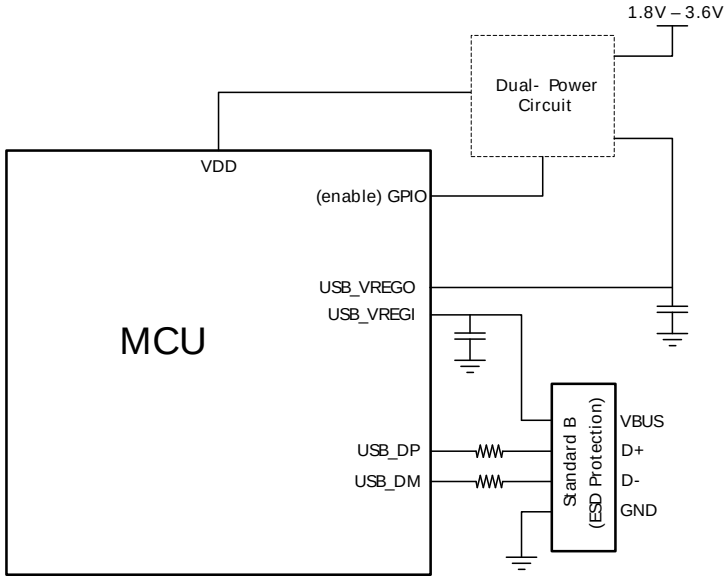
15.3.3.3 Self-powered Device (with bus-power switch)

A self-powered device (with bus-power switch) may switch power supply to VBUS when connected to a host. This is typically useful for extending the life of battery-powered devices and enables the use of coin-cell driven systems with low maximum peak current. The external components required typically include 2 transistors, 2 diodes and a few resistors. See application note for details. This allows seamless power supply switching between a battery and the voltage regulator output.

The VREGO Sense High interrupt is used to detect when VBUS becomes present. Software can then enable the external transistor connected to USB_VREGO, effectively switching the power source. A regular GPIO pin is used to control this transistor. If necessary, the application may have to reduce the current consumption before switching to the USB power source. If VBUS voltage is removed, the circuit switches automatically back to the battery power supply. If necessary software must react quickly to this event and reduce the current consumption (for example by reducing the clock frequency) to avoid excessive voltage drop. This configuration is shown in Figure 15.4 (p. 191) .

In this configuration, the VREGO sense circuit must be enabled.

Figure 15.4. Self-powered Device (with bus-power switch)



15.3.4 PHY

The USB includes an internal full-speed/low-speed PHY with built-in pull-up/pull-down resistors. During suspend, the PHY enters a low-power state where only the single-ended receivers are active. The PHY is disabled by default and should be enabled by setting PHYPEN in USB_ROUTE before the USB core clock is enabled.

The PHY is powered by the internal voltage regulator output (USB_VREGO). To power the PHY directly from an external source (for example an external 3.3 V LDO), connect both USB_VREGO and USB_VREGI to the external 3.3 V supply voltage. To stop the quiescent current present with the voltage regulator enabled in this configuration, disable the the regulator by setting VREGDIS in USB_CTRL after power up. Then the regulator is effectively bypassed.

When VREGO Sense is enabled, the PHY is automatically disabled internally when the VREGO Sense output is low. This will happen if VBUS-power disappears. The application can detect this by keeping the VREGO Sense Low Interrupt enabled. Note that PHYPEN in USB_ROUTE will not be set to 0 in this case. Also, the PHY must always be disabled manually when there is no voltage applied to VREGO.

15.3.5 Voltage Regulator

The voltage regulator is used to regulate the 5 V VBUS voltage down to 3.3 V which is the operating voltage for the PHY.

A decoupling capacitor is required on USB_VREGI and USB_VREGO. Note that the USB standard requires the total capacitance on VBUS to be 1 uF minimum and 10 uF maximum for regular devices. OTG devices can have maximum 6.5 uF capacitance on VBUS.

The voltage regulator is enabled by default and can thus be used to power the EZR32 itself. Systems not using the USB should disable the regulator by setting VREGDIS in USB_CTRL. A voltage sense circuit monitors the output voltage and can be used to detect when the voltage regulator becomes active. This sense circuit can also be used to detect when the voltage drops (typically due to the USB cable being unplugged). If regulator voltage monitoring is not required (i.e. it is known that the VREGO voltage is always present), the sense circuit should be left disabled.

During suspend, the bias current for the regulator can be reduced if the current requirements in EM2/3 are low. The bias current in EM2/3 is controlled by BIASPROGEM23 in USB_CTRL. When EM2/3 is entered, the bias current for the regulator switches to what is specified in BIASPROGEM23 in USB_CTRL. When

entering EM0 again (due to USB resume/reset signaling or any other wake-up interrupt) the regulator switches back to using the value specified in BIASPROGEM01 in USB_CTRL.

15.3.6 Interrupts and PRS

Interrupts from the core and system part share a common USB interrupt line to the CPU. The interrupt flags for the system part are grouped together in the USB_IF register. The interrupt events from the core are controlled by several core interrupt flag registers.

There are two PRS outputs from the USB: SOF and SOFSR. SOF toggles every time an SOF token is received from the USB host or when an SOF token is missed at the start of frame, while SOFSR toggles only when a valid SOF token is received from the USB host. Both PRS outputs must be synchronized in the PRS when used (i.e. it is an asynchronous PRS output). The edge-to-pulse converter in the PRS can be used to convert the edges into pulses if needed. The PRS outputs go to 0 in EM2/3.

15.3.7 USB Low Energy Mode

The USB also features a Low Energy Mode (LEM) that can be used to reduce the current consumption of the USB when there is no data on the lines that can be received. When such a condition is detected the USB can be configured to turn off the clock to the USB Core and possibly suspend the USHFRCO. Note that if the system is also running off of the USHFRCO, the oscillator will not be suspended when a Low Energy condition is detected.

The condition that can trigger Low Energy Mode is:

- Idle - There is no traffic on the lines.

This condition have the enable bit in the USB_CTRL; LEMIDLEEN.

Even though most of the USB operation is identical irregardless of whether Low Energy Mode is enabled, there are two subtle functional differences when Low Energy Mode is enabled: (1) higher access time to the core registers; and (2) no interrupts/PRS on missed SOFs. The higher access time to the core registers is only applicable when LEMOSCCTRL in USB_CTRL is set to SUSPEND, and is due to the fact that on a bus access, the system needs to restart the USHFRCO to complete the transfer. If the application will have several bus USB transactions in a short time, e.g. in the IRQ handler, it is recommended to set the LEMOSCCTRL to GATE during this time. Also, missed SOFs will not generate interrupts or PRS toggles when Low Energy Mode is enabled.

15.3.8 USB in EM2

During suspend and session-off EM2 should be used to save power and meet the average current requirements dictated by the USB standard. Before entering EM2, $HFCORECLK_{USBC}$ must be switched from 48 MHz to 32 kHz (LFXO or LFRCO). This is done using the CMU_CMD and CMU_STATUS registers. Upon EM2 wake-up, $HFCORECLK_{USBC}$ must be switched back to 48 MHz before accessing the core registers. The device always starts up from HFRCO so software must restart HFXO and switch from HFRCO to HFXO. The USB system clock, $HFCORECLK_{USB}$, must be kept enabled during EM2. The USB system registers can be accessed immediately upon EM2 wake-up, while running from HFRCO. Follow the steps outlined the USB Core Description when entering EM2 during suspend and session-off.

The FIFO content is lost when entering EM2. In addition, most of the USB core registers are reset and therefore need to be backed up in RAM.

EM3 cannot be used when the USB is active. However, EM3 can be used while waiting for the internal voltage regulator to be activated (i.e. VBUS becomes 5V).

15.4 USB Core Description

This section describes the programming requirements for the USB Core.

Important features/parameters for the core are:

- Device only
- Internal DMA (Buffer Pointer Based)
- Dedicated TX FIFOS for each endpoint in device mode
- 3 IN/OUT endpoints in addition to endpoint 0 (in device mode)
- Dynamic FIFO sizing
- Non-Periodic Request Queue Depth: 8

The core has the following limitations:

- Link Power Management (LPM) is not supported
- ADP is not supported
- No OTG support (HNP and SRP not supported).

15.4.1 Overview: Programming the Core

Each significant programming feature of the core is discussed in a separate section.

This chapter uses abbreviations for register names and their fields. For detailed information on registers, see Section 15.6 (p. 294) .

The application must perform a core initialization sequence. If the cable is connected during power-up, the Current Mode of Operation bit in the Core Interrupt register (USB_GINTSTS.CURMOD) reflects the mode. The core enters Host mode when an “A” plug is connected, or Device mode when a “B” plug is connected.

This section explains the initialization of the core after power-on. The application must follow the initialization sequence irrespective of Host or Device mode operation. All core global registers are initialized according to the core’s configuration.

1. Program the following fields in the Global AHB Configuration (USB_GAHBCFG) register.
   - DMA Mode bit
   - AHB Burst Length field
   - Global Interrupt Mask bit = 1
   - Non-periodic TxFIFO Empty Level (can be enabled only when the core is operating in Slave mode as a host.)
   - Periodic TxFIFO Empty Level (can be enabled only when the core is operating in Slave mode)
2. Program the following field in the Global Interrupt Mask (USB_GINTMSK) register:
   - USB_GINTMSK.RXFLVLMSK = 0
3. Program the following fields in USB_GUSBCFG register.
   - HNP Capable bit
   - SRP Capable bit
   - External HS PHY or Internal FS Serial PHY Selection bit
   - Time-Out Calibration field
   - USB Turnaround Time field
4. The software must unmask the following bits in the USB_GINTMSK register.
   - OTG Interrupt Mask
   - Mode Mismatch Interrupt Mask
5. The software can read the USB_GINTSTS.CURMOD bit to determine whether the core is operating in Host or Device mode. The software the follows either the Section 15.4.1.1 (p. 194) or Device Initialization (p. 195) sequence.

Note

The core is designed to be interrupt-driven. Polling interrupt mechanism is not recommended: this may result in undefined resolutions.

Note

In device mode, just after Power On Reset or a Soft Reset, the USB_GINTSTS.SOF bit is set to 1 for debug purposes. This status must be cleared and can be ignored.

15.4.1.1 Host Initialization

To initialize the core as host, the application must perform the following steps.

1. Program USB_GINTMSK.PRTINT to unmask.
2. Program the USB_HCFG register to select full-speed host.
3. Program the USB_HPRT.PRTPWR bit to 1. This drives VBUS on the USB.
4. Wait for the USB_HPRT.PRTCONNDET interrupt. This indicates that a device is connect to the port.
5. Program the USB_HPRT.PRTRST bit to 1. This starts the reset process.
6. Wait at least 10 ms for the reset process to complete.
7. Program the USB_HPRT.PRTRST bit to 0.
8. Wait for the USB_HPRT.PRTENCHNG interrupt.
9. Read the USB_HPRT.PRTSPD field to get the enumerated speed.
10. Program the USB_HFIR register with a value corresponding to the selected PHY clock. At this point, the host is up and running and the port register begins to report device disconnects, etc. The port is active with SOFs occurring down the enabled port.
11. Program the RXFSIZE register to select the size of the receive FIFO.
12. Program the NPTXFSIZE register to select the size and the start address of the Non-periodic Transmit FIFO for non-periodic transactions.
13. Program the USB_HPTXFSIZ register to select the size and start address of the Periodic Transmit FIFO for periodic transactions.

To communicate with devices, the system software must initialize and enable at least one channel as described in Device Initialization (p. 195) .

15.4.1.1.1 Host Connection

The following steps explain the host connection flow:

1. When the USB Cable is plugged to the Host port, the core triggers USB_GINTSTS.CONIDSTSCHNG interrupt.
2. When the Host application detects USB_GINTSTS.CONIDSTSCHNG interrupt, the application can perform one of the following actions:
   - Turn on VBUS by setting USB_HPRT.PRTPWR = 1 or
   - Wait for SRP Signaling from Device to turn on VBUS.
3. The PHY indicates VBUS power-on by detecting a VBUS valid voltage level.
4. When the Host Core detects the device connection, it triggers the Host Port Interrupt (USB_GINTSTS.PRTINT) to the application.
5. When USB_GINTSTS.PRTINT is triggered, the application reads the USB_HPRT register to check if the Port Connect Detected (USB_HPRT.PRTCONNDET) bit is set or not.

15.4.1.1.2 Host Disconnection

The following steps explain the host disconnection flow:

1. When the Device is disconnected from the USB Cable (but the cable is still connected to the USB host), the Core triggers USB_GINTSTS.DISCONNINT (Disconnect Detected) interrupt.

Note

If the USB cable is disconnected from the Host port without removing the device, the core generates an additional interrupt - USB_GINTSTS.CONIDSTSCHNG (Connector ID Status Change).

2. The Host application can choose to turn off the VBUS by programming USB_HPRT.PRTPWR = 0.

15.4.1.2 Device Initialization

The application must perform the following steps to initialize the core at device on, power on, or after a mode change from Host to Device.

1. Program the following fields in USB_DCFG register.
   - Device Speed
   - Non-Zero-Length Status OUT Handshake
   - Periodic Frame Interval
2. Program the USB_GINTMSK register to unmask the following interrupts.
   - USB Reset
   - Enumeration Done
   - Early Suspend
   - USB Suspend
3. Wait for the USB_GINTSTS.USBRST interrupt, which indicates a reset has been detected on the USB and lasts for about 10 ms. On receiving this interrupt, the application must perform the steps listed in Initialization on USB Reset (p. 228)
4. Wait for the USB_GINTSTS.ENUMDONE interrupt. This interrupt indicates the end of reset on the USB. On receiving this interrupt, the application must read the USB_DSTS register to determine the enumeration speed and perform the steps listed in Initialization on Enumeration Completion (p. 228)

At this point, the device is ready to accept SOF packets and perform control transfers on control endpoint 0.

15.4.1.2.1 Device Connection

The device connect process varies depending on the if the VBUS is on or off when the device is connected to the USB cable.

When VBUS is on When the Device is Connected

If VBUS is on when the device is connected to the USB cable, there is no SRP from the device. The device connection flow is as follows:

1. The device triggers the USB_GINTSTS.SESSREQINT [bit 30] interrupt bit.
2. When the device application detects the USB_GINTSTS.SESSREQINT interrupt, it programs the required bits in the USB_DCFG register.
3. When the Host drives Reset, the Device triggers USB_GINTSTS.USBRST [bit 12] on detecting the Reset. The host then follows the USB 2.0 Enumeration sequence.

When VBUS is off When the Device is Connected

If VBUS is off when the device is connected to the USB cable, the device initiates SRP in OTG Revision 1.3 mode. The device connection flow is as follows:

1. The application initiates SRP by writing the Session Request bit in the OTG Control and Status register. The core perform data-line pulsing followed by VBUS pulsing.
2. The host starts a new session by turning on VBUS, indicating SRP success. The core interrupts the application by setting the Session Request Success Status Change bit in the OTG Interrupt Status register.

3. The application reads the Session Request Success bit in the OTG Control and Status register and programs the required bits in USB_DCFG register.
4. When Host drives Reset, the Device triggers USB_GINTSTS.USBRST on detecting the Reset. The host then follows the USB 2.0 Enumeration sequence.

15.4.1.2.2 Device Disconnection

The device session ends when the USB cable is disconnected or if the VBUS is switched off by the Host.

The device disconnect flow is as follows:

1. When the USB cable is unplugged or when the VBUS is switched off by the Host, the Device core trigger USB_GINTSTS.OTGINT [bit 2] interrupt bit.
2. When the device application detects USB_GINTSTS.OTGINT interrupt, it checks that the USB_GOTGINT.SESENDDET (Session End Detected) bit is set to 1.

15.4.1.2.3 Device Soft Disconnection

The application can perform a soft disconnect by setting the Soft disconnect bit (SFTDISCON) in Device Control Register (USB_DCTL).

Send/Receive USB Transfers -> Soft disconnect->Soft reset->USB Device Enumeration

Sequence of operations:

1. The application configures the device to send or receive transfers.
2. The application sets the Soft disconnect bit (SFTDISCON) in the Device Control Register (USB_DCTL).
3. The application sets the Soft Reset bit (CSFTRST) in the Reset Register (USB_GRSTCTL).
4. Poll the USB_GRSTCTL register until the core clears the soft reset bit, which ensures the soft reset is completed properly.
5. Initialize the core according to the instructions in Device Initialization (p. 195) .

Suspend-> Soft disconnect->Soft reset->USB Device Enumeration

Sequence of operations:

1. The core detects a USB suspend and generates a Suspend Detected interrupt.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core puts the PHY in suspend mode, and the PHY clock stops.
3. The application clears the Stop PHY Clock bit in the Power and Clock Gating Control register, and waits for the PHY clock to come back. The core takes the PHY back to normal mode, and the PHY clock comes back.
4. The application sets the Soft disconnect bit (SFTDISCON) in Device Control Register (USB_DCTL).
5. The application sets the Soft Reset bit (CSFTRST) in the Reset Register (USB_GRSTCTL).
6. Poll the USB_GRSTCTL register until the core clears the soft reset bit, which ensures the soft reset is completed properly.
7. Initialize the core according to the instructions in Device Initialization (p. 195) .

15.4.2 Modes of operation

- Overview: DMA/Slave modes (p. 197)
- DMA Mode (p. 197)
- Slave Mode (p. 197)

15.4.2.1 Overview: DMA/Slave modes

The application can operate the core in either of two modes:

- In DMA Mode (p. 197) - The core fetches the data to be transmitted or updates the received data on the AHB.
- In Slave Mode (p. 197) — The application initiates the data transfers for data fetch and store.

15.4.2.2 DMA Mode

In DMA Mode, the OTG host uses the AHB master Interface for transmit packet data fetch (AHB to USB) and receive data update (USB to AHB). The AHB master uses the programmed DMA address (USB_HCx_DMAADDR register in host mode and USB_DIEPx_DMAADDR/USB_DOEPx_DMAADDR register in device mode) to access the data buffers.

15.4.2.2.1 Transfer-Level Operation

In DMA mode, the application is interrupted only after the programmed transfer size is transmitted or received (provided the core detects no NAK/Timeout/Error response in Host mode, or Timeout/CRC Error in Device mode). The application must handle all transaction errors. In Device mode, all the USB errors are handled by the core itself.

15.4.2.2.2 Transaction-Level Operation

This mode is similar to transfer-level operation with the programmed transfer size equal to one packet size (either maximum packet size, or a short packet size).

15.4.2.3 Slave Mode

In Slave mode, the application can operate the core either in transaction-level (packet-level) operation or in pipelined transaction-level operation.

15.4.2.3.1 Transaction-Level Operation

The application handles one data packet at a time per channel/endpoint in transaction-level operations. Based on the handshake response received on the USB, the application determines whether to retry the transaction or proceed with the next, until the end of the transfer. The application is interrupted on completion of every packet. The application performs transaction-level operations for a channel/endpoint for a transmission (host: OUT/device: IN) or reception (host: IN/device: OUT) as shown in Figure 15.5 (p. 198) and Figure 15.6 (p. 198) .

Host Mode
For an OUT transaction, the application enables the channel and writes the data packet into the corresponding (Periodic or Non-periodic) transmit FIFO. The core automatically writes the channel number into the corresponding (Periodic or Non-periodic) Request Queue, along with the last DWORD write of the packet. For an IN transaction, the application enables the channel and the core automatically writes the channel number into the corresponding Request queue. The application must wait for the packet received interrupt, then empty the packet from the receive FIFO.

Device Mode
For an IN transaction, the application enables the endpoint, writes the data packet into the corresponding transmit FIFO, and waits for the packet completion interrupt from the core. For an OUT transaction, the application enables the endpoint, waits for the packet received interrupt from the core, then empties the packet from the receive FIFO.

Note

The application has to finish writing one complete packet before switching to a different channel/endpoint FIFO. Violating this rule results in an error.

Figure 15.5. Transmit Transaction-Level Operation in Slave Mode

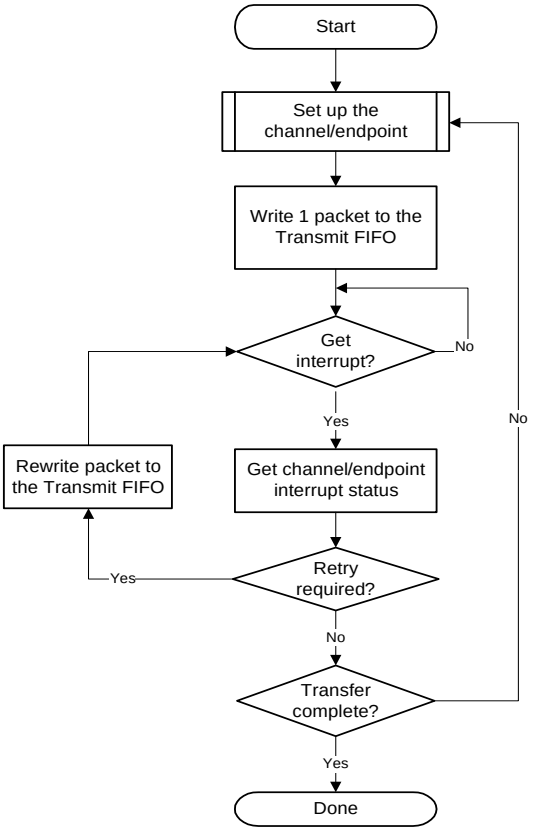
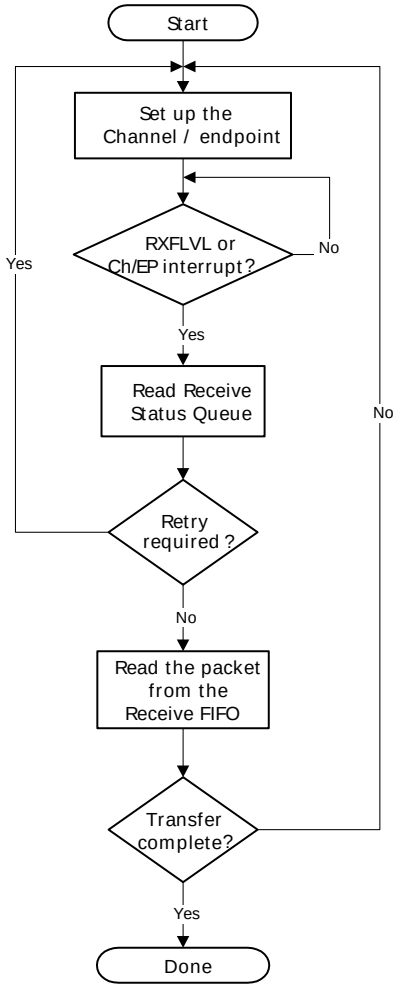


Figure 15.6. Receive Transaction-Level Operation in Slave Mode



15.4.2.3.2 Pipelined Transaction-Level Operation

The application can pipeline more than one transaction (IN or OUT) with pipelined transaction-level operation, which is analogous to Transfer mode in DMA mode. In pipelined transaction-level operation, the application can program the core to perform multiple transactions. The advantage of this mode compared to transaction-level operation is that the application is not interrupted on a packet basis.

15.4.2.3.2.1 Host mode

For an OUT transaction, the application sets up a transfer and enables the channel. The application can write multiple packets back-to-back for the same channel into the transmit FIFO, based on the space availability. It can also pipeline OUT transactions for multiple channels by writing into the HCHARn register, followed by a packet write to that channel. The core writes the channel number, along with the last DWORD write for the packet, into the Request queue and schedules transactions on the USB in the same order.

For an IN transaction, the application sets up a transfer and enables the channel, and the core writes the channel number into the Request queue. The application can schedule IN transactions on multiple channels, provided space is available in the Request queue. The core initiates an IN token on the USB only when there is enough space to receive at least of one maximum-packet-size packet of the channel in the top of the Request queue.

15.4.2.3.2.2 Device mode

For an IN transaction, the application sets up a transfer and enables the endpoint. The application can write multiple packets back-to-back for the same endpoint into the transmit FIFO, based on available space. It can also pipeline IN transactions for multiple channels by writing into the USB_DIEPx_CTL register followed by a packet write to that endpoint. The core writes the endpoint number, along with the last DWORD write for the packet into the Request queue. The core transmits the data in the transmit FIFO when an IN token is received on the USB.

For an OUT transaction, the application sets up a transfer and enables the endpoint. The core receives the OUT data into the receive FIFO, when it has available space. As the packets are received into the FIFO, the application must empty data from it.

From this point on in this chapter, the terms “Pipelined Transaction mode” and “Transfer mode” are used interchangeably.

15.4.3 Host Programming Model

Before you program the Host, read Overview: Programming the Core (p. 193) and Modes of operation (p. 196) .

This section discusses the following topics:

- Channel Initialization (p. 199)
- Halting a Channel (p. 200)
- Zero-Length Packets (p. 201)
- Handling Babble Conditions (p. 201)
- Handling Disconnects (p. 201)
- Host Programming Operations (p. 201)
  - Writing the Transmit FIFO in Slave Mode (p. 202)
  - Reading the Receive FIFO in Slave Mode (p. 203)

15.4.3.1 Channel Initialization

The application must initialize one or more channels before it can communicate with connected devices. To initialize and enable a channel, the application must perform the following steps.

1. Program the USB_GINTMSK register to unmask the following:
2. Channel Interrupt
   - Non-periodic Transmit FIFO Empty for OUT transactions (applicable for Slave mode that operates in pipelined transaction-level with the Packet Count field programmed with more than one).
   - Non-periodic Transmit FIFO Half-Empty for OUT transactions (applicable for Slave mode that operates in pipelined transaction-level with the Packet Count field programmed with more than one).
3. Program the USB_USB_HAINTMSK register to unmask the selected channels' interrupts.
4. Program the HCINTMSK register to unmask the transaction-related interrupts of interest given in the Host Channel Interrupt register.
5. Program the selected channel's USB_HCx_TSIZ register.

   Program the register with the total transfer size, in bytes, and the expected number of packets, including short packets. The application must program the PID field with the initial data PID (to be used on the first OUT transaction or to be expected from the first IN transaction).
6. Program the selected channels' USB_HCx_DMAADDR register(s) with the buffer start address (DMA mode only).
7. Program the USB_HCx_CHAR register of the selected channel with the device's endpoint characteristics, such as type, speed, direction, and so forth. (The channel can be enabled by setting the Channel Enable bit to 1 only when the application is ready to transmit or receive any packet).

Repeat the above steps for other channels.

Note

De-allocate channel means after the transfer has completed, the channel is disabled. When the application is ready to start the next transfer, the application re-initializes the channel by following these steps.

15.4.3.2 Halting a Channel

The application can disable any channel by programming the USB_HCx_CHAR register with the USB_HCx_CHAR.CHDIS and USB_HCx_CHAR.CHENA bits set to 1. This enables the host to flush the posted requests (if any) and generates a Channel Halted interrupt. The application must wait for the USB_HCx_INT.CHHLTD interrupt before reallocating the channel for other transactions. The host does not interrupt the transaction that has been already started on USB.

In Slave mode operation, before disabling a channel, the application must ensure that there is at least one free space available in the Non-periodic Request Queue (when disabling a non-periodic channel) or the Periodic Request Queue (when disabling a periodic channel). The application can simply flush the posted requests when the Request queue is full (before disabling the channel), by programming the USB_HCx_CHAR register with the USB_HCx_CHAR.CHDIS bit set to 1, and the USB_HCx_CHAR.CHENA bit reset to 0.

The core generates a RXFLVL interrupt when there is an entry in the queue. The application must read/pop the USB_GRXSTSP register to generate the Channel Halted interrupt.

To disable a channel in DMA mode operation, the application need not check for space in the Request queue. The host checks for space in which to write the Disable request on the disabled channel's turn during arbitration. Meanwhile, all posted requests are dropped from the Request queue when the USB_HCx_CHAR.CHDIS bit is set to 1.

The application is expected to disable a channel under any of the following conditions:

1. When a USB_HCx_INT.XFERCOMPL interrupt is received during a non-periodic IN transfer or high-bandwidth interrupt IN transfer (Slave mode only)
2. When a USB_HCx_INT.STALL, USB_HCx_INT.XACTERR, USB_HCx_INT.BBLERR, or USB_HCx_INT.DATATGLERR interrupt is received for an IN or OUT channel (Slave mode only).

For high-bandwidth interrupt INs in Slave mode, once the application has received a DATATGLERR interrupt it must disable the channel and wait for a Channel Halted interrupt. The application must be able to receive other interrupts (DATATGLERR, NAK, Data, XACTERR, BBLERR) for the same channel before receiving the halt.

3. When a USB_GINTSTS.DISCONNINT (Disconnect Device) interrupt is received. The application must check for the USB_HPRT.PRTCONNSTS, because when the device directly connected to the host is disconnected, USB_HPRT.PRTCONNSTS is reset. The software must issue a soft reset to ensure that all channels are cleared. When the device is reconnected, the host must issue a USB Reset.
4. When the application aborts a transfer before normal completion (Slave and DMA modes).

Note

In DMA mode, keep the following guideline in mind:

- Channel disable must not be programmed for periodic channels. At the end of the next frame (in the worst case), the core generates a channel halted and disables the channel automatically.

15.4.3.3 Sending a Zero-Length Packet in Slave/DMA Modes

To send a zero-length data packet, the application must initialize an OUT channel as follows.

1. Program the USB_HCx_TSIZ register of the selected channel with a correct PID, XFERSIZE = 0, and PKTCNT = 1.
2. Program the USB_HCx_CHAR register of the selected channel with CHENA = 1 and the device's endpoint characteristics, such as type, speed, and direction.

The application must treat a zero-length data packet as a separate transfer, and cannot combine it with a non-zero-length transfer.

15.4.3.4 Handling Babble Conditions

The core handles two cases of babble: packet babble and port babble. Packet babble occurs if the device sends more data than the maximum packet size for the channel. Port babble occurs if the core continues to receive data from the device at EOF2 (the end of frame 2, which is very close to SOF).

When the core detects a packet babble, it stops writing data into the Rx buffer and waits for the end of packet (EOP). When it detects an EOP, it flushes already-written data in the Rx buffer and generates a Babble interrupt to the application.

When detects a port babble, it flushes the RxFIFO and disables the port. The core then generates a Port Disabled Interrupt (USB_GINTSTS.PRTINT, USB_HPRT.PRTENCHNG). On receiving this interrupt, the application must determine that this is not due to an overcurrent condition (another cause of the Port Disabled interrupt) by checking USB_HPRT.PRTOVRCURRACT, then perform a soft reset. The core does not send any more tokens after it has detected a port babble condition.

15.4.3.5 Handling Disconnects

If the device is disconnected suddenly, a USB_GINTSTS.DISCONNINT interrupt is generated. When the application receives this interrupt, it must issue a soft reset by programming the USB_GRSTCTL.CSFTRST bit.

15.4.3.6 Host Programming Operations

Table 15.1 (p. 202) provides links to the programming sequence for the different types of USB transactions.

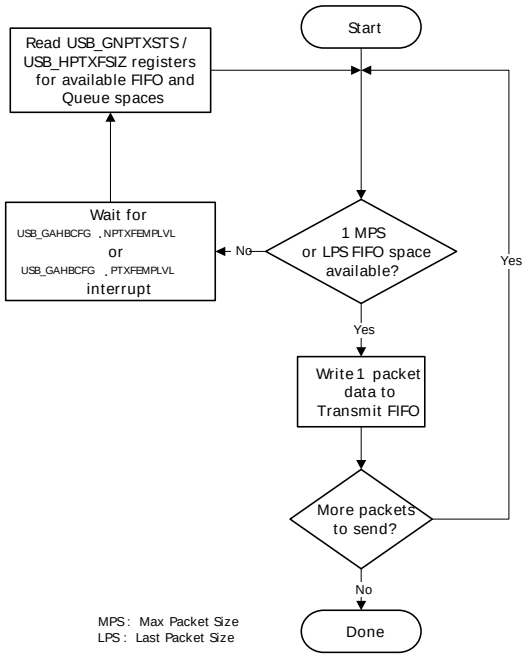
Table 15.1. Host Programming Operations

Mode	IN	OUT/SETUP
Control		
Slave	Bulk and Control IN Transactions in Slave Mode (p. 206)	Bulk and Control OUT/SETUP Transactions in Slave Mode (p. 204)
DMA	Bulk and Control IN Transactions in DMA Mode (p. 212)	Bulk and Control OUT/SETUP Transactions in DMA Mode (p. 208)
Bulk		
Slave	Bulk and Control IN Transactions in Slave Mode (p. 206)	Bulk and Control OUT/SETUP Transactions in Slave Mode (p. 204)
DMA	Bulk and Control IN Transactions in DMA Mode (p. 212)	Bulk and Control OUT/SETUP Transactions in DMA Mode (p. 208)
Interrupt		
Slave	Interrupt IN Transactions in Slave Mode (p. 216)	Interrupt OUT Transactions in Slave Mode (p. 214)
DMA	Interrupt IN Transactions in DMA Mode (p. 220)	Interrupt OUT Transactions in DMA Mode (p. 218)
Isochronous		
Slave	Isochronous IN Transactions in Slave Mode (p. 224)	Isochronous OUT Transactions in Slave Mode (p. 222)
DMA	Isochronous IN Transactions in DMA Mode (p. 226)	Isochronous OUT Transactions in DMA Mode (p. 225)

15.4.3.6.1 Writing the Transmit FIFO in Slave Mode

Figure 15.7 (p. 203) shows the flow diagram for writing to the transmit FIFO in Slave mode. The host automatically writes an entry (OUT request) to the Periodic/Non-periodic Request Queue, along with the last DWORD write of a packet. The application must ensure that at least one free space is available in the Periodic/Non-periodic Request Queue before starting to write to the transmit FIFO. The application must always write to the transmit FIFO in DWORDs. If the packet size is non-DWORD aligned, the application must use padding. The host determines the actual packet size based on the programmed maximum packet size and transfer size.

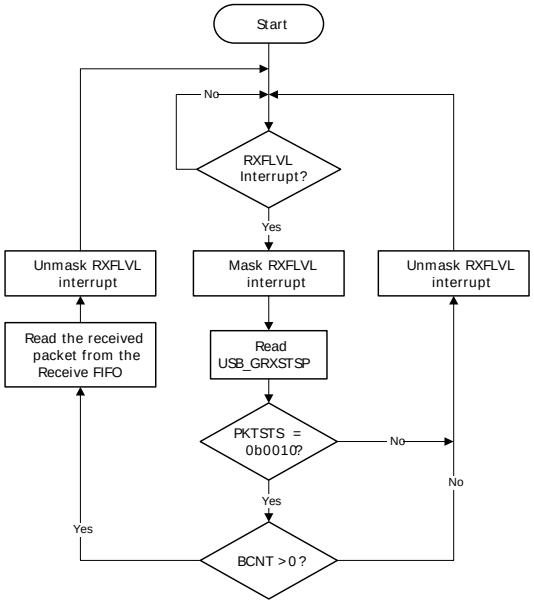
Figure 15.7. Transmit FIFO Write Task in Slave Mode



15.4.3.6.2 Reading the Receive FIFO in Slave Mode

Figure 15.8 (p. 203) shows the flow diagram for reading the receive FIFO in Slave mode. The application must ignore all packet statuses other than IN Data Packet (0b0010).

Figure 15.8. Receive FIFO Read Task in Slave Mode



15.4.3.6.3 Control Transactions in Slave Mode

Setup, Data, and Status stages of a control transfer must be performed as three separate transfers. Setup- Data- or Status-stage OUT transactions are performed similarly to the bulk OUT transactions explained in Bulk and Control OUT/SETUP Transactions in Slave Mode(p. 204) . Data- or Status-stage IN transactions are performed similarly to the bulk IN transactions explained in Bulk and Control IN Transactions in Slave Mode (p. 206) For all three stages, the application is expected to set the USB_HC1_CHAR.EPTYPE field to Control. During the Setup stage, the application is expected to set the USB_HC1_TSIZ.PID field to SETUP.

15.4.3.6.4 Bulk and Control OUT/SETUP Transactions in Slave Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the connected device, it must initialize a channel as described in Channel Initialization (p. 199). See Figure 15.7 (p. 203) and Figure 15.8 (p. 203) for Read or Write data to and from the FIFO in Slave mode.

A typical bulk or control OUT/SETUP pipelined transaction-level operation in Slave mode is shown in Figure 15.9 (p. 205). See channel 1 (ch_1). Two bulk OUT packets are transmitted. A control SETUP transaction operates the same way but has only one packet. The assumptions are:

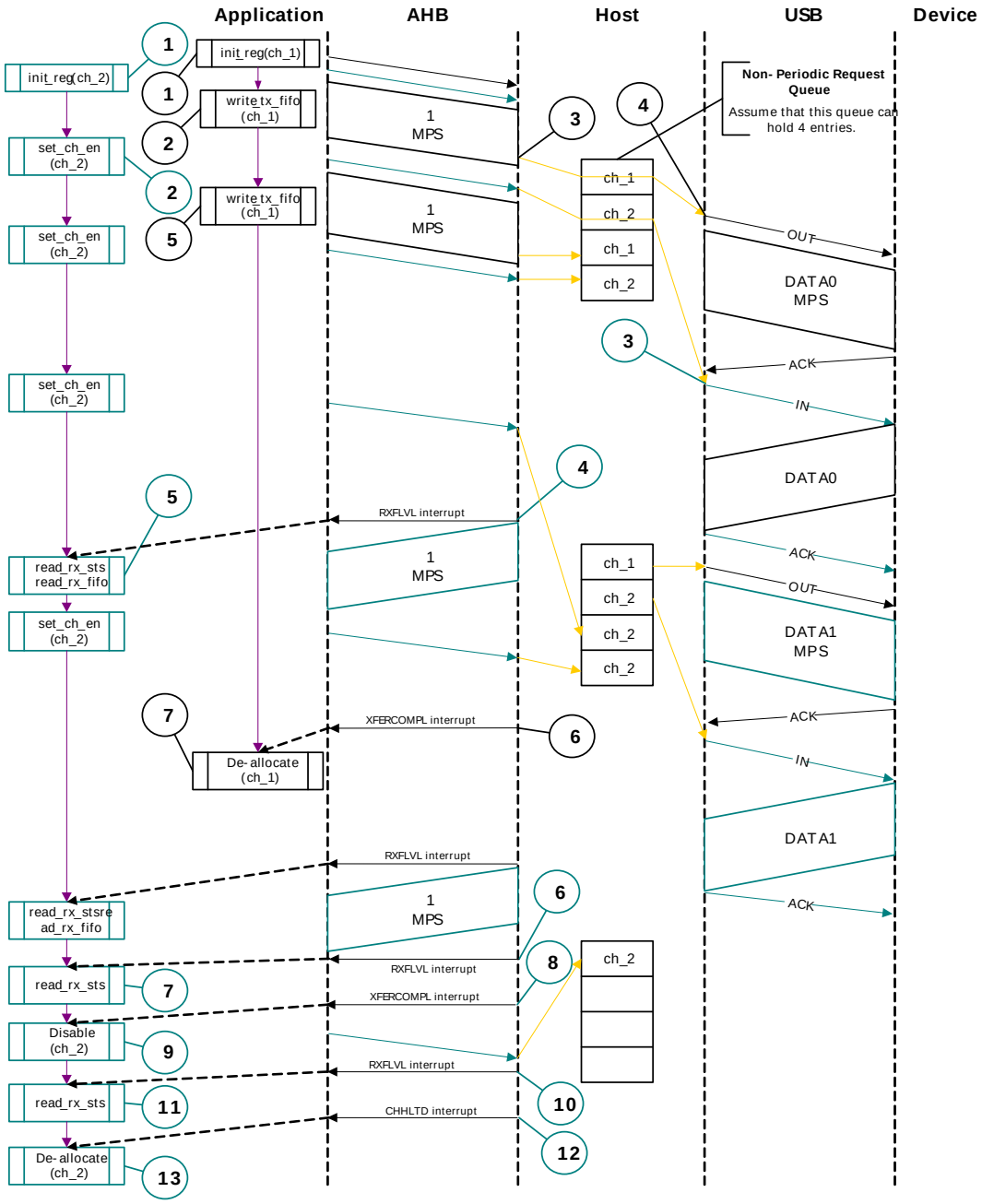
- The application is attempting to send two maximum-packet-size packets (transfer size = 1,024 bytes).
- The Non-periodic Transmit FIFO can hold two packets (128 bytes for FS).
- The Non-periodic Request Queue depth = 4.

15.4.3.6.4.1 Normal Bulk and Control OUT/SETUP Operations

The sequence of operations in Figure 15.9 (p. 205) (channel 1) is as follows:

1. Initialize channel 1 as explained in Channel Initialization (p. 199) .
2. Write the first packet for channel 1.
3. Along with the last DWORD write, the core writes an entry to the Non-periodic Request Queue.
4. As soon as the non-periodic queue becomes non-empty, the core attempts to send an OUT token in the current frame.
5. Write the second (last) packet for channel 1.
6. The core generates the XFERCOMPL interrupt as soon as the last transaction is completed successfully.
7. In response to the XFERCOMPL interrupt, de-allocate the channel for other transfers.

Figure 15.9. Normal Bulk/Control OUT/SETUP and Bulk/Control IN Transactions in Slave Mode



15.4.3.6.4.2 Handling Interrupts

The channel-specific interrupt service routine for bulk and control OUT/SETUP transactions in Slave mode is shown in the following code samples.

Interrupt Service Routine for Bulk/Control OUT/SETUP Transactions in Slave Mode

Bulk/Control OUT/SETUP

```
Unmask (NAK/XACTERR/STALL/XFERCOMPL)
if (XFERCOMPL)
{
    Reset Error Count
    Mask ACK
    De-allocate Channel
}
else if (STALL)
```

```
{
    Transfer Done = 1
    Unmask CHHLTD
    Disable Channel
}
else if (NAK or XACTERR)
{
    Rewind Buffer Pointers
    Unmask CHHLTD
    Disable Channel
    if (XACTERR)
    {
        Increment Error Count
        Unmask ACK
    }
    else
    {
        Reset Error Count
    }
}
else if (CHHLTD)
{
    Mask CHHLTD
    if (Transfer Done or (Error_count == 3))
    {
        De-allocate Channel
    }
    else
    {
        Re-initialize Channel
    }
}
else if (ACK)
{
    Reset Error Count
    Mask ACK
}
```

The application is expected to write the data packets into the transmit FIFO when space is available in the transmit FIFO and the Request queue. The application can make use of USB_GINTSTS.NPTXFEMP interrupt to find the transmit FIFO space.

The application is expected to write the requests as and when the Request queue space is available and until the XFERCOMPL interrupt is received.

15.4.3.6.5 Bulk and Control IN Transactions in Slave Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the connected device, it must initialize a channel as described in Channel Initialization (p. 199). See Figure 15.7 (p. 203) and Figure 15.8 (p. 203) for read or write data to and from the FIFO in Slave mode.

A typical bulk or control IN pipelined transaction-level operation in Slave mode is shown in Figure 15.9 (p. 205) . See channel 2 (ch_2). The assumptions are:

1. The application is attempting to receive two maximum-sized packets (transfer size = 1,024 bytes).
2. The receive FIFO can contain at least one maximum-packet-size packet and two status DWORDs per packet (72 bytes for FS).
3. The Non-periodic Request Queue depth = 4.

15.4.3.6.5.1 Normal Bulk and Control IN Operations

The sequence of operations in Figure 15.9 (p. 205) is as follows:

1. Initialize channel 2 as explained in Channel Initialization (p. 199) .
2. Set the USB_HC2_CHAR.CHENA bit to write an IN request to the Non-periodic Request Queue.
3. The core attempts to send an IN token after completing the current OUT transaction.
4. The core generates an RXFLVL interrupt as soon as the received packet is written to the receive FIFO.
5. In response to the RXFLVL interrupt, mask the RXFLVL interrupt and read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. Following this, unmask the RXFLVL interrupt.
6. The core generates the RXFLVL interrupt for the transfer completion status entry in the receive FIFO.
7. The application must read and ignore the receive packet status when the receive packet status is not an IN data packet (USB_GRXSTSR.PKTSTS != 0b0010).
8. The core generates the XFERCOMPL interrupt as soon as the receive packet status is read.
9. In response to the XFERCOMPL interrupt, disable the channel (see Halting a Channel (p. 200)) and stop writing the USB_HC2_CHAR register for further requests. The core writes a channel disable request to the non-periodic request queue as soon as the USB_HC2_CHAR register is written.
10. The core generates the RXFLVL interrupt as soon as the halt status is written to the receive FIFO.
11. Read and ignore the receive packet status.
12. The core generates a CHHLTD interrupt as soon as the halt status is popped from the receive FIFO.
13. In response to the CHHLTD interrupt, de-allocate the channel for other transfers.

Note

For Bulk/Control IN transfers, the application must write the requests when the Request queue space is available, and until the XFERCOMPL interrupt is received.

15.4.3.6.5.2 Handling Interrupts

The channel-specific interrupt service routine for bulk and control IN transactions in Slave mode is shown in the following code samples.

Interrupt Service Routine for Bulk/Control IN Transactions in Slave Mode

```
Unmask (XACTERR/XFERCOMPL/BBLERR/STALL/DATATGLERR)
if (XFERCOMPL)
{
    Reset Error Count
    Unmask CHHLTD
    Disable Channel
    Reset Error Count
    Mask ACK
}
else if (XACTERR or BBLERR or STALL)
{
    Unmask CHHLTD
    Disable Channel
    if (XACTERR)
    {
        Increment Error Count
        Unmask ACK
    }
}
else if (CHHLTD)
{
    Mask CHHLTD
    if (Transfer Done or (Error_count == 3))
    {
        De-allocate Channel
    }
    else
    {
        Re-initialize Channel
    }
```

```
}
else if (ACK)
{
    Reset Error Count
    Mask ACK
}
else if (DATATGLERR)
{
    Reset Error Count
}
```

15.4.3.6.6 Control Transactions in DMA Mode

Setup, Data, and Status stages of a control transfer must be performed as three separate transfers. Setup- and Data- or Status-stage OUT transactions are performed similarly to the bulk OUT transactions explained in Bulk and Control OUT/SETUP Transactions in DMA Mode(p. 208) . Data- or Status-stage IN transactions are performed similarly to the bulk IN transactions explained in Bulk and Control IN Transactions in DMA Mode (p. 212) . For all three stages, the application is expected to set the USB_HC1_CHAR.EPTYPE field to Control. During the Setup stage, the application is expected to set the USB_HC1_TSIZ.PID field to SETUP.

15.4.3.6.7 Bulk and Control OUT/SETUP Transactions in DMA Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the connected device, it must initialize a channel as described in Channel Initialization (p. 199) .

This section discusses the following topics:

- Overview (p. 208)
- Normal Bulk and Control OUT/SETUP Operations (p. 208)
- NAK Handling with DMA (p. 208)
- Handling Interrupts (p. 210)

15.4.3.6.7.1 Overview

- The application is attempting to send two maximum-packet-size packets (transfer size = 1,024 bytes).
- The Non-periodic Transmit FIFO can hold two packets (128 bytes for FS).
- The Non-periodic Request Queue depth = 4.

15.4.3.6.7.2 Normal Bulk and Control OUT/SETUP Operations

The sequence of operations in Figure 15.9 (p. 205) is as follows:

1. Initialize and enable channel 1 as explained in Channel Initialization (p. 199) .
2. The host starts fetching the first packet as soon as the channel is enabled. For DMA mode, the host uses the programmed DMA address to fetch the packet.
3. After fetching the last DWORD of the second (last) packet, the host masks channel 1 internally for further arbitration.
4. The host generates a CHHLTD interrupt as soon as the last packet is sent.
5. In response to the CHHLTD interrupt, de-allocate the channel for other transfers.

The channel-specific interrupt service routine for bulk and control OUT/SETUP transactions in DMA mode is shown in Handling Interrupts (p. 210) .

15.4.3.6.7.3 NAK Handling with DMA

1. The Host sends a Bulk OUT Transaction.

2. The Device responds with NAK.
3. If the application has unmasked NAK, the core generates the corresponding interrupt(s) to the application.

   The application is not required to service these interrupts, since the core takes care of rewinding of buffer pointers and re-initializing the Channel without application intervention.
4. When the Device returns an ACK, the core continues with the transfer.

Optionally, the application can utilize these interrupts. If utilized by the application:

- The NAK interrupt is masked by the application.
- The core does not generate a separate interrupt when NAK is received by the Host functionality.

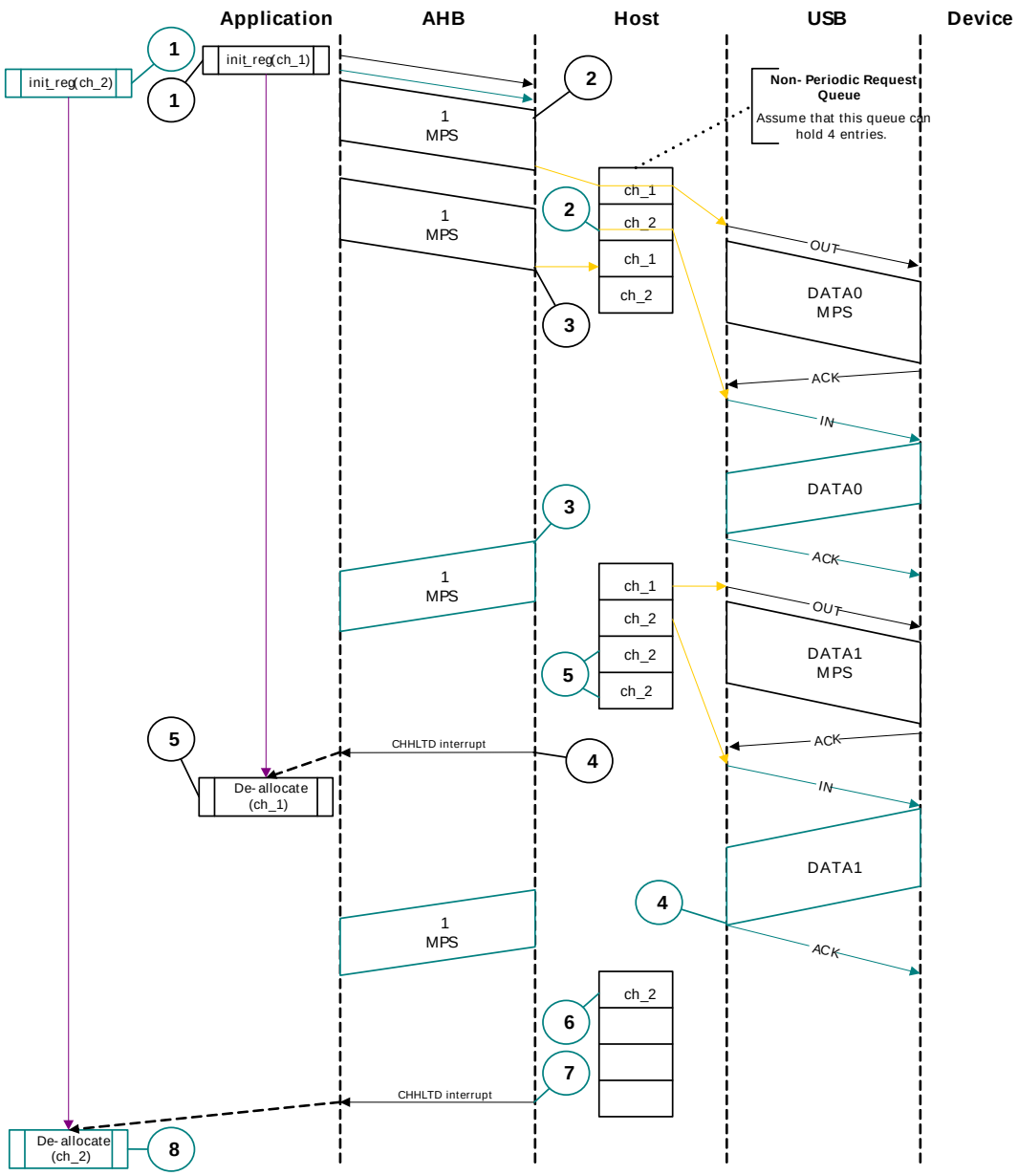
Application Programming Flow

1. The application programs a channel to do a bulk transfer for a particular data size in each transaction.
   - Packet Data size can be up to 512 KBytes
   - Zero-length data must be programmed as a separate transaction.
2. Program the transfer size register with:
   - Transfer size
   - Packet Count
3. Program the DMA address.
4. Program the USB_HCx_CHAR to enable the channel.
5. The Interrupt handling by the application is as depicted in the flow diagram.

Note

The NAK interrupts are still generated internally. The application can mask off these interrupts from reaching it. The application can use these interrupts optionally.

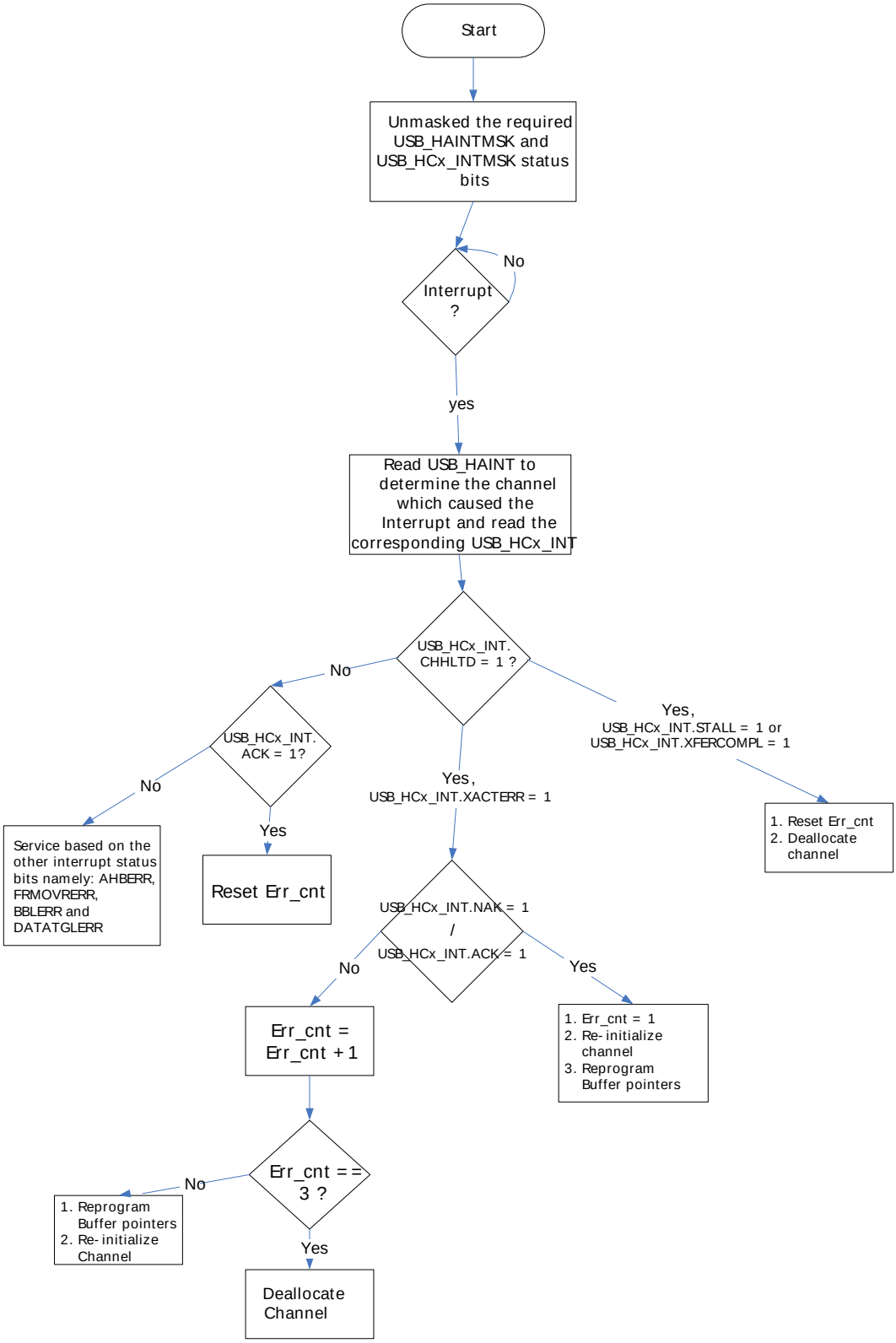
Figure 15.10. Normal Bulk/Control OUT/SETUP and Bulk/Control IN Transactions in DMA Mode



15.4.3.6.7.4 Handling Interrupts

The channel-specific interrupt service routine for bulk and control OUT/SETUP transactions in DMA mode is shown in the following code samples.

Figure 15.11. Interrupt Service Routine for Bulk/Control OUT Transaction in DMA Mode



In Figure 15.11 (p. 211) that the Interrupt Service Routine is not required to handle NAK responses. This is the difference of proposed flow with respect to current flow. Similar flow is applicable for Control flow also.

The NAK status bits in USB_HCx_INT registers are updated. The application can unmask these interrupts when it requires the core to generate an interrupt for NAK. The NAK status is updated because during Xact_err scenarios, this status provides a means for the application to determine whether the Xact_err occurred three times consecutively or there were NAK responses in between two Xact_err. This provides a mechanism for the application to reset the error counter accordingly. The application

must read the NAK/ACK along with the xact_err. If NAK/ACK is not set, the Xact_err count must be incremented otherwise application must initialize the Xact_err count to 1.

Bulk/Control OUT/SETUP

```
Unmask (CHHLTD)
if (CHHLTD)
{
    if (XFERCOMPL or STALL)
    {
        Reset Error Count (Error_count=1)
        Mask ACK
        De-allocate Channel
    }
    else if (XACTERR)
    {
        if (NAK/ACK)
        {
            Error_count = 1
            Re-initialize Channel
            Rewind Buffer Pointers
        }
        else
        {
            Error_count = Error_count + 1
            if (Error_count == 3)
            {
                De allocate channel
            }
            else
            {
                Re-initialize Channel
                Rewind Buffer Pointers
            }
        }
    }
}
else if (ACK)
{
    Reset Error Count (Error_count=1)
    Mask ACK
}
```

As soon as the channel is enabled, the core attempts to fetch and write data packets, in multiples of the maximum packet size, to the transmit FIFO when space is available in the transmit FIFO and the Request queue. The core stops fetching as soon as the last packet is fetched.

15.4.3.6.8 Bulk and Control IN Transactions in DMA Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the connected device, it must initialize a channel as described in Channel Initialization (p. 199) .

A typical bulk or control IN operation in DMA mode is shown in Figure 15.10 (p. 210) . See channel 2 (ch_2).

The assumptions are:

1. The application is attempting to receive two maximum-packet-size packets (transfer size = 1,024 bytes).
2. The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet (72 bytes for FS).

3. The Non-periodic Request Queue depth = 4.

15.4.3.6.8.1 Normal Bulk and Control IN Operations

The sequence of operations in Figure 15.10 (p. 210) is as follows:

1. Initialize and enable channel 2 as explained in Channel Initialization (p. 199) .
2. The host writes an IN request to the Request queue as soon as channel 2 receives the grant from the arbiter. (Arbitration is performed in a round-robin fashion, with fairness.).
3. The host starts writing the received data to the system memory as soon as the last byte is received with no errors.
4. When the last packet is received, the host sets an internal flag to remove any extra IN requests from the Request queue.
5. The host flushes the extra requests.
6. The final request to disable channel 2 is written to the Request queue. At this point, channel 2 is internally masked for further arbitration.
7. The host generates the CHHLTD interrupt as soon as the disable request comes to the top of the queue.
8. In response to the CHHLTD interrupt, de-allocate the channel for other transfers.

15.4.3.6.8.2 Handling Interrupts

The channel-specific interrupt service routine for bulk and control IN transactions in DMA mode is shown in the following flow:

Interrupt Service Routines for Bulk/Control Bulk/Control IN Transactions in DMA Mode

Bulk/Control IN

```
Unmask (CHHLTD)
if (CHHLTD)
{
    if (XFERCOMPL or STALL or BBLERR)
    {
        Reset Error Count Mask ACK De-allocate Channel
    }
    else if (XACTERR)
    {
        if (Error_count == 2)
        {
            De-allocate Channel
        }
        else
        {
            Unmask ACK
            Unmask NAK
            Unmask DATATGLERR
            Increment Error
            Count Re-initialize Channel
        }
    }
}
else if (ACK or NAK or DATATGLERR)
{
    Reset Error Count
    Mask ACK
    Mask NAK
    Mask DATATGLERR
}
```

15.4.3.6.9 Interrupt OUT Transactions in Slave Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the connected device, it must initialize a channel as described in Channel Initialization (p. 199). See Figure 15.7 (p. 203) and Figure 15.8 (p. 203) for read or write data to and from the FIFO in Slave mode.

A typical interrupt OUT operation in Slave mode is shown in Figure 15.12 (p. 215). See channel 1 (ch_1). The assumptions are:

- The application is attempting to send one packet in every frame (up to 1 maximum packet size), starting with the odd frame (transfer size = 1,024 bytes).
- The Periodic Transmit FIFO can hold one packet.
- Periodic Request Queue depth = 4.

15.4.3.6.9.1 Normal Interrupt OUT Operation

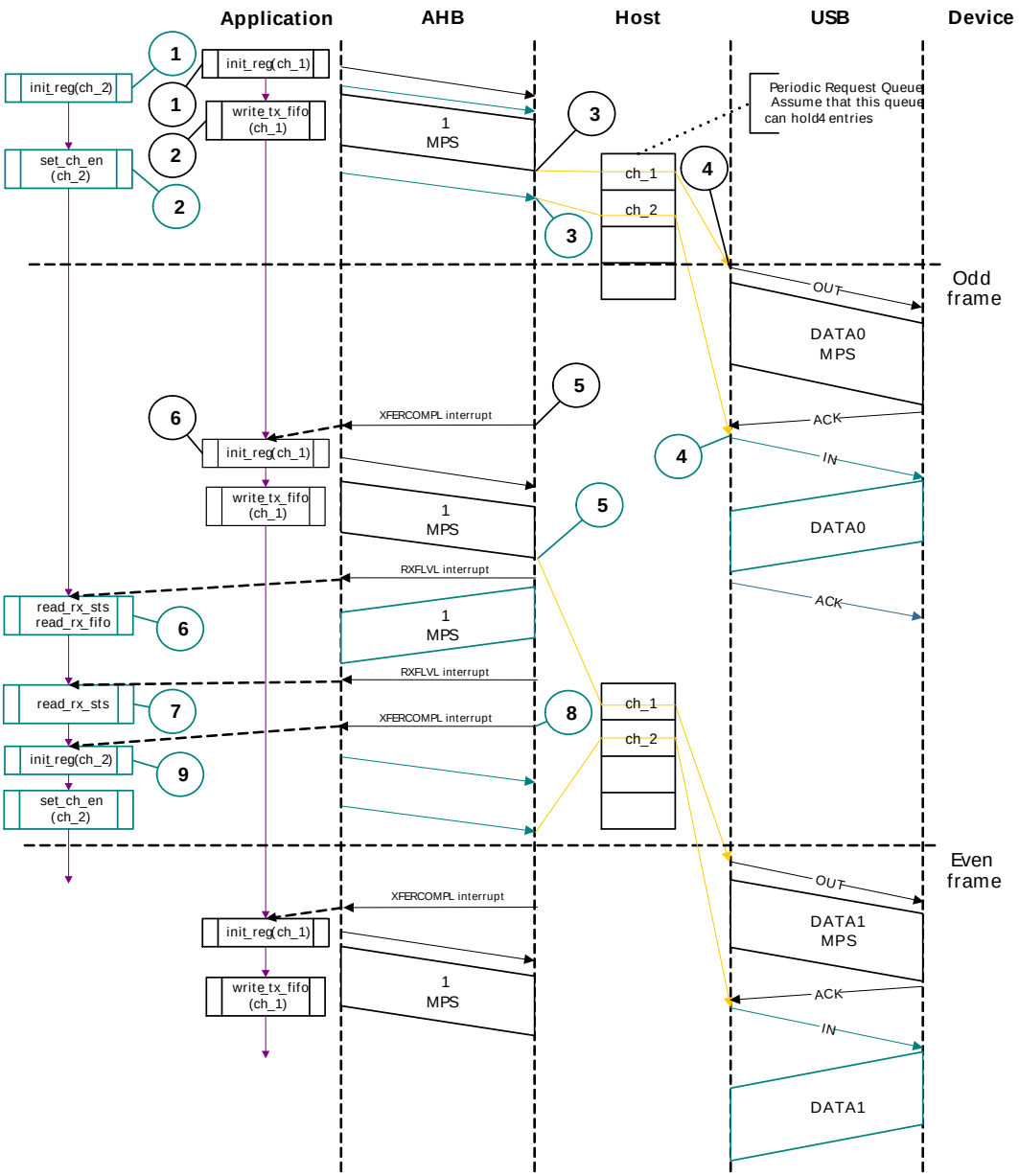
The sequence of operations in Figure 15.12 (p. 215) is as follows:

1. Initialize and enable channel 1 as explained in Channel Initialization (p. 199). The application must set the USB_HC1_CHAR.ODDFRM bit.
2. Write the first packet for channel 1. For a high-bandwidth interrupt transfer, the application must write the subsequent packets up to MC (maximum number of packets to be transmitted in the next frame times before switching to another channel).
3. Along with the last DWORD write of each packet, the host writes an entry to the Periodic Request Queue.
4. The host attempts to send an OUT token in the next (odd) frame.
5. The host generates an XFERCOMPL interrupt as soon as the last packet is transmitted successfully.
6. In response to the XFERCOMPL interrupt, reinitialize the channel for the next transfer.

15.4.3.6.9.2 Handling Interrupts

The channel-specific interrupt service routine for Interrupt OUT transactions in Slave mode is shown in the following flow:

Figure 15.12. Normal Interrupt OUT/IN Transactions in Slave Mode



Interrupt Service Routine for Interrupt OUT Transactions in Slave Mode

Interrupt OUT

```
Unmask (NAK/XACTERR/STALL/XFERCOMPL/FRMOVRUN)
if (XFERCOMPL)
{
    Reset Error Count
    Mask ACK
    De-allocate Channel
}
else if (STALL or FRMOVRUN)
{
    Mask ACK
    Unmask CHHLTD
    Disable Channel
    if (STALL)
    {
        Transfer Done = 1
    }
```

```
}
else if (NAK or XACTERR)
{
    Rewind Buffer Pointers
    Reset Error Count
    Mask ACK
    Unmask CHHLTD
    Disable Channel
}
else if (CHHLTD)
{
    Mask CHHLTD
    if (Transfer Done or (Error_count == 3))
    {
        De-allocate Channel
    }
    else
    {
        Re-initialize Channel (in next b_interval - 1 Frame)
    }
}
else if (ACK)
{
    Reset Error Count
    Mask ACK
}
```

The application is expected to write the data packets into the transmit FIFO when the space is available in the transmit FIFO and the Request queue up to the count specified in the MC field before switching to another channel. The application uses the USB_GINTSTS.NPTXFEMP interrupt to find the transmit FIFO space.

15.4.3.6.10 Interrupt IN Transactions in Slave Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the connected device, it must initialize a channel as described in Channel Initialization (p. 199). See Transmit FIFO Write Task in Slave Mode and Receive FIFO Read Task in Slave Mode for read or write data to and from the FIFO in Slave mode.

A typical interrupt-IN operation in Slave mode is shown in Figure 15.12 (p. 215). See channel 2 (ch_2). The assumptions are:

1. The application is attempting to receive one packet (up to 1 maximum packet size) in every frame, starting with odd. (transfer size = 1,024 bytes).
2. The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet (1,031 bytes for FS).
3. Periodic Request Queue depth = 4.

15.4.3.6.10.1 Normal Interrupt IN Operation

The sequence of operations in Figure 15.12 (p. 215) (channel 2) is as follows:

1. Initialize channel 2 as explained in Channel Initialization (p. 199). The application must set the USB_HC2_CHAR.ODDFRM bit.
2. Set the USB_HC2_CHAR.CHENA bit to write an IN request to the Periodic Request Queue. For a high-bandwidth interrupt transfer, the application must write the USB_HC2_CHAR register MC (maximum number of expected packets in the next frame) times before switching to another channel.
3. The host writes an IN request to the Periodic Request Queue for each USB_HC2_CHAR register write with a CHENA bit set.
4. The host attempts to send an IN token in the next (odd) frame.

5. As soon as the IN packet is received and written to the receive FIFO, the host generates an RXFLVL interrupt.
6. In response to the RXFLVL interrupt, read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. The application must mask the RXFLVL interrupt before reading the receive FIFO, and unmask after reading the entire packet.
7. The core generates the RXFLVL interrupt for the transfer completion status entry in the receive FIFO. The application must read and ignore the receive packet status when the receive packet status is not an IN data packet (USB_GRXSTSR.PKTSTS != 0b0010).
8. The core generates an XFERCOMPL interrupt as soon as the receive packet status is read.
9. In response to the XFERCOMPL interrupt, read the USB_HC2_TSIZ.PKTCNT field. If USB_HC2_TSIZ.PKTCNT != 0, disable the channel (as explained in Halting a Channel (p. 200)) before re-initializing the channel for the next transfer, if any). If USB_HC2_TSIZ.PKTCNT == 0, reinitialize the channel for the next transfer. This time, the application must reset the USB_HC2_CHAR.ODDFRM bit.

15.4.3.6.10.2 Handling Interrupts

The channel-specific interrupt service routine for an interrupt IN transaction in Slave mode is a follows.

Interrupt IN

```
Unmask (NAK/XACTERR/XFERCOMPL/BBLERR/STALL/FRMOVRUN/DATATGLERR)
if (XFERCOMPL)
{
    Reset Error Count
    Mask ACK
    if (USB_HCx_TSIZ.PKTCNT == 0)
    {
        De-allocate Channel
    }
    else
    {
        Transfer Done = 1
        Unmask CHHLTD
        Disable Channel
    }
}
else if (STALL or FRMOVRUN or NAK or DATATGLERR or BBLERR)
{
    Mask ACK
    Unmask CHHLTD
    Disable Channel
    if (STALL or BBLERR)
    {
        Reset Error Count
        Transfer Done = 1
    }
    else if (!FRMOVRUN)
    {
        Reset Error Count
    }
}
else if (XACTERR)
{
    Increment Error Count
    Unmask ACK
    Unmask CHHLTD
    Disable Channel
}
else if (CHHLTD)
{
    Mask CHHLTD
```

```
    if (Transfer Done or (Error_count == 3))
    {
        De-allocate Channel
    }
    else
    {
        Re-initialize Channel (in next b_interval - 1 Frame)
    }
}
else if (ACK)
{
    Reset Error Count
    Mask ACK
}
```

The application is expected to write the requests for the same channel when the Request queue space is available up to the count specified in the MC field before switching to another channel (if any).

15.4.3.6.11 Interrupt OUT Transactions in DMA Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the connected device, it must initialize a channel as described in Channel Initialization (p. 199) .

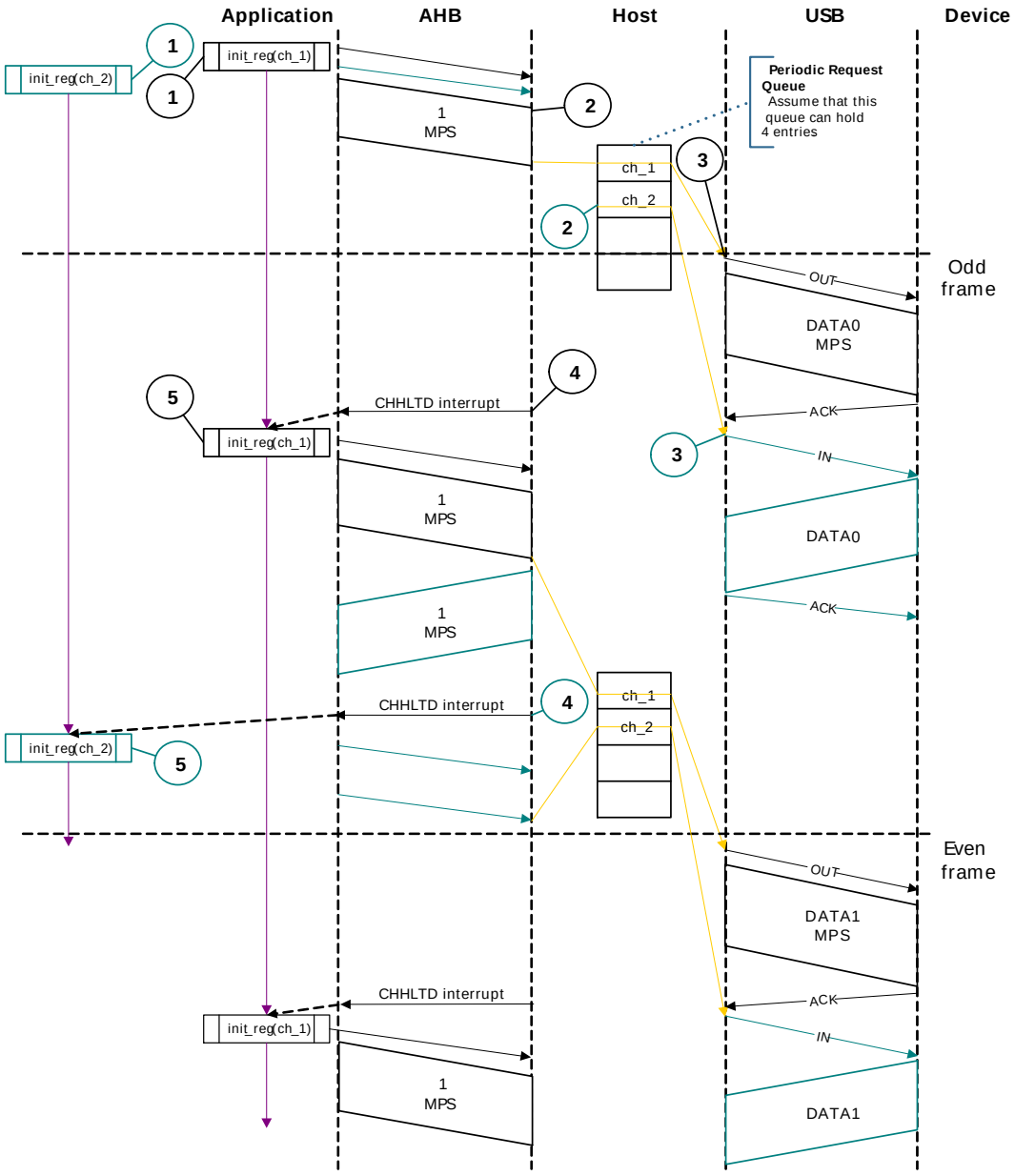
A typical interrupt OUT operation in DMA mode is shown in Figure 15.13 (p. 219) . See channel 1 (ch_1). The assumptions are:

- The application is attempting to transmit one packet in every frame (up to 1 maximum packet size of 1,024 bytes).
- The Periodic Transmit FIFO can hold one packet (1 KB for FS).
- Periodic Request Queue depth = 4.

15.4.3.6.11.1 Normal Interrupt OUT Operation

1. Initialize and enable channel 1 as explained in Channel Initialization (p. 199) .
2. The host starts fetching the first packet as soon the channel is enabled and writes the OUT request along with the last DWORD fetch. In high-bandwidth transfers, the host continues fetching the next packet (up to the value specified in the MC field) before switching to the next channel.
3. The host attempts to send the OUT token in the beginning of the next odd frame.
4. After successfully transmitting the packet, the host generates a CHHLTD interrupt.
5. In response to the CHHLTD interrupt, reinitialize the channel for the next transfer.

Figure 15.13. Normal Interrupt OUT/IN Transactions in DMA Mode



15.4.3.6.11.2 Handling Interrupts

The following code sample shows the channel-specific ISR for an interrupt OUT transaction in DMA mode.

Interrupt OUT

```
Unmask (CHHLTD)
if (CHHLTD)
{
    if (XFERCOMPL)
    {
        Reset Error Count
        Mask ACK
        if (Transfer Done)
        {
            De-allocate Channel
        }
        else
        {
```

```
                Re-initialize Channel (in next b_interval - 1 Frame)
            }
        }
        else if (STALL)
        {
            Transfer Done = 1
            Reset Error Count
            Mask ACK
            De-allocate Channel
        }
        else if (NAK or FRMOVRUN)
        {
            Mask ACK
            Rewind Buffer Pointers
            Re-initialize Channel (in next b_interval - 1 Frame)
            if (NAK)
            {
                Reset Error Count
            }
        }
        else if (XACTERR)
        {
            if (Error_count == 2)
            {
                De-allocate Channel
            }
            else
            {
                Increment Error Count
                Rewind Buffer Pointers
                Unmask ACK
                Re-initialize Channel (in next b_interval - 1 Frame)
            }
        }
    }
    else if (ACK)
    {
        Reset Error Count
        Mask ACK
    }
```

As soon as the channel is enabled, the core attempts to fetch and write data packets, in maximum packet size multiples, to the transmit FIFO when the space is available in the transmit FIFO and the Request queue. The core stops fetching as soon as the last packet is fetched (the number of packets is determined by the MC field of the USB_HCx_CHAR register).

15.4.3.6.12 Interrupt IN Transactions in DMA Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the connected device, it must initialize a channel as described in Channel Initialization (p. 199) .

A typical interrupt IN operation in DMA mode is shown in Figure 15.13 (p. 219). See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one packet in every frame (up to 1 maximum packet size of 1,024 bytes).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet (1,032 bytes for FS).
- Periodic Request Queue depth = 4.

15.4.3.6.12.1 Normal Interrupt IN Operation

The sequence of operations in Figure 15.13 (p. 219) (channel 2) is as follows:

1. Initialize and enable channel 2 as explained in Channel Initialization (p. 199) .
2. The host writes an IN request to the Request queue as soon as the channel 2 gets the grant from the arbiter (round-robin with fairness). In high-bandwidth transfers, the host writes consecutive writes up to MC times.
3. The host attempts to send an IN token at the beginning of the next (odd) frame.
4. As soon the packet is received and written to the receive FIFO, the host generates a CHHLTD interrupt.
5. In response to the CHHLTD interrupt, reinitialize the channel for the next transfer.

15.4.3.6.12.2 Handling Interrupts

The channel-specific interrupt service routine for Interrupt IN transactions in DMA mode is as follows.

Interrupt Service Routine for Interrupt IN Transactions in DMA Mode

```
Unmask (CHHLTD)
if (CHHLTD)
{
    if (XFERCOMPL)
    {
        Reset Error Count
        Mask ACK
        if (Transfer Done)
        {
            De-allocate Channel
        }
        else
        {
            Re-initialize Channel (in next b_interval - 1 Frame)
        }
    }
    else if (STALL or BBLERR)
    {
        Reset Error Count
        Mask ACK
        De-allocate Channel
    }
    else if (NAK or DATATGLERR or FRMOVRUN)
    {
        Mask ACK
        Re-initialize Channel (in next b_interval - 1 Frame)
        if (DATATGLERR or NAK)
        {
            Reset Error Count
        }
    }
    else if (XACTERR)
    {
        if (Error_count == 2)
        {
            De-allocate Channel
        }
        else
        {
            Increment Error Count
            Unmask ACK
            Re-initialize Channel (in next b_interval - 1 Frame)
        }
    }
}
else if (ACK)
{
    Reset Error Count
    Mask ACK
```

```
}
```

As soon as the channel is enabled, the core attempts to write the requests into the Request queue when the space is available up to the count specified in the MC field.

15.4.3.6.13 Isochronous OUT Transactions in Slave Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193) . Before it can communicate with the connected device, it must initialize a channel as described in Channel Initialization (p. 199) . See TFigure 15.7 (p. 203) and Figure 15.8 (p. 203) for read or write data to and from the FIFO in Slave mode.

A typical isochronous OUT operation in Slave mode is shown in Figure 15.14 (p. 223) . See channel 1 (ch_1). The assumptions are:

- The application is attempting to send one packet every frame (up to 1 maximum packet size), starting with an odd frame. (transfer size = 1,024 bytes).
- The Periodic Transmit FIFO can hold one packet (1 KB).
- Periodic Request Queue depth = 4.

15.4.3.6.13.1 Normal Isochronous OUT Operation

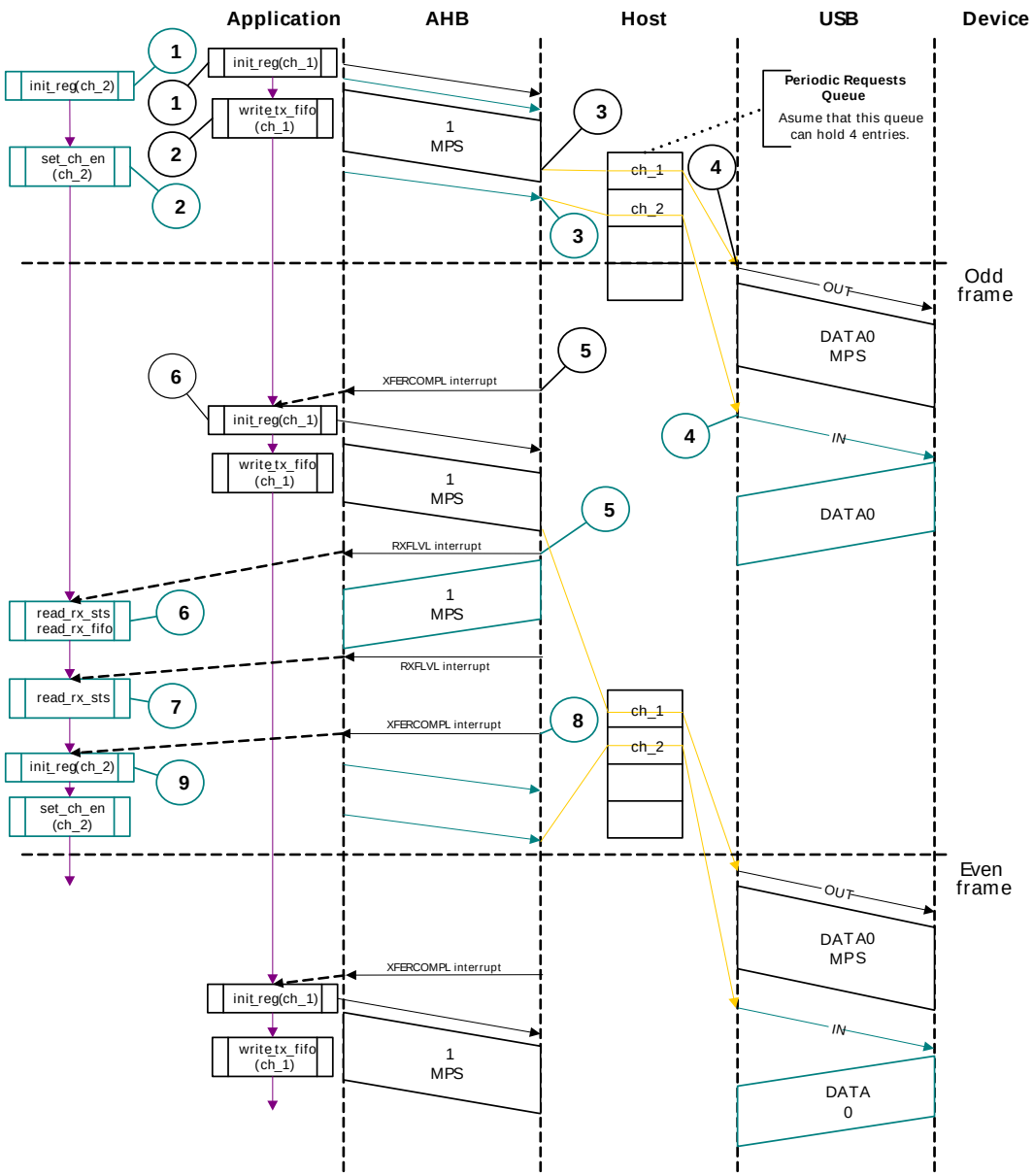
The sequence of operations in Figure 15.14 (p. 223) (channel 1) is as follows:

1. Initialize and enable channel 1 as explained in Channel Initialization (p. 199) . The application must set the USB_HC1_CHAR.ODDFRM bit.
2. Write the first packet for channel 1. For a high-bandwidth isochronous transfer, the application must write the subsequent packets up to MC (maximum number of packets to be transmitted in the next frame) times before switching to another channel.
3. Along with the last DWORD write of each packet, the host writes an entry to the Periodic Request Queue.
4. The host attempts to send the OUT token in the next frame (odd).
5. The host generates the XFERCOMPL interrupt as soon as the last packet is transmitted successfully.
6. In response to the XFERCOMPL interrupt, reinitialize the channel for the next transfer.

15.4.3.6.13.2 Handling Interrupts

The channel-specific interrupt service routine for isochronous OUT transactions in Slave mode is shown in the following flow:

Figure 15.14. Normal Isochronous OUT/IN Transactions in Slave Mode



Interrupt Service Routine for Isochronous OUT Transactions in Slave Mode

Isochronous OUT

```
Unmask (FRMOVRUN/XFERCOMPL)
if (XFERCOMPL)
{
    De-allocate Channel
}
else if (FRMOVRUN)
{
    Unmask CHHLTD
    Disable Channel
}
else if (CHHLTD)
{
    Mask CHHLTD
    De-allocate Channel
}
```

15.4.3.6.14 Isochronous IN Transactions in Slave Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the connected device, it must initialize a channel as described in Channel Initialization (p. 199). See Figure 15.7 (p. 203) and Figure 15.8 (p. 203) for read or write data to and from the FIFO in Slave mode.

A typical isochronous IN operation in Slave mode is shown in Figure 15.14 (p. 223). See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one packet (up to 1 maximum packet size) in every frame starting with the next odd frame. (transfer size = 1,024 bytes).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet (1,031 bytes for FS).
- Periodic Request Queue depth = 4.

15.4.3.6.14.1 Normal Isochronous IN Operation

The sequence of operations in Figure 15.14 (p. 223) (channel 2) is as follows:

1. Initialize channel 2 as explained in Channel Initialization (p. 199). The application must set the USB_HC2_CHAR.ODDFRM bit.
2. Set the USB_HC2_CHAR.CHENA bit to write an IN request to the Periodic Request Queue. For a high-bandwidth isochronous transfer, the application must write the USB_HC2_CHAR register MC (maximum number of expected packets in the next frame) times before switching to another channel.
3. The host writes an IN request to the Periodic Request Queue for each USB_HC2_CHAR register write with the CHENA bit set.
4. The host attempts to send an IN token in the next odd frame.
5. As soon as the IN packet is received and written to the receive FIFO, the host generates an RXFLVL interrupt.
6. In response to the RXFLVL interrupt, read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. The application must mask the RXFLVL interrupt before reading the receive FIFO, and unmask it after reading the entire packet.
7. The core generates an RXFLVL interrupt for the transfer completion status entry in the receive FIFO. This time, the application must read and ignore the receive packet status when the receive packet status is not an IN data packet (USB_GRXSTSR.PKTSTS != 0b0010).
8. The core generates an XFERCOMPL interrupt as soon as the receive packet status is read.
9. In response to the XFERCOMPL interrupt, read the USB_HC2_TSIZ.PKTCNT field. If USB_HC2_TSIZ.PKTCNT != 0, disable the channel (as explained in Halting a Channel (p. 200)) before re-initializing the channel for the next transfer, if any. If USB_HC2_TSIZ.PKTCNT == 0, reinitialize the channel for the next transfer. This time, the application must reset the USB_HC2_CHAR.ODDFRM bit.

15.4.3.6.14.2 Handling Interrupts

The channel-specific interrupt service routine for an isochronous IN transaction in Slave mode is as follows.

Isochronous IN

```
Unmask (XACTERR/XFERCOMPL/FRMOVRUN/BBLERR)
if (XFERCOMPL or FRMOVRUN)
{
    if (XFERCOMPL and (USB_HCx_TSIZ.PKTCNT == 0))
```

```
    {
        Reset Error Count
        De-allocate Channel
    }
    else
    {
        Unmask CHHLTD
        Disable Channel
    }
}
else if (XACTERR or BBLERR)
{
    Increment Error Count
    Unmask CHHLTD
    Disable Channel
}
else if (CHHLTD)
{
    Mask CHHLTD
    if (Transfer Done or (Error_count == 3))
    {
        De-allocate Channel
    }
    else
    {
        Re-initialize Channel
    }
}
```

15.4.3.6.15 Isochronous OUT Transactions in DMA Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the connected device, it must initialize a channel as described in Channel Initialization (p. 199) .

A typical isochronous OUT operation in DMA mode is shown in Figure 15.15 (p. 226) . See channel 1 (ch_1). The assumptions are:

- The application is attempting to transmit one packet every frame (up to 1 maximum packet size of 1,024 bytes).
- The Periodic Transmit FIFO can hold one packet (1 KB).
- Periodic Request Queue depth = 4.

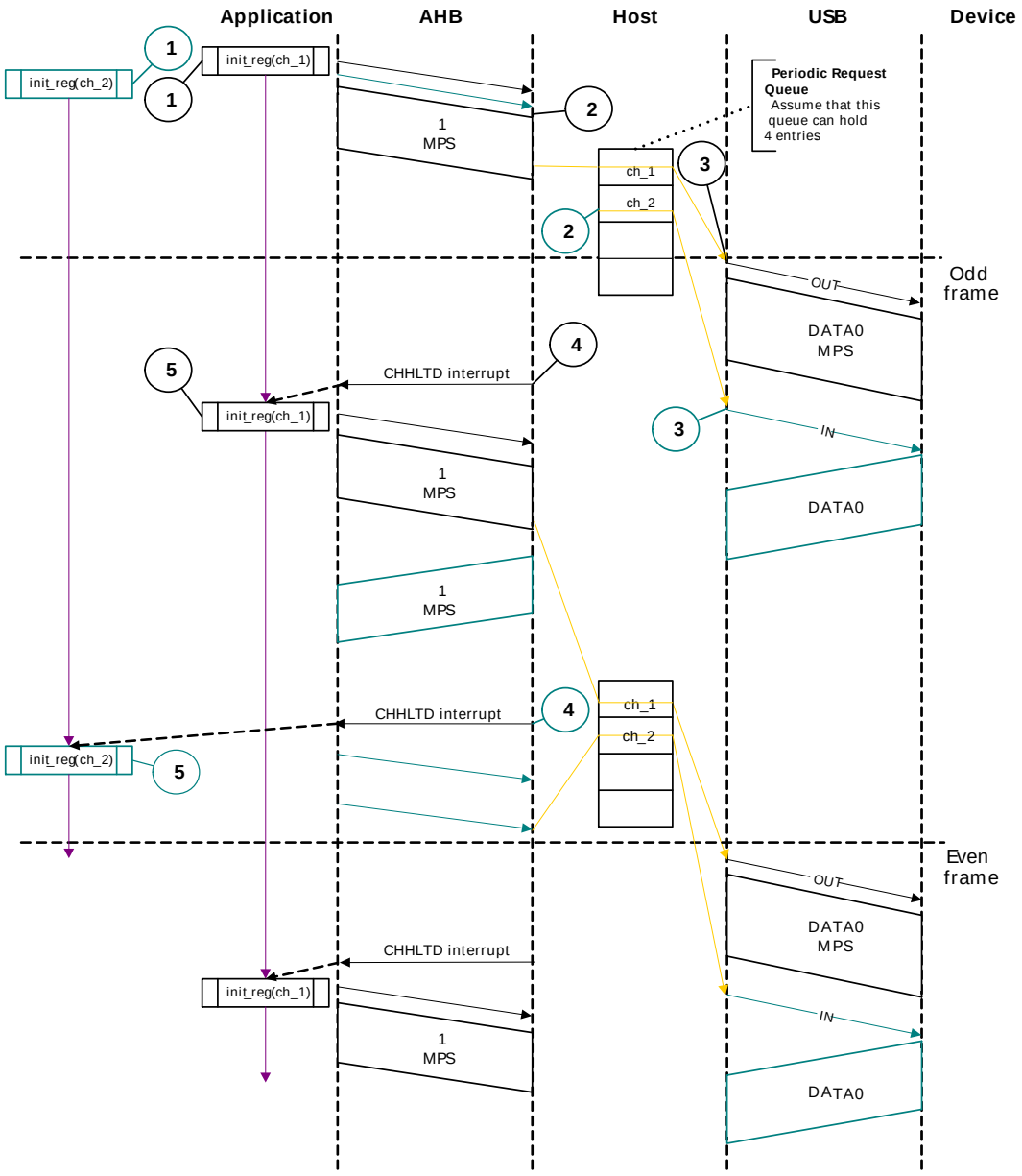
15.4.3.6.15.1 Normal Isochronous OUT Operation

1. Initialize and enable channel 1 as explained in Channel Initialization (p. 199) .
2. The host starts fetching the first packet as soon as the channel is enabled, and writes the OUT request along with the last DWORD fetch. In high-bandwidth transfers, the host continues fetching the next packet (up to the value specified in the MC field) before switching to the next channel.
3. The host attempts to send an OUT token in the beginning of the next (odd) frame.
4. After successfully transmitting the packet, the host generates a CHHLTD interrupt.
5. In response to the CHHLTD interrupt, reinitialize the channel for the next transfer.

15.4.3.6.15.2 Handling Interrupts

The channel-specific interrupt service routine for Isochronous OUT transactions in DMA mode is shown in the following flow:

Figure 15.15. Normal Isochronous OUT/IN Transactions in DMA Mode



Interrupt Service Routine for Isochronous OUT Transactions in DMA Mode

Isochronous OUT

```
Unmask (CHHLTD)
if (CHHLTD)
{
    if (XFERCOMPL or FRMOVRUN)
    {
        De-allocate Channel
    }
}
```

15.4.3.6.16 Isochronous IN Transactions in DMA Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the connected device, it must initialize a channel as described in Channel Initialization (p. 199) .

A typical isochronous IN operation in DMA mode is shown in Figure 15.15 (p. 226) . See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one packet in every frame (up to 1 maximum packet size of 1,024 bytes).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDS per packet (1,031 bytes).
- Periodic Request Queue depth = 4.

15.4.3.6.16.1 Normal Isochronous IN Operation

The sequence of operations in Figure 15.15 (p. 226) (channel 2) is as follows:

1. Initialize and enable channel 2 as explained in Channel Initialization (p. 199) .
2. The host writes an IN request to the Request queue as soon as the channel 2 gets the grant from the arbiter (round-robin with fairness). In high-bandwidth transfers, the host performs consecutive writes up to MC times.
3. The host attempts to send an IN token at the beginning of the next (odd) frame.
4. As soon the packet is received and written to the receive FIFO, the host generates a CHHLTD interrupt.
5. In response to the CHHLTD interrupt, reinitialize the channel for the next transfer.

15.4.3.6.16.2 Handling Interrupts

The channel-specific interrupt service routine for an isochronous IN transaction in DMA mode is as follows.

Isochronous IN

```
Unmask (CHHLTD)
if (CHHLTD)
{
    if (XFERCOMPL or FRMOVRUN)
    {
        if (XFERCOMPL and (USB_HCx_TSIZ.PKTCNT == 0))
        {
            Reset Error Count
            De-allocate Channel
        }
        else
        {
            De-allocate Channel
        }
    }
    else if (XACTERR or BBLERR)
    {
        if (Error_count == 2)
        {
            De-allocate Channel
        }
        else
        {
            Increment Error Count
            Re-enable Channel (in next b_interval - 1 Frame)
        }
    }
}
```

15.4.4 Device Programming Model

Before you program the Device, be sure to read Overview: Programming the Core (p. 193) and Modes of operation (p. 196)

15.4.4.1 Endpoint Initialization

This section addresses the following topics:

- Initialization on USB Reset (p. 228)
- Initialization on Enumeration Completion (p. 228)
- Initialization on SetAddress Command (p. 229)
- Initialization on SetConfiguration/SetInterface Command (p. 229)
- Endpoint Activation (p. 229)
- Endpoint Deactivation (p. 229)
- Device DMA/Slave Mode Initialization (p. 230)

15.4.4.1.1 Initialization on USB Reset

1. Set the NAK bit for all OUT endpoints
   - USB_DOEPx_CTL.SNAK = 1 (for all OUT endpoints)
2. Unmask the following interrupt bits:
   - USB_USB_DAINTMSK.INEP0 = 1 (control 0 IN endpoint)
   - USB_USB_DAINTMSK.OUTEP0 = 1 (control 0 OUT endpoint)
   - USB_DOEPMSK.SETUP = 1
   - USB_DOEPMSK.XFERCOMPL = 1
   - USB_DIEPMSK.XFERCOMPL = 1
   - USB_DIEPMSK.TIMEOUTMSK = 1
3. To transmit or receive data, the device must initialize more registers as specified in Device DMA/ Slave Mode Initialization (p. 230) .
4. Set up the Data FIFO RAM for each of the FIFOs
   - Program the USB_GRXFSIZ Register, to be able to receive control OUT data and setup data. At a minimum, this must be equal to 1 max packet size of control endpoint 0 + 2 DWORDs (for the status of the control OUT data packet) + 10 DWORDs (for setup packets).
   - Program the Device IN Endpoint Transmit FIFO size register (depending on the FIFO number chosen), to be able to transmit control IN data. At a minimum, this must be equal to 1 max packet size of control endpoint 0.
5. Program the following fields in the endpoint-specific registers for control OUT endpoint 0 to receive a SETUP packet
   - USB_DOEP0TSIZ.SUPCNT = 3 (to receive up to 3 back-to-back SETUP packets)
   - In DMA mode, USB_DOEP0DMAADDR register with a memory address to store any SETUP packets received

At this point, all initialization required to receive SETUP packets is done, except for enabling control OUT endpoint 0 in DMA mode.

15.4.4.1.2 Initialization on Enumeration Completion

1. On the Enumeration Done interrupt (USB_GINTSTS.ENUMDONE, read the USB_DSTS register to determine the enumeration speed.
2. Program the USB_DIEP0CTL.MPS field to set the maximum packet size. This step configures control endpoint 0. The maximum packet size for a control endpoint depends on the enumeration speed.
3. In DMA mode, program the USB_DOEP0CTL register to enable control OUT endpoint 0, to receive a SETUP packet.
   - USB_DOEP0CTL.EPENA = 1

At this point, the device is ready to receive SOF packets and is configured to perform control transfers on control endpoint 0.

15.4.4.1.3 Initialization on SetAddress Command

This section describes what the application must do when it receives a SetAddress command in a SETUP packet.

1. Program the USB_DCFG register with the device address received in the SetAddress command
2. Program the core to send out a status IN packet.

15.4.4.1.4 Initialization on SetConfiguration/SetInterface Command

This section describes what the application must do when it receives a SetConfiguration or SetInterface command in a SETUP packet.

1. When a SetConfiguration command is received, the application must program the endpoint registers to configure them with the characteristics of the valid endpoints in the new configuration.
2. When a SetInterface command is received, the application must program the endpoint registers of the endpoints affected by this command.
3. Some endpoints that were active in the prior configuration or alternate setting are not valid in the new configuration or alternate setting. These invalid endpoints must be deactivated.
4. For details on a particular endpoint's activation or deactivation, see Endpoint Activation (p. 229) and Endpoint Deactivation (p. 229) .
5. Unmask the interrupt for each active endpoint and mask the interrupts for all inactive endpoints in the USB_USB_DAINTMSK register.
6. Set up the Data FIFO RAM for each FIFO. See Data FIFO RAM Allocation (p. 274) for more detail.
7. After all required endpoints are configured, the application must program the core to send a status IN packet.

At this point, the device core is configured to receive and transmit any type of data packet.

15.4.4.1.5 Endpoint Activation

This section describes the steps required to activate a device endpoint or to configure an existing device endpoint to a new type.

1. Program the characteristics of the required endpoint into the following fields of the USB_DIEPx_CTL register (for IN or bidirectional endpoints) or the USB_DOEPx_CTL register (for OUT or bidirectional endpoints).
   - Maximum Packet Size
   - USB Active Endpoint = 1
   - Endpoint Start Data Toggle (for interrupt and bulk endpoints)
   - Endpoint Type
   - TxFIFO Number
2. Once the endpoint is activated, the core starts decoding the tokens addressed to that endpoint and sends out a valid handshake for each valid token received for the endpoint.

15.4.4.1.6 Endpoint Deactivation

This section describes the steps required to deactivate an existing endpoint.

1. In the endpoint to be deactivated, clear the USB Active Endpoint bit in the USB_DIEPx_CTL register (for IN or bidirectional endpoints) or the USB_DOEPx_CTL register (for OUT or bidirectional endpoints).
2. Once the endpoint is deactivated, the core ignores tokens addressed to that endpoint, resulting in a timeout on the USB.

15.4.4.1.7 Device DMA/Slave Mode Initialization

The application must meed the following conditions to set up the device core to handle traffic.

- In Slave mode, USB_GINTMSK.NPTXFEMPMSK, and USB_GINTMSK.RXFLVLMSK must be unset.
- In DMA mode, the aforementioned interrupts must be masked.

15.4.4.1.8 Transfer Stop Process

When the core is operating as a device, use the following programing sequence if you want to stop any transfers (because of an interrupt from the host, typically a reset).

15.4.4.1.8.1 Transfer Stop Programming Flow for IN Endpoints

Sequence of operations:

1. Disable the IN endpoint by programming USB_DIEP0CTL/USB_DIEPx_CTL.EPDIS = 1.
2. Wait for the USB_DIEPx_INT.EPDISBLD interrupt, which indicates that the IN endpoint is completely disabled. When the EPDISBLD interrupt is asserted, the core clears the following bits:
   - USB_DIEP0CTL/USB_DIEPx_CTL.EPDIS = 0
   - USB_DIEP0CTL/USB_DIEPx_CTL.EPENA = 0
3. Flush the TX FIFO by programming the following bits:
   - USB_GRSTCTL.TXFFLSH = 1
   - USB_GRSTCTL.TXFNUM = FIFO number specific to endpoint
4. The application can start polling till USB_GRSTCTL.TXFFLSH is cleared. When this bit is cleared, it ensures that there is no data left in the TX FIFO.

15.4.4.1.8.2 Transfer Stop Programming Flow for OUT Endpoints

Sequence of operations:

1. Enable all OUT endpoints by setting USB_DOEP0CTL/USB_DOEPx_CTL.EPENA = 1.
2. Before disabling any OUT endpoint, the application must enable Global OUT NAK mode in the core, according to the instructions in Setting the Global OUT NAK (p. 238) . This ensures that data in the RX FIFO is sent to the application successfully. Set USB_DCTL.USB_DCTL.SGOUTNAK = 1.
3. Wait for the USB_GINTSTS.GOUTNAKEFF interrupt.
4. Disable all active OUT endpoints by programming the following register bits:
   - USB_DOEP0CTL/USB_DOEPx_CTL.EPENA = 1
   - USB_DOEP0CTL/USB_DOEPx_CTL.EPDIS = 1
   - USB_DOEP0CTL/USB_DOEPx_CTL.SNAK = 1
5. Wait for the USB_DOEP0INT/USB_DOEPx_INT.EPDISBLD interrupt for each OUT endpoint programmed in the previous step. The USB_DOEP0INT/USB_DOEPx_INT.EPDISBLD interrupt indicates that the corresponding OUT endpoint is completely disabled. When the EPDISBLD interrupt is asserted, the core clears the following bits:
   - USB_DOEP0CTL/USB_DOEPx_CTL.EPENA = 0
   - USB_DOEP0CTL/USB_DOEPx_CTL.EPDIS = 0

Note

The application must not flush the Rx FIFO, as the Global OUT NAK effective interrupt earlier ensures that there is no data left in the Rx FIFO.

15.4.4.2 Device Programming Operations

Table 15.2 (p. 231) provides links to the programming sequence for different USB transaction types.

Table 15.2.

Device Mode	IN	SETUP	OUT
Control			
Slave	Generic Non-Periodic (Bulk and Control) IN Data Transfers Without Thresholding in DMA and Slave Mode (p. 255)	OUT Data Transfers in Slave and DMA Modes (p. 232)	Generic Non-Isochronous OUT Data Transfers Without Thresholding in DMA and Slave Modes (p. 240)
DMA	Generic Non-Periodic (Bulk and Control) IN Data Transfers Without Thresholding in DMA and Slave Mode (p. 255)	OUT Data Transfers in Slave and DMA Modes (p. 232)	Generic Non-Isochronous OUT Data Transfers Without Thresholding in DMA and Slave Modes (p. 240)
Bulk			
Slave	Generic Non-Periodic (Bulk and Control) IN Data Transfers Without Thresholding in DMA and Slave Mode (p. 255)		Generic Non-Isochronous OUT Data Transfers Without Thresholding in DMA and Slave Modes (p. 240)
DMA	Generic Non-Periodic (Bulk and Control) IN Data Transfers Without Thresholding in DMA and Slave Mode (p. 255)		Generic Non-Isochronous OUT Data Transfers Without Thresholding in DMA and Slave Modes (p. 240)
Interrupt			
Slave	Generic Periodic IN (Interrupt and Isochronous) Data Transfers Without Thresholding (p. 260) and Generic Periodic IN Data Transfers Without Thresholding Using the Periodic Transfer Interrupt Feature (p. 262)		Generic Non-Isochronous OUT Data Transfers Without Thresholding in DMA and Slave Modes (p. 240) and Generic Interrupt OUT Data Transfers Without Thresholding Using Periodic Transfer Interrupt Feature (p. 244)
DMA	Generic Periodic IN (Interrupt and Isochronous) Data Transfers Without Thresholding (p. 260) and Generic Periodic IN Data Transfers Without Thresholding Using the Periodic Transfer Interrupt Feature (p. 262)		Generic Non-Isochronous OUT Data Transfers Without Thresholding in DMA and Slave Modes (p. 240) and Generic Interrupt OUT Data Transfers Without Thresholding Using Periodic Transfer Interrupt Feature (p. 244)
Isochronous			
Slave	Generic Periodic IN (Interrupt and Isochronous) Data Transfers Without Thresholding (p. 260)		Control Read Transfers (SETUP, Data IN, Status OUT) (p. 235) and Incomplete Isochronous OUT Data Transfers

			in DMA and Slave Modes (p. 248)
DMA	Generic Periodic IN (Interrupt and Isochronous) Data Transfers Without Thresholding (p. 260) and Generic Periodic IN Data Transfers Without Thresholding Using the Periodic Transfer Interrupt Feature (p. 262)		Control Read Transfers (SETUP, Data IN, Status OUT) (p. 235) and Incomplete Isochronous OUT Data Transfers in DMA and Slave Modes (p. 248)

15.4.4.2.1 OUT Data Transfers in Slave and DMA Modes

This section describes the internal data flow and application-level operations during data OUT transfers and setup transactions.

15.4.4.2.1.1 Control Setup Transactions

This section describes how the core handles SETUP packets and the application's sequence for handling setup transactions. To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the host, it must initialize an endpoint as described in Endpoint Initialization (p. 228). See Packet Read from FIFO in Slave Mode (p. 237).

Application Requirements

1. To receive a SETUP packet, the USB_DOEPx_TSIZ.SUPCNT field in a control OUT endpoint must be programmed to a non-zero value. When the application programs the SUPCNT field to a non-zero value, the core receives SETUP packets and writes them to the receive FIFO, irrespective of the USB_DOEPx_CTL.NAK status and USB_DOEPx_CTL.EPENA bit setting. The SUPCNT field is decremented every time the control endpoint receives a SETUP packet. If the SUPCNT field is not programmed to a proper value before receiving a SETUP packet, the core still receives the SETUP packet and decrements the SUPCNT field, but the application possibly is not be able to determine the correct number of SETUP packets received in the Setup stage of a control transfer.
   - USB_DOEPx_TSIZ.SUPCNT = 3
2. In DMA mode, the OUT endpoint must also be enabled, to transfer the received SETUP packet data from the internal receive FIFO to the external memory.
   - USB_DOEPx_CTL.EPENA = 1
3. The application must always allocate some extra space in the Receive Data FIFO, to be able to receive up to three SETUP packets on a control endpoint.
   - The space to be Reserved is (4 * n) + 6 DWORDs, where n is the number of control endpoints supported by the device. Three DWORDs are required for the first SETUP packet, 1 DWORD is required for the Setup Stage Done DWORD, and 6 DWORDs are required to store two extra SETUP packets among all control endpoints.
   - 3 DWORDs per SETUP packet are required to store 8 bytes of SETUP data and 4 bytes of SETUP status (Setup Packet Pattern). The core reserves this space in the receive data
   - FIFO to write SETUP data only, and never uses this space for data packets.
4. In Slave mode, the application must read the 2 DWORDs of the SETUP packet from the receive FIFO. In DMA mode, the core writes the 2 DWORDs of SETUP data to the memory.
5. The application must read and discard the Setup Stage Done DWORD from the receive FIFO.

Internal Data Flow

1. When a SETUP packet is received, the core writes the received data to the receive FIFO, without checking for available space in the receive FIFO and irrespective of the endpoint's NAK and Stall bit settings.
   - The core internally sets the IN NAK and OUT NAK bits for the control IN/OUT endpoints on which the SETUP packet was received.
2. For every SETUP packet received on the USB, 3 DWORDs of data is written to the receive FIFO, and the SUPCNT field is decremented by 1.
   - The first DWORD contains control information used internally by the core
   - The second DWORD contains the first 4 bytes of the SETUP command
   - The third DWORD contains the last 4 bytes of the SETUP command
3. When the Setup stage changes to a Data IN/OUT stage, the core writes an entry (Setup Stage Done DWORD) to the receive FIFO, indicating the completion of the Setup stage.
4. On the AHB side, SETUP packets are emptied either by the DMA or the application. In DMA mode, the SETUP packets (2 DWORDs) are written to the memory location programmed in the USB_DOEPx_DMAADDR register, only if the endpoint is enabled. If the endpoint is not enabled, the data remains in the receive FIFO until the enable bit is set.
5. When either the DMA or the application pops the Setup Stage Done DWORD from the receive FIFO, the core interrupts the application with a USB_DOEPx_INT.SETUP interrupt, indicating it can process the received SETUP packet.
   - The core clears the endpoint enable bit for control OUT endpoints.

Application Programming Sequence

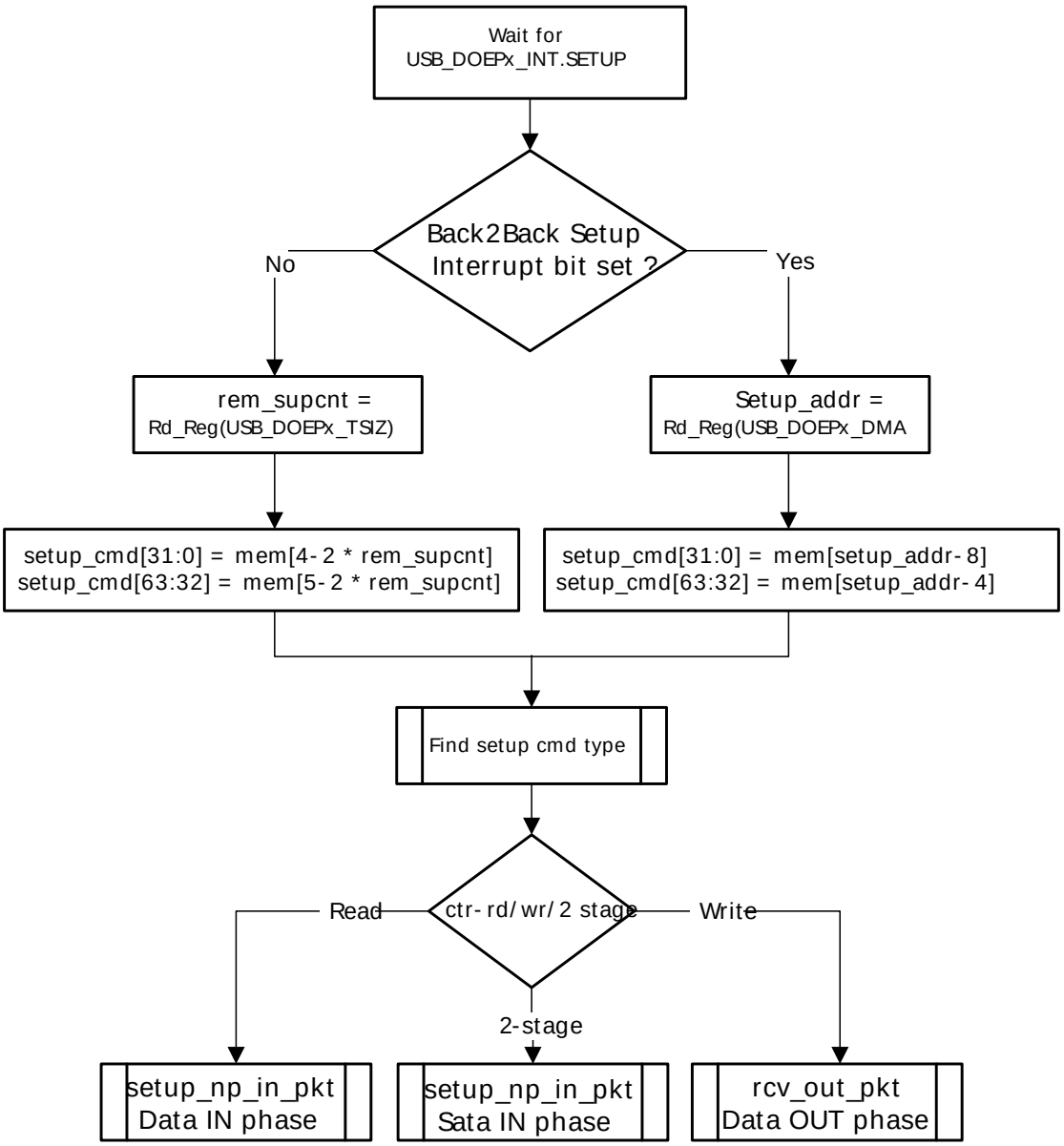
1. Program the USB_DOEPx_TSIZ register.
   - USB_DOEPx_TSIZ.SUPCNT = 3
2. In DMA mode, program the USB_DOEPx_DMAADDR register and USB_DOEPx_CTL register with the endpoint characteristics and set the Endpoint Enable bit (USB_DOEPx_CTL.EPENA).
   - Endpoint Enable = 1
3. In Slave mode, wait for the USB_GINTSTS.RXFLVL interrupt and empty the data packets from the receive FIFO, as explained in Packet Read from FIFO in Slave Mode (p. 237) . This step can be repeated many times.
4. Assertion of the USB_DOEPx_INT.SETUP interrupt marks a successful completion of the SETUP Data Transfer.
   - On this interrupt, the application must read the USB_DOEPx_TSIZ register to determine the number of SETUP packets received and process the last received SETUP packet.
   - In DMA mode, the application must also determine if the interrupt bit USB_DOEPx_INT.BACK2BACKSETUP is set. This bit is set if the core has received more than three back-to-back SETUP packets. If this is the case, the application must ignore the USB_DOEPx_TSIZ.SUPCNT value and use the USB_DOEPx_DMAADDR directly to read out the last SETUP packet received. USB_DOEPx_DMAADDR-8 provides the pointer to the last valid SETUP data.

Note

If the application has not enabled EP0 before the host sends the SETUP packet, the core ACKs the SETUP packet and stores it in the FIFO, but does not write to the memory until EP0 is enabled. When the application enables the EP0 (first enable) and clears the NAK bit at the same time the Host sends DATA OUT, the DATA OUT is stored in the RxFIFO. The OTG core then writes the setup data to the memory and disables the endpoint. Though the application expects a Transfer Complete interrupt for the Data OUT phase, this does not occur, because the SETUP packet, rather than the DATA OUT packet, enables EP0 the first time. Thus, the DATA OUT packet is still in the RxFIFO until the application re-enables EP0. The application must enable EP0 one more time for the core to process the DATA OUT packet.

Figure 15.16 (p. 234) charts this flow.

Figure 15.16. Processing a SETUP Packet



15.4.4.2.1.2 Handling More Than Three Back-to-Back SETUP Packets

Per the USB 2.0 specification, normally, during a SETUP packet error, a host does not send more than three back-to-back SETUP packets to the same endpoint. However, the USB 2.0 specification does not limit the number of back-to-back SETUP packets a host can send to the same endpoint. When this condition occurs, the core generates an interrupt (USB_DOEPx_INT.BACK2BACKSETUP). In DMA mode, the core also rewinds the DMA address for that endpoint (USB_DOEPx_DMAADDR) and overwrites the first SETUP packet in system memory with the fourth, second with the fifth, and so on. If the BACK2BACKSETUP interrupt is asserted, the application must read the OUT endpoint DMA register (USB_DOEPx_DMAADDR) to determine the final SETUP data in system memory.

In DMA mode, the application can mask the BACK2BACKSETUP interrupt, but after receiving the DOEPINT.SETUP interrupt, the application can read the DOEPINT.BACK2BACKSETUP interrupt bit. In Slave mode, the application can use the USB_GINTSTS.RXFLVL interrupt to read out the SETUP packets from the FIFO whenever the core receives the SETUP packet.

15.4.4.2.2 Control Transfers

This section describes the various types of control transfers.

15.4.4.2.2.1 Control Write Transfers (SETUP, Data OUT, Status IN)

This section describes control write transfers.

Application Programming Sequence

1. Assertion of the USB_DOEPx_INT.SETUP Packet interrupt indicates that a valid SETUP packet has been transferred to the application. See OUT Data Transfers in Slave and DMA Modes (p. 232) for more details. At the end of the Setup stage, the application must reprogram the USB_DOEPx_TSIZ.SUPCNT field to 3 to receive the next SETUP packet.
2. If the last SETUP packet received before the assertion of the SETUP interrupt indicates a data OUT phase, program the core to perform a control OUT transfer as explained in Generic Non-Isochronous OUT Data Transfers Without Thresholding in DMA and Slave Modes (p. 240) .

   In DMA mode, the application must reprogram the USB_DOEPx_DMAADDR register to receive a control OUT data packet to a different memory location.
3. In a single OUT data transfer on control endpoint 0, the application can receive up to 64 bytes. If the application is expecting more than 64 bytes in the Data OUT stage, the application must re-enable the endpoint to receive another 64 bytes, and must continue to do so until it has received all the data in the Data stage.
4. Assertion of the USB_DOEPx_INT.Transfer Completed interrupt on the last data OUT transfer indicates the completion of the data OUT phase of the control transfer.
5. On completion of the data OUT phase, the application must do the following.
   - To transfer a new SETUP packet in DMA mode, the application must re-enable the control OUT endpoint as explained in OUT Data Transfers in Slave and DMA Modes (p. 232) .
     - USB_DOEPx_CTL.EPENA = 1
   - To execute the received Setup command, the application must program the required registers in the core. This step is optional, based on the type of Setup command received.
6. For the status IN phase, the application must program the core as described in Generic Non-Periodic (Bulk and Control) IN Data Transfers Without Thresholding in DMA and Slave Mode(p. 255) to perform a data IN transfer.
7. Assertion of the USB_DIEPx_INT.XFERCOMPL interrupt indicates completion of the status IN phase of the control transfer.
8. The previous step must be repeated until the USB_DIEPx_INT.XFERCOMPL interrupt is detected on the endpoint, marking the completion of the control write transfer.

15.4.4.2.2.2 Control Read Transfers (SETUP, Data IN, Status OUT)

This section describes control read transfers.

Application Programming Sequence

1. Assertion of the USB_DOEPx_INT.SETUP Packet interrupt indicates that a valid SETUP packet has been transferred to the application. See OUT Data Transfers in Slave and DMA Modes (p. 232) for more details. At the end of the Setup stage, the application must reprogram the USB_DOEPx_TSIZ.SUPCNT field to 3 to receive the next SETUP packet.
2. If the last SETUP packet received before the assertion of the SETUP interrupt indicates a data IN phase, program the core to perform a control IN transfer as explained in Generic Non-Periodic (Bulk and Control) IN Data Transfers Without Thresholding in DMA and Slave Mode (p. 255) .
3. On a single IN data transfer on control endpoint 0, the application can transmit up to 64 bytes. To transmit more than 64 bytes in the Data IN stage, the application must re-enable the endpoint to transmit another 64 bytes, and must continue to do so, until it has transmitted all the data in the Data stage.
4. The previous step must be repeated until the USB_DIEPx_INT.XFERCOMPL interrupt is detected for every IN transfer on the endpoint.
5. The USB_DIEPx_INT.XFERCOMPL interrupt on the last IN data transfer marks the completion of the control transfer's Data stage.

6. To perform a data OUT transfer in the status OUT phase, the application must program the core as described in OUT Data Transfers in Slave and DMA Modes (p. 232) .
   - The application must program the USB_DCFG.NZSTSOUTHSHK handshake field to a proper setting before transmitting an data OUT transfer for the Status stage.
   - In DMA mode, the application must reprogram the USB_DOEPx_DMAADDR register to receive the control OUT data packet to a different memory location.
7. Assertion of the USB_DOEPx_INT.XFERCOMPL interrupt indicates completion of the status OUT phase of the control transfer. This marks the successful completion of the control read transfer.
   - To transfer a new SETUP packet in DMA mode, the application must re-enable the control OUT endpoint as explained in OUT Data Transfers in Slave and DMA Modes (p. 232) .
   - USB_DOEPx_CTL.EPENA = 1

15.4.4.2.2.3 Two-Stage Control Transfers (SETUP/Status IN)

This section describes two-stage control transfers.

Application Programming Sequence

1. Assertion of the USB_DOEPx_INT.SETUP interrupt indicates that a valid SETUP packet has been transferred to the application. See OUT Data Transfers in Slave and DMA Modes (p. 232) for more detail. To receive the next SETUP packet, the application must reprogram the USB_DOEPx_TSIZ.SUPCNT field to 3 at the end of the Setup stage.
2. Decode the last SETUP packet received before the assertion of the SETUP interrupt. If the packet indicates a two-stage control command, the application must do the following.
   - To transfer a new SETUP packet in DMA mode, the application must re-enable the control OUT endpoint. See OUT Data Transfers in Slave and DMA Modes (p. 232) for details.
     - USB_DOEPx_CTL.EPENA = 1
   - Depending on the type of Setup command received, the application can be required to program registers in the core to execute the received Setup command.
3. For the status IN phase, the application must program the core described in Generic Non-Periodic (Bulk and Control) IN Data Transfers Without Thresholding in DMA and Slave Mode(p. 255) to perform a data IN transfer.
4. Assertion of the USB_DIEPx_INT.XFERCOMPL interrupt indicates the completion of the status IN phase of the control transfer.
5. The previous step must be repeated until the USB_DIEPx_INT.XFERCOMPL interrupt is detected on the endpoint, marking the completion of the two-stage control transfer.

Example: Two-Stage Control Transfer

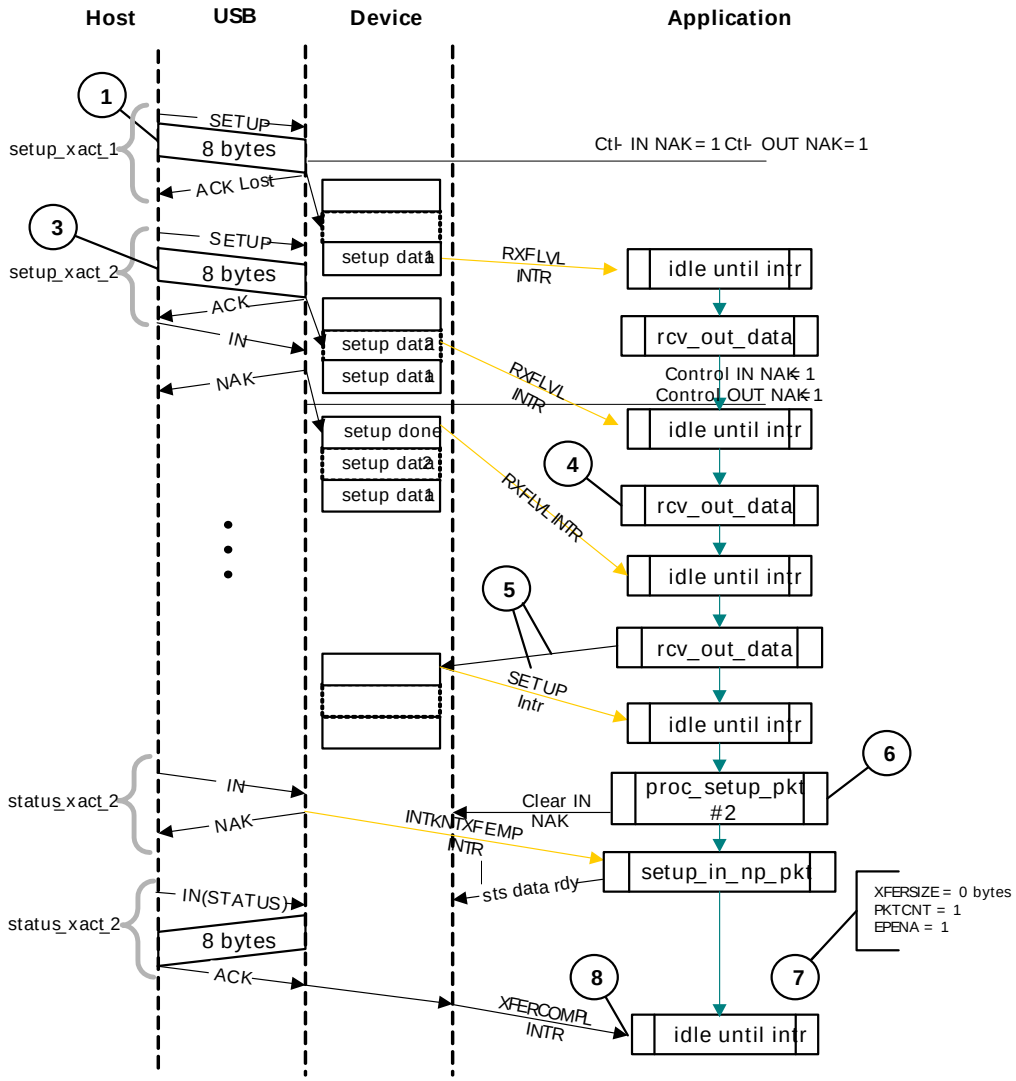
These notes refer to Figure 15.17 (p. 237) .

1. SETUP packet #1 is received on the USB and is written to the receive FIFO, and the core responds with an ACK handshake. This handshake is lost and the host detects a timeout.
2. The SETUP packet in the receive FIFO results in a USB_GINTSTS.RXFLVL interrupt to the application, causing the application to empty the receive FIFO.
3. SETUP packet #2 on the USB is written to the receive FIFO, and the core responds with an ACK handshake.
4. The SETUP packet in the receive FIFO sends the application the USB_GINTSTS.RXFLVL interrupt and the application empties the receive FIFO.
5. After the second SETUP packet, the host sends a control IN token for the status phase. The core issues a NAK response to this token, and writes a Setup Stage Done entry to the receive FIFO. This entry results in a USB_GINTSTS.RXFLVL interrupt to the application, which empties the receive FIFO. After reading out the Setup Stage Done DWORD, the core asserts the USB_DOEPx_INT.SETUP packet interrupt to the application.
6. On this interrupt, the application processes SETUP Packet #2, decodes it to be a two-stage control command, and clears the control IN NAK bit.

- USB_DIEPx_CTL.CNAK = 1

7. When the application clears the IN NAK bit, the core interrupts the application with a USB_DIEPx_INT.INTKNTXFEMP. On this interrupt, the application enables the control IN endpoint with a USB_DIEPx_TSIZ.XFERSIZE of 0 and a USB_DIEPx_TSIZ.PKTCNT of 1. This results in a zero-length data packet for the status IN token on the USB.
8. At the end of the status IN phase, the core interrupts the application with a USB_DIEPx_INT.XFERCOMPL interrupt.

Figure 15.17. Two-Stage Control Transfer



15.4.4.2.2.4 Packet Read from FIFO in Slave Mode

This section describes how to read packets (OUT data and SETUP packets) from the receive FIFO in Slave mode.

1. On catching a USB_GINTSTS.RXFLVL interrupt, the application must read the Receive Status Pop register (USB_GRXSTSP).
2. The application can mask the USB_GINTSTS.RXFLVL interrupt by writing to USB_GINTMSK.RXFLVL = 0, until it has read the packet from the receive FIFO.
3. If the received packet's byte count is not 0, the byte count amount of data is popped from the receive Data FIFO and stored in memory. If the received packet byte count is 0, no data is popped from the Receive Data FIFO.
4. The receive FIFO's packet status readout indicates one of the following.
5. Global OUT NAK Pattern: PKTSTS = Global OUT NAK, BCNT = 0x000, EPNUM = Dont Care (0x0), DPID = Dont Care (0b00). This data indicates that the global OUT NAK bit has taken effect.

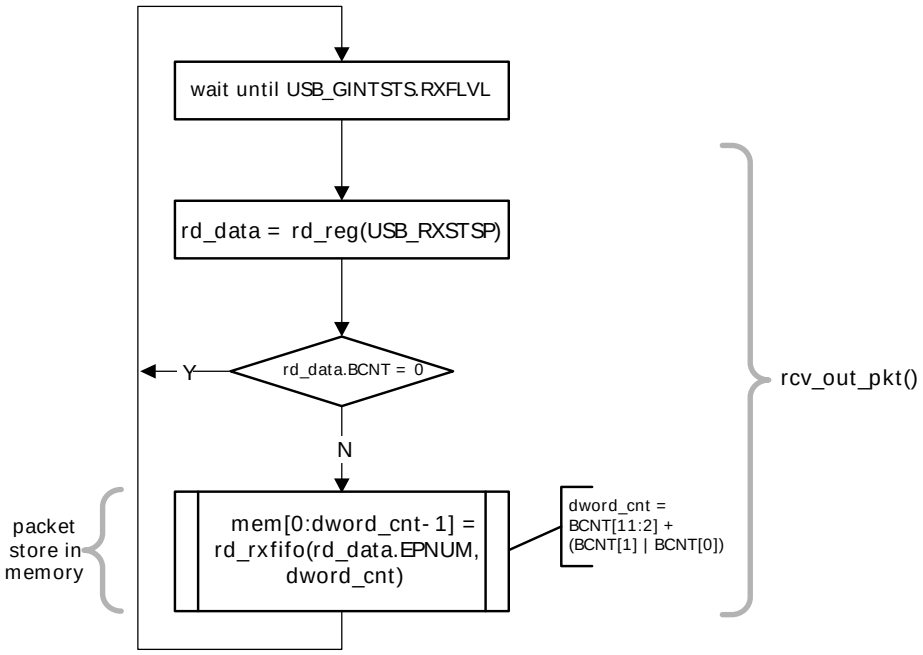
a. SETUP Packet Pattern: PKTSTS = SETUP, BCNT = 0x008, EPNUM = Control EP Num, DPID = D0. This data indicates that a SETUP packet for the specified endpoint is now available for reading from the receive FIFO.
b. Setup Stage Done Pattern: PKTSTS = Setup Stage Done, BCNT = 0x0, EPNUM = Control EP Num, DPID = Don't Care (0b00). This data indicates that the Setup stage for the specified endpoint has completed and the Data stage has started. After this entry is popped from the receive FIFO, the core asserts a Setup interrupt on the specified control OUT endpoint.
c. Data OUT Packet Pattern: PKTSTS = DataOUT, BCNT = size of the Received data OUT packet, EPNUM = EPNum on which the packet was received, DPID = Actual Data PID.
d. Data Transfer Completed Pattern: PKTSTS = Data OUT Transfer Done, BCNT = 0x0, EPNUM = OUT EP Num on which the data transfer is complete, DPID = Dont Care (0b00). This data indicates that a OUT data transfer for the specified OUT endpoint has completed. After this entry is popped from the receive FIFO, the core asserts a Transfer Completed interrupt on the specified OUT endpoint.

The encoding for the PKTSTS is listed in Section 15.6 (p. 294) .

6. After the data payload is popped from the receive FIFO, the USB_GINTSTS.RXFLVL interrupt must be unmasked.
7. Steps 1–5 are repeated every time the application detects assertion of the interrupt line due to USB_GINTSTS.RXFLVL. Reading an empty receive FIFO can result in undefined core behavior.

Figure 15.18 (p. 238) provides a flow chart of this procedure.

Figure 15.18. Receive FIFO Packet Read in Slave Mode



15.4.4.2.2.5 Setting the Global OUT NAK

Internal Data Flow

1. When the application sets the Global OUT NAK (USB_DCTL.SGOUTNAK), the core stops writing data, except SETUP packets, to the receive FIFO. Irrespective of the space availability in the receive FIFO, non-isochronous OUT tokens receive a NAK handshake response, and the core ignores isochronous OUT data packets
2. The core writes the Global OUT NAK pattern to the receive FIFO. The application must reserve enough receive FIFO space to write this data pattern. See Data FIFO RAM Allocation (p. 274) .
3. When either the core (in DMA mode) or the application (in Slave mode) pops the Global OUT NAK pattern DWORD from the receive FIFO, the core sets the USB_GINTSTS.GOUTNAKEFF interrupt.

4. Once the application detects this interrupt, it can assume that the core is in Global OUT NAK mode. The application can clear this interrupt by clearing the USB_DCTL.SGOUTNAK bit.

Application Programming Sequence

1. To stop receiving any kind of data in the receive FIFO, the application must set the Global OUT NAK bit by programming the following field.
   - USB_DCTL.SGOUTNAK = 1
2. Wait for the assertion of the interrupt USB_GINTSTS.GOUTNAKEFF. When asserted, this interrupt indicates that the core has stopped receiving any type of data except SETUP packets.
3. The application can receive valid OUT packets after it has set USB_DCTL.SGOUTNAK and before the core asserts the USB_GINTSTS.GOUTNAKEFF interrupt.
4. The application can temporarily mask this interrupt by writing to the USB_GINTMSK.GOUTNAKEFFMSK bit.
   - USB_GINTMSK.GINNAKEFFMSK = 0
5. Whenever the application is ready to exit the Global OUT NAK mode, it must clear the USB_DCTL.SGOUTNAK bit. This also clears the USB_GINTSTS.GOUTNAKEFF interrupt.
   - USB_DCTL.CGOUTNAK = 1
6. If the application has masked this interrupt earlier, it must be unmasked as follows:
   - USB_GINTMSK.GOUTNAKEFFMSK = 1

15.4.4.2.2.6 Disabling an OUT Endpoint

The application must use this sequence to disable an OUT endpoint that it has enabled.

Application Programming Sequence

1. Before disabling any OUT endpoint, the application must enable Global OUT NAK mode in the core, as described in Setting the Global OUT NAK (p. 238) .
   - USB_DCTL.SGOUTNAK = 1
   - Wait for the USB_GINTSTS.GOUTNAKEFF interrupt
2. Disable the required OUT endpoint by programming the following fields.
   - USB_DOEPx_CTL.EPDIS = 1
   - USB_DOEPx_CTL.SNAK = 1
3. Wait for the USB_DOEPx_INT.EPDISBLD interrupt, which indicates that the OUT endpoint is completely disabled. When the EPDISBLD interrupt is asserted, the core also clears the following bits.
   - USB_DOEPx_CTL.EPDIS = 0
   - USB_DOEPx_CTL.EPENA = 0
4. The application must clear the Global OUT NAK bit to start receiving data from other non-disabled OUT endpoints.
   - USB_DCTL.SGOUTNAK = 0

15.4.4.2.2.7 Stalling a Non-Isochronous OUT Endpoint

This section describes how the application can stall a non-isochronous endpoint.

1. Put the core in the Global OUT NAK mode, as described in Setting the Global OUT NAK (p. 238) .
2. Disable the required endpoint, as described in Section 15.4.4.2.2.6 (p. 239) .
   - When disabling the endpoint, instead of setting the USB_DOEPx_CTL.SNAK bit, set USB_DOEPx_CTL.STALL = 1.
     - The Stall bit always takes precedence over the NAK bit.
3. When the application is ready to end the STALL handshake for the endpoint, the USB_DOEPx_CTL.STALL bit must be cleared.

4. If the application is setting or clearing a STALL for an endpoint due to a SetFeature.Endpoint Halt or ClearFeature.Endpoint Halt command, the Stall bit must be set or cleared before the application sets up the Status stage transfer on the control endpoint.

15.4.4.2.2.8 Generic Non-Isochronous OUT Data Transfers in DMA and Slave Modes

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193) . Before it can communicate with the host, it must initialize an endpoint as described in Endpoint Initialization (p. 228) . See Packet Read from FIFO in Slave Mode (p. 237) .

This section describes a regular non-isochronous OUT data transfer (control, bulk, or interrupt).

Application Requirements

1. Before setting up an OUT transfer, the application must allocate a buffer in the memory to accommodate all data to be received as part of the OUT transfer, then program that buffer's size and start address (in DMA mode) in the endpoint-specific registers.

1. For OUT transfers, the Transfer Size field in the endpoint's Transfer Size register must be a multiple of the maximum packet size of the endpoint, adjusted to the DWORD boundary.

```
if (mps[epnum] mod 4) == 0
    transfer size[epnum] = n * (mps[epnum])  //Dword Aligned
else
    transfer size[epnum] = n * (mps[epnum] + 4 - (mps[epnum] mod 4)) //Non Dword Aligned

packet count[epnum] = n
n > 0
```

2. In DMA mode, the core stores a received data packet in the memory, always starting on a DWORD boundary. If the maximum packet size of the endpoint is not a multiple of 4, the core inserts byte pads at end of a maximum-packet-size packet up to the end of the DWORD.
3. On any OUT endpoint interrupt, the application must read the endpoint's Transfer Size register to calculate the size of the payload in the memory. The received payload size can be less than the programmed transfer size.
   - Payload size in memory = application-programmed initial transfer size – core updated final transfer size
   - Number of USB packets in which this payload was received = application-programmed initial packet count – core updated final packet count

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers, clear the NAK bit, and enable the endpoint to receive the data.
2. Once the NAK bit is cleared, the core starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every packet (maximum packet size or short packet) written to the receive FIFO decrements the Packet Count field for that endpoint by 1.
   - OUT data packets received with Bad Data CRC are flushed from the receive FIFO automatically.
   - After sending an ACK for the packet on the USB, the core discards non-isochronous OUT data packets that the host, which cannot detect the ACK, re-sends. The application does not detect multiple back-to-back data OUT packets on the same endpoint with the same data PID. In this case the packet count is not decremented.
   - If there is no space in the receive FIFO, isochronous or non-isochronous data packets are ignored and not written to the receive FIFO. Additionally, non-isochronous OUT tokens receive a NAK handshake reply.

- In all the above three cases, the packet count is not decremented because no data is written to the receive FIFO.

3. When the packet count becomes 0 or when a short packet is received on the endpoint, the NAK bit for that endpoint is set. Once the NAK bit is set, the isochronous or non-isochronous data packets are ignored and not written to the receive FIFO, and non-isochronous OUT tokens receive a NAK handshake reply.
4. After the data is written to the receive FIFO, either the application (in Slave mode) or the core's DMA engine (in DMA mode), reads the data from the receive FIFO and writes it to external memory, one packet at a time per endpoint.
5. At the end of every packet write on the AHB to external memory, the transfer size for the endpoint is decremented by the size of the written packet.
6. The OUT Data Transfer Completed pattern for an OUT endpoint is written to the receive FIFO on one of the following conditions.
   - The transfer size is 0 and the packet count is 0
   - The last OUT data packet written to the receive FIFO is a short packet (0 <= packet size < maximum packet size)
7. When either the application or the DMA pops this entry (OUT Data Transfer Completed), a Transfer Completed interrupt is generated for the endpoint and the endpoint enable is cleared.

Application Programming Sequence

1. Program the USB_DOEPx_TSIZ register for the transfer size and the corresponding packet count. Additionally, in DMA mode, program the USB_DOEPx_DMAADDR register.
2. Program the USB_DOEPx_CTL register with the endpoint characteristics, and set the Endpoint Enable and ClearNAK bits.
   - USB_DOEPx_CTL.EPENA = 1
   - USB_DOEPx_CTL.CNAK = 1
3. In Slave mode, wait for the USB_GINTSTS.RXFLVL level interrupt and empty the data packets from the receive FIFO as explained in Packet Read from FIFO in Slave Mode (p. 237) .
   - This step can be repeated many times, depending on the transfer size.
4. Asserting the USB_DOEPx_INT.XFERCOMPL interrupt marks a successful completion of the non-isochronous OUT data transfer.
5. Read the USB_DOEPx_TSIZ register to determine the size of the received data payload.

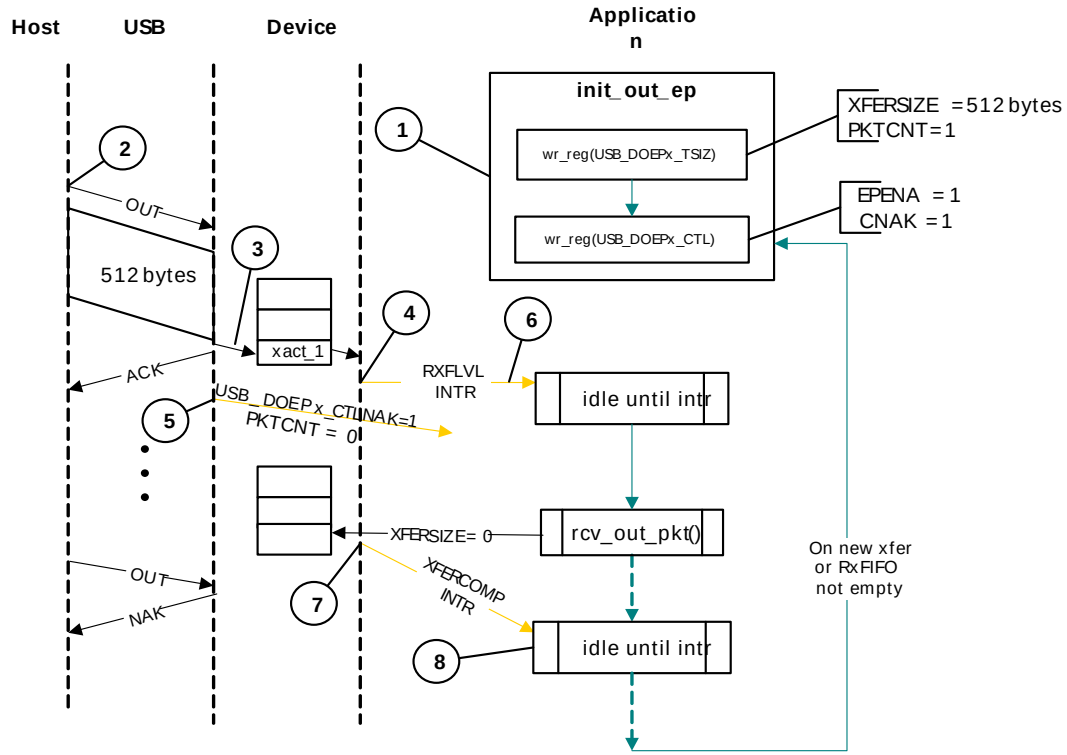
Note

The XFERSIZE is not decremented for the last packet. This is as per design behavior.

Slave Mode Bulk OUT Transaction

Figure 15.19 (p. 242) depicts the reception of a single bulk OUT data packet from the USB to the AHB and describes the events involved in the process.

Figure 15.19. Slave Mode Bulk OUT Transaction



After a SetConfiguration/SetInterface command, the application initializes all OUT endpoints by setting USB_DOEPx_CTL.CNAK = 1 and USB_DOEPx_CTL.EPENA = 1, and setting a suitable XFERSIZE and PKTCNT in the USB_DOEPx_TSIZ register.

1. Host attempts to send data (OUT token) to an endpoint.
2. When the core receives the OUT token on the USB, it stores the packet in the RxFIFO because space is available there.
3. After writing the complete packet in the RxFIFO, the core then asserts the USB_GINTSTS.RXFLVL interrupt.
4. On receiving the PKTCNT number of USB packets, the core sets the NAK bit for this endpoint internally to prevent it from receiving any more packets.
5. The application processes the interrupt and reads the data from the RxFIFO.
6. When the application has read all the data (equivalent to XFERSIZE), the core generates a USB_DOEPx_INT.XFERCOMPL interrupt.
7. The application processes the interrupt and uses the setting of the USB_DOEPx_INT.XFERCOMPL interrupt bit to determine that the intended transfer is complete.

15.4.4.2.2.9 Generic Isochronous OUT Data Transfer in DMA and Slave Modes

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193) . Before it can communicate with the host, it must initialize an endpoint as described in Endpoint Initialization (p. 228) . See Packet Read from FIFO in Slave Mode (p. 237) .

This section describes a regular isochronous OUT data transfer.

Application Requirements:

1. All the application requirements for non-isochronous OUT data transfers also apply to isochronous OUT data transfers
2. For isochronous OUT data transfers, the Transfer Size and Packet Count fields must always be set to the number of maximum-packet-size packets that can be received in a single frame and no more. Isochronous OUT data transfers cannot span more than 1 frame.
   - 1 <= packet count[epnum] <= 3

3. In Slave mode, when isochronous OUT endpoints are supported in the device, the application must read all isochronous OUT data packets from the receive FIFO (data and status) before the end of the periodic frame (USB_GINTSTS.EOPF interrupt). In DMA mode, the application must guarantee enough bandwidth to allow emptying the isochronous OUT data packet from the receive FIFO before the end of each periodic frame.
4. To receive data in the following frame, an isochronous OUT endpoint must be enabled after the USB_GINTSTS.EOPF and before the USB_GINTSTS.SOF.

Internal Data Flow

1. The internal data flow for isochronous OUT endpoints is the same as that for non-isochronous OUT endpoints, but for a few differences.
2. When an isochronous OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame bit must also be set appropriately. The core receives data on a isochronous OUT endpoint in a particular frame only if the following condition is met.
   - USB_DOEPx_CTL.DPIDEOF (Even/Odd frame) = USB_DSTS.SOFFN[0]
3. When either the application or the internal DMA completely reads an isochronous OUT data packet (data and status) from the receive FIFO, the core updates the USB_DOEPx_TSIZ.RXDPIDSUPCNT (Received DPID) field with the data PID of the last isochronous OUT data packet read from the receive FIFO.

Application Programming Sequence

1. Program the USB_DOEPx_TSIZ register for the transfer size and the corresponding packet count. When in DMA mode, also program the USB_DOEPx_DMAADDR register.
2. Program the USB_DOEPx_CTL register with the endpoint characteristics and set the Endpoint Enable, ClearNAK, and Even/Odd frame bits.
   - Endpoint Enable = 1
   - CNAK = 1
   - Even/Odd frame = (0: Even/1: Odd)

1. In Slave mode, wait for the USB_GINTSTS.Rx StsQ level interrupt and empty the data packets from the receive FIFO as explained in Packet Read from FIFO in Slave Mode (p. 237) .
   - This step can be repeated many times, depending on the transfer size.

1. The assertion of the USB_DOEPx_INT.XFERCOMPL interrupt marks the completion of the isochronous OUT data transfer. This interrupt does not necessarily mean that the data in memory is good.
2. This interrupt can not always be detected for isochronous OUT transfers. Instead, the application can detect the USB_GINTSTS.INCOMPLP (Incomplete Isochronous OUT data) interrupt. See Incomplete Isochronous OUT Data Transfers in DMA and Slave Modes (p. 248) , for more details
3. Read the USB_DOEPx_TSIZ register to determine the size of the received transfer and to determine the validity of the data received in the frame. The application must treat the data received in memory as valid only if one of the following conditions is met.
   - USB_DOEPx_TSIZ.RXDPID = D0 and the number of USB packets in which this payload was received = 1
   - USB_DOEPx_TSIZ.RXDPID = D1 and the number of USB packets in which this payload was received = 2
   - USB_DOEPx_TSIZ.RXDPID = D2 and the number of USB packets in which this payload was received = 3
     - The number of USB packets in which this payload was received = App Programmed Initial Packet Count – Core Updated Final Packet Count

The application can discard invalid data packets.

15.4.4.2.2.10 Generic Interrupt OUT Data Transfers Using Periodic Transfer Interrupt Feature

This section describes a regular INTR OUT data transfer with the Periodic Transfer Interrupt feature.

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193) . Before it can communicate with the host, it must initialize an endpoint as described in Endpoint Initialization (p. 228) . See Packet Read from FIFO in Slave Mode (p. 237) .

Application Requirements

1. Before setting up a periodic OUT transfer, the application must allocate a buffer in the memory to accommodate all data to be received as part of the OUT transfer, then program that buffer's size and start address in the endpoint-specific registers.
2. For Interrupt OUT transfers, the Transfer Size field in the endpoint's Transfer Size register must be a multiple of the maximum packet size of the endpoint, adjusted to the DWORD boundary. The Transfer Size programmed can span across multiple frames based on the periodicity after which the application want to receive the USB_DOEPx_INT.XFERCOMPL interrupt
   - transfer size[epnum] = n * (mps[epnum] + 4 - (mps[epnum] mod 4))
   - packet count[epnum] = n
   - n > 0 (Higher value of n reduces the periodicity of the USB_DOEPx_INT.XFERCOMPL interrupt)
   - 1 < packet count[epnum] < n (Higher value of n reduces the periodicity of the USB_DOEPx_INT.XFERCOMPL interrupt)
3. In DMA mode, the core stores a received data packet in the memory, always starting on a DWORD boundary. If the maximum packet size of the endpoint is not a multiple of 4, the core inserts byte pads at end of a maximum-packet-size packet up to the end of the DWORD. The application will not be informed about the frame number on which a specific packet has been received.
4. On USB_DOEPx_INT.XFERCOMPL interrupt, the application must read the endpoint's Transfer Size register to calculate the size of the payload in the memory. The received payload size can be less than the programmed transfer size.
   - Payload size in memory = application-programmed initial transfer size – core updated final transfer size
   - Number of USB packets in which this payload was received = application-programmed initial packet count – core updated final packet count.
   - If for some reason, the host stops sending tokens, there are no interrupts to the application, and the application must timeout on its own.
5. The assertion of the USB_DOEPx_INT.XFERCOMPL interrupt marks the completion of the interrupt OUT data transfer. This interrupt does not necessarily mean that the data in memory is good.
6. Read the USB_DOEPx_TSIZ register to determine the size of the received transfer and to determine the validity of the data received in the frame.

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers, clear the NAK bit, and enable the endpoint to receive the data.
   - The application must enable the USB_DCTL.IGNRFRMNUM
2. When an interrupt OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame will be ignored by the core.

1. Once the NAK bit is cleared, the core starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every packet (maximum packet size or short packet) written to the receive FIFO decrements the Packet Count field for that endpoint by 1.
   - OUT data packets received with Bad Data CRC or any packet error are flushed from the receive FIFO automatically.
   - Interrupt packets with PID errors are not passed to application. Core discards the packet, sends ACK and does not decrement packet count.

- If there is no space in the receive FIFO, interrupt data packets are ignored and not written to the receive FIFO. Additionally, interrupt OUT tokens receive a NAK handshake reply.

2. When the packet count becomes 0 or when a short packet is received on the endpoint, the NAK bit for that endpoint is set. Once the NAK bit is set, the isochronous or interrupt data packets are ignored and not written to the receive FIFO, and interrupt OUT tokens receive a NAK handshake reply.
3. After the data is written to the receive FIFO, the core's DMA engine reads the data from the receive FIFO and writes it to external memory, one packet at a time per endpoint.
4. At the end of every packet write on the AHB to external memory, the transfer size for the endpoint is decremented by the size of the written packet.
5. The OUT Data Transfer Completed pattern for an OUT endpoint is written to the receive FIFO on one of the following conditions.
   - The transfer size is 0 and the packet count is 0.
   - The last OUT data packet written to the receive FIFO is a short packet (0 < packet size < maximum packet size)
6. When either the application or the DMA pops this entry (OUT Data Transfer Completed), a Transfer Completed interrupt is generated for the endpoint and the endpoint enable is cleared.

15.4.4.2.2.11 Generic Isochronous OUT Data Transfers Using Periodic Transfer Interrupt Feature

This section describes a regular isochronous OUT data transfer with the Periodic Transfer Interrupt feature.

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193) . Before it can communicate with the host, it must initialize an endpoint as described in Endpoint Initialization (p. 228) . For packet writes in Slave mode, see: Packet Read from FIFO in Slave Mode (p. 237) .

Application Requirements

1. Before setting up ISOC OUT transfers spanned across multiple frames, the application must allocate buffer in the memory to accommodate all data to be received as part of the OUT transfers, then program that buffer's size and start address in the endpoint-specific registers.
   - The application must mask the USB_GINTSTS.INCOMPLP (Incomplete ISO OUT).
   - The application must enable the USB_DCTL.IGNRFRMNUM
2. For ISOC transfers, the Transfer Size field in the USB_DOEPx_TSIZ.XFERSIZE register must be a multiple of the maximum packet size of the endpoint, adjusted to the DWORD boundary. The Transfer Size programmed can span across multiple frames based on the periodicity after which the application wants to receive the USB_DOEPx_INT.XFERCOMPL interrupt
   - transfer size[epnum] = n * (mps[epnum] + 4 - (mps[epnum] mod 4))
   - packet count[epnum] = n
   - n > 0 (Higher value of n reduces the periodicity of the USB_DOEPx_INT.XFERCOMPL interrupt)
   - 1 =< packet count[epnum] =< n (Higher value of n reduces the periodicity of the USB_DOEPx_INT.XFERCOMPL interrupt).
3. In DMA mode, the core stores a received data packet in the memory, always starting on a DWORD boundary. If the maximum packet size of the endpoint is not a multiple of 4, the core inserts byte pads at end of a maximum-packet-size packet up to the end of the DWORD. The application will not be informed about the frame number and the PID value on which a specific OUT packet has been received.
4. The assertion of the USB_DOEPx_INT.XFERCOMPL interrupt marks the completion of the isochronous OUT data transfer. This interrupt does not necessarily mean that the data in memory is good.
   - On USB_DOEPx_INT.XFERCOMPL, the application must read the endpoint's Transfer Size register to calculate the size of the payload in the memory.
   - Payload size in memory = application-programmed initial transfer size - core updated final transfer size

- Number of USB packets in which this payload was received = application-programmed initial packet count – core updated final packet count.
- If for some reason, the host stop sending tokens, there will be no interrupt to the application, and the application must timeout on its own.

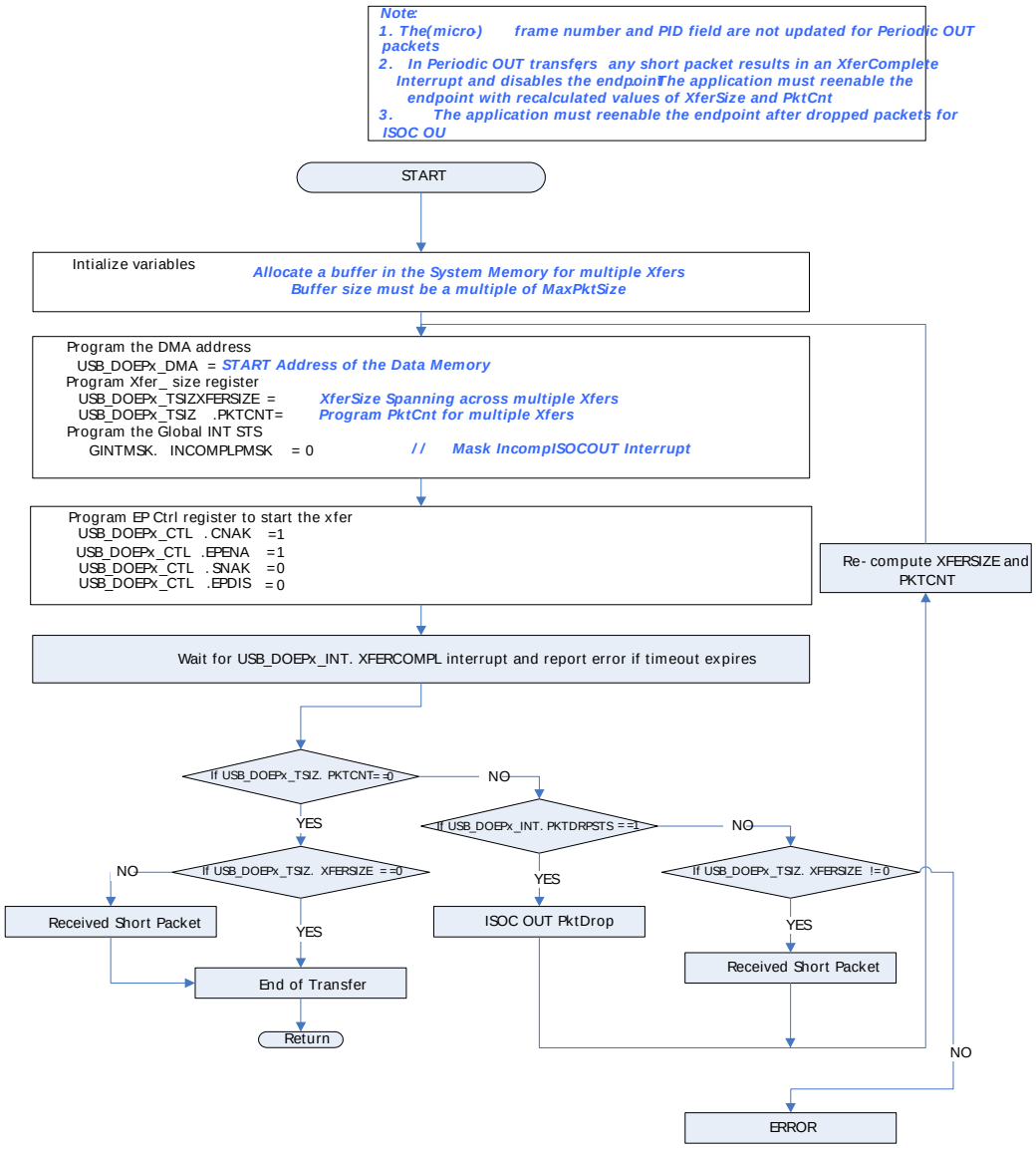
5. The assertion of the USB_DOEPx_INT.XFERCOMPL can also mark a packet drop on USB due to unavailability of space in the RxFifo or due to any packet errors.
   - The application must read the USB_DOEPx_INT.PKTDRPSTS (USB_DOEPx_INT.Bit[11] is now used as the USB_DOEPx_INT.PKTDRPSTS) register to differentiate whether the USB_DOEPx_INT.XFERCOMPL was generated due to the normal end of transfer or due to dropped packets. In case of packets being dropped on the USB due to unavailability of space in the RxFifo or due to any packet errors the endpoint enable bit is cleared.
   - In case of packet drop on the USB application must re-enable the endpoint after recalculating the values USB_DOEPx_TSIZ.XFERSIZE and USB_DOEPx_TSIZ.PKTCNT.
   - Payload size in memory = application-programmed initial transfer size - core updated final transfer size
   - Number of USB packets in which this payload was received = application-programmed initial packet count - core updated final packet count.

Note

Due to application latencies it is possible that DOEPINT.XFERCOMPL interrupt is generated without DOEPINT.PKTDRPSTS being set, This scenario is possible only if back-to-back packets are dropped for consecutive frames and the PKTDRPSTS is merged, but the XFERSIZE and PktCnt values for the endpoint are nonzero. In this case, the application must proceed further by programming the PKTCNT and XFERSIZE register for the next frame, as it would if PKTDRPSTS were being set.

Figure 15.20 (p. 247) gives the application flow for Isochronous OUT Periodic Transfer Interrupt feature.

Figure 15.20. ISOC OUT Application Flow for Periodic Transfer Interrupt Feature



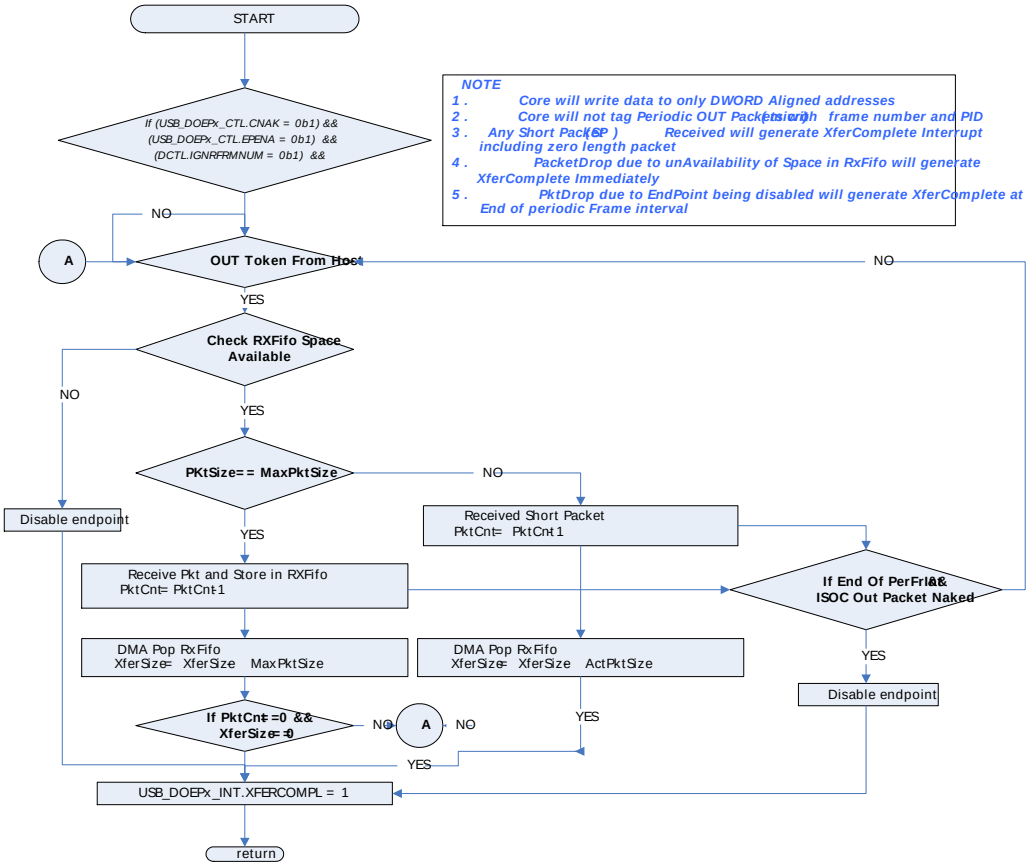
Internal Data Flow

1. The application must set the Transfer Size, Packets to be received in a frame and Packet Count Fields in the endpoint-specific registers, clear the NAK bit, and enable the endpoint to receive the data.
2. When an isochronous OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame will be ignored by the core.
3. Once the NAK bit is cleared, the core starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every packet (maximum packet size or short packet) written to the receive FIFO decrements the Packet Count field for that endpoint by 1.
4. When the packet count becomes 0 or when a short packet is received on the endpoint, the NAK bit for that endpoint is set. Once the NAK bit is set, the ISOC packets are ignored and not written to the receive FIFO.
5. After the data is written to the receive FIFO, the core's DMA engine, reads the data from the receive FIFO and writes it to external memory, one packet at a time per endpoint.
6. At the end of every packet write on the AHB to external memory, the transfer size for the endpoint is decremented by the size of the written packet.
7. The OUT Data Transfer Completed pattern for an OUT endpoint is written to the receive FIFO on one of the following conditions.
   - The transfer size is 0 and the packet count is 0

- The last OUT data packet written to the receive FIFO is a short packet (0 < packet size < maximum packet size).

8. When the DMA pops this entry (OUT Data Transfer Completed), a Transfer Completed interrupt is generated for the endpoint or the endpoint enable is cleared.
9. OUT data packets received with Bad Data CRC or any packet error are flushed from the receive FIFO automatically.
   - In these two cases, the packet count and transfer size registers are not decremented because no data is written to the receive FIFO.

Figure 15.21. Isochronous OUT Core Internal Flow for Periodic Transfer Interrupt Feature



15.4.4.2.12 Incomplete Isochronous OUT Data Transfers in DMA and Slave Modes

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193) . Before it can communicate with the host, it must initialize an endpoint as described in Endpoint Initialization (p. 228) . See Packet Read from FIFO in Slave Mode (p. 237) .

This section describes the application programming sequence when isochronous OUT data packets are dropped inside the core.

Internal Data Flow

1. For isochronous OUT endpoints, the USB_DOEPx_INT.XFERCOMPL interrupt possibly is not always asserted. If the core drops isochronous OUT data packets, the application could fail to detect the USB_DOEPx_INT.XFERCOMPL interrupt under the following circumstances.
   - When the receive FIFO cannot accommodate the complete ISO OUT data packet, the core drops the received ISO OUT data.
   - When the isochronous OUT data packet is received with CRC errors
   - When the isochronous OUT token received by the core is corrupted
   - When the application is very slow in reading the data from the receive FIFO

2. When the core detects an end of periodic frame before transfer completion to all isochronous OUT endpoints, it asserts the USB_GINTSTS.INCOMPLP (Incomplete Isochronous OUT data) interrupt, indicating that a USB_DOEPx_INT.XFERCOMPL interrupt is not asserted on at least one of the isochronous OUT endpoints. At this point, the endpoint with the incomplete transfer remains enabled, but no active transfers remains in progress on this endpoint on the USB.
3. This step is applicable only if the core is operating in slave mode. Application Programming Sequence
4. This step is applicable only if the core is operating in slave mode. Asserting the USB_GINTSTS.INCOMPLP (Incomplete Isochronous OUT data) interrupt indicates that in the current frame, at least one isochronous OUT endpoint has an incomplete transfer.
5. If this occurs because isochronous OUT data is not completely emptied from the endpoint, the application must ensure that the DMA or the application empties all isochronous OUT data (data and status) from the receive FIFO before proceeding.
   - When all data is emptied from the receive FIFO, the application can detect the USB_DOEPx_INT.XFERCOMPL interrupt. In this case, the application must re-enable the endpoint to receive isochronous OUT data in the next frame, as described in Control Read Transfers (SETUP, Data IN, Status OUT) (p. 235) .
6. When it receives a USB_GINTSTS.incomplete Isochronous OUT data interrupt, the application must read the control registers of all isochronous OUT endpoints (USB_DOEPx_CTL) to determine which endpoints had an incomplete transfer in the current frame. An endpoint transfer is incomplete if both the following conditions are met.
   - USB_DOEPx_CTL.DPIDEOF (Even/Odd frame) = USB_DSTS.SOFFN[0]
   - USB_DOEPx_CTL.EPENA (Endpoint Enable) = 1
7. The previous step must be performed before the USB_GINTSTS.SOF interrupt is detected, to ensure that the current frame number is not changed.
8. For isochronous OUT endpoints with incomplete transfers, the application must discard the data in the memory and disable the endpoint by setting the USB_DOEPx_CTL.EPDIS (Endpoint Disable) bit.
9. Wait for the USB_DOEPx_INT.EPDIS (Endpoint Disabled) interrupt and enable the endpoint to receive new data in the next frame as explained in Control Read Transfers (SETUP, Data IN, Status OUT) (p. 235) .
   - Because the core can take some time to disable the endpoint, the application possibly is not able to receive the data in the next frame after receiving bad isochronous data.

15.4.4.2.3 IN Data Transfers in Slave and DMA Modes

This section describes the internal data flow and application-level operations during IN data transfers.

- Packet Write in Slave Mode (p. 250)
- Setting Global Non-Periodic IN Endpoint NAK (p. 250)
- Setting IN Endpoint NAK (p. 250)
- IN Endpoint Disable (p. 251)
- Bulk IN Stall (p. 252)
- Incomplete Isochronous IN Data Transfers (p. 252)
- Stalling Non-Isochronous IN Endpoints (p. 253)
- Worst-Case Response Time (p. 254)
- Choosing the Value of USB_GUSBCFG.USBTRDTIM (p. 254)
- Handling Babble Conditions (p. 255)
- Generic Non-Periodic (Bulk and Control) IN Data Transfers Without Thresholding in DMA and Slave Mode (p. 255)
- Examples (p. 257)
-
- Generic Periodic IN Data Transfers Without Thresholding Using the Periodic Transfer Interrupt Feature (p. 262)

15.4.4.2.3.1 Packet Write in Slave Mode

This section describes how the application writes data packets to the endpoint FIFO in Slave mode.

1. The application can either choose polling or interrupt mode.
   - In polling mode, application monitors the status of the endpoint transmit data FIFO, by reading the USB_DIEPx_TXFSTS register, to determine, if there is enough space in the data FIFO.
   - In interrupt mode, application waits for the USB_DIEPx_INT.TXFEMP interrupt and then reads the USB_DIEPx_TXFSTS register, to determine, if there is enough space in the data FIFO.
   - To write a single non-zero length data packet, there must be space to write the entire packet is the data FIFO.
   - For writing zero length packet, application must not look for FIFO space.
2. Using one of the above mentioned methods, when the application determines that there is enough space to write a transmit packet, the application must first write into the endpoint control register, before writing the data into the data FIFO. The application, typically must do a read modify write on the USB_DIEPx_CTL, to avoid modifying the contents of the register, except for setting the Endpoint Enable bit.

The application can write multiple packets for the same endpoint, into the transmit FIFO, if space is available. For periodic IN endpoints, application must write packets only for one frame. It can write packets for the next periodic transaction, only after getting transfer complete for the previous transaction.

15.4.4.2.3.2 Setting Global Non-Periodic IN Endpoint NAK

Internal Data Flow

1. When the application sets the Global Non-periodic IN NAK bit (USB_DCTL.SGNPINNAK), the core stops transmitting data on the non-periodic endpoint, irrespective of data availability in the Non-periodic Transmit FIFO.
2. Non-isochronous IN tokens receive a NAK handshake reply
3. The core asserts the USB_GINTSTS.GINNAKEFF interrupt in response to the USB_DCTL.SGNPINNAK bit.
4. Once the application detects this interrupt, it can assume that the core is in the Global Non-periodic IN NAK mode. The application can clear this interrupt by clearing the USB_DCTL.SGNPINNAK bit.

Application Programming Sequence

1. To stop transmitting any data on non-periodic IN endpoints, the application must set the USB_DCTL.SGNPINNAK bit. To set this bit, the following field must be programmed
   - USB_DCTL.SGNPINNAK = 1
2. Wait for the assertion of the USB_GINTSTS.GINNAKEFF interrupt. This interrupt indicates the core has stopped transmitting data on the non-periodic endpoints.
3. The core can transmit valid non-periodic IN data after the application has set the USB_DCTL.SGNPINNAK bit, but before the assertion of the USB_GINTSTS.GINNAKEFF interrupt.
4. The application can optionally mask this interrupt temporarily by writing to the USB_GINTMSK.GINNAKEFFMSK bit.
   - USB_GINTMSK.GINNAKEFFMSK = 0
5. To exit Global Non-periodic IN NAK mode, the application must clear the USB_DCTL.SGNPINNAK. This also clears the USB_GINTSTS.GINNAKEFF interrupt.
   - USB_DCTL.SGNPINNAK = 1
6. If the application has masked this interrupt earlier, it must be unmasked as follows:
   - USB_GINTMSK.GINNAKEFFMSK = 1

15.4.4.2.3.3 Setting IN Endpoint NAK

Internal Data Flow

1. When the application sets the IN NAK for a particular endpoint, the core stops transmitting data on the endpoint, irrespective of data availability in the endpoint's transmit FIFO.
2. Non-isochronous IN tokens receive a NAK handshake reply
   - Isochronous IN tokens receive a zero-data-length packet reply
3. The core asserts the USB_DIEPx_INT.INEPNAKEFF (IN NAK Effective) interrupt in response to the USB_DIEPx_CTL.SNAK (Set NAK) bit.
4. Once this interrupt is seen by the application, the application can assume that the endpoint is in IN NAK mode. This interrupt can be cleared by the application by setting the USB_DIEPx_CTL. Clear NAK bit.

Application Programming Sequence

1. To stop transmitting any data on a particular IN endpoint, the application must set the IN NAK bit. To set this bit, the following field must be programmed.
   - USB_DIEPx_CTL.SNAK = 1
2. Wait for assertion of the USB_DIEPx_INT.INEPNAKEFF (NAK Effective) interrupt. This interrupt indicates the core has stopped transmitting data on the endpoint.
3. The core can transmit valid IN data on the endpoint after the application has set the NAK bit, but before the assertion of the NAK Effective interrupt.
4. The application can mask this interrupt temporarily by writing to the USB_DIEPMSK.INEPNAKEFFMSK (NAK Effective) bit.
   - USB_DIEPMSK.INEPNAKEFFMSK (NAK Effective) = 0
5. To exit Endpoint NAK mode, the application must clear the USB_DIEPx_CTL.NAK status. This also clears the USB_DIEPx_INT.INEPNAKEFF (NAK Effective) interrupt.
   - USB_DIEPx_CTL.CNAK = 1
6. If the application masked this interrupt earlier, it must be unmasked as follows:
   - USB_DIEPMSK.INEPNAKEFFMSK (NAK Effective) = 1

15.4.4.2.3.4 IN Endpoint Disable

Use the following sequence to disable a specific IN endpoint (periodic/non-periodic) that has been previously enabled.

Application Programming Sequence:

1. In Slave mode, the application must stop writing data on the AHB, for the IN endpoint to be disabled.
2. The application must set the endpoint in NAK mode. See Setting IN Endpoint NAK (p. 250) .
   - USB_DIEPx_CTL.SNAK = 1
3. Wait for USB_DIEPx_INT.INEPNAKEFF (NAK Effective) interrupt.
4. Set the following bits in the USB_DIEPx_CTL register for the endpoint that must be disabled.
   - USB_DIEPx_CTL.EPDIS (Endpoint Disable) = 1
   - USB_DIEPx_CTL.SNAK = 1
5. Assertion of USB_DIEPx_INT.EPDISBLD (Endpoint Disabled) interrupt indicates that the core has completely disabled the specified endpoint. Along with the assertion of the interrupt, the core also clears the following bits.
   - USB_DIEPx_CTL.EPENA = 0
   - USB_DIEPx_CTL.EPDIS = 0
6. The application must read the USB_DIEPx_TSIZ register for the periodic IN EP, to calculate how much data on the endpoint was transmitted on the USB.
7. The application must flush the data in the Endpoint transmit FIFO, by setting the following fields in the USB_GRSTCTL register.
   - USB_GRSTCTL.TXFNUM = Endpoint Transmit FIFO Number
   - USB_GRSTCTL.TXFFLSH = 1

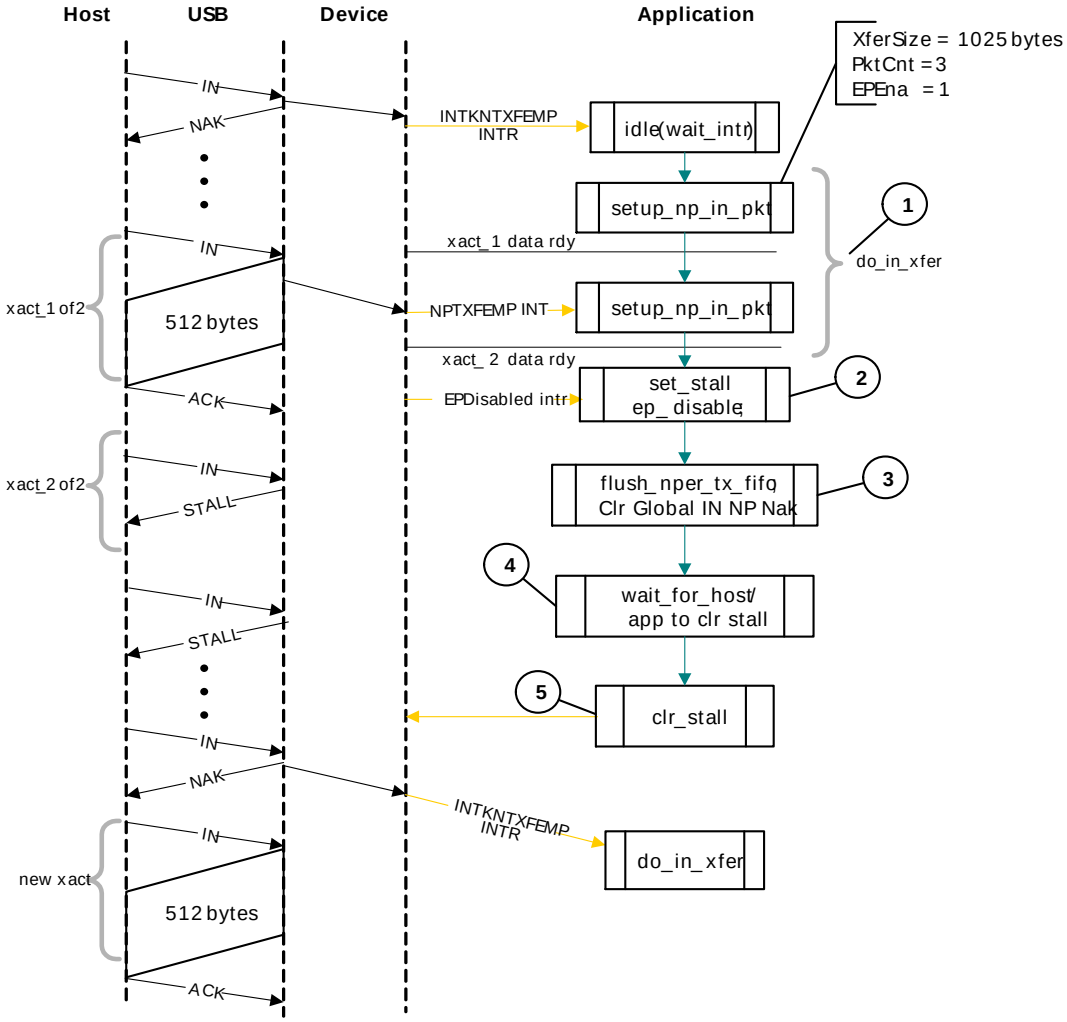
The application must poll the USB_GRSTCTL register, until the TXFFLSH bit is cleared by the core, which indicates the end of flush operation. To transmit new data on this endpoint, the application can re-enable the endpoint at a later point.

15.4.4.2.3.5 Bulk IN Stall

These notes refer to Figure 15.22 (p. 252)

1. The application has scheduled an IN transfer on receiving the USB_DIEPx_INT.INTKNTXFEMP (IN Token Received When TxFIFO Empty) interrupt.
2. When the transfer is in progress, the application must force a STALL on the endpoint. This could be because the application has received a SetFeature.Endpoint Halt command. The application sets the Stall bit, disables the endpoint and waits for the USB_DIEPx_INT.EPDISBLD (Endpoint Disabled) interrupt. This generates STALL handshakes for the endpoint on the USB.
3. On receiving the interrupt, the application flushes the Non-periodic Transmit FIFO and clears the USB_DCTL.SGNPINNAK (Global IN NP NAK) bit.
4. On receiving the ClearFeature.Endpoint Halt command, the application clears the Stall bit.
5. The endpoint behaves normally and the application can re-enable the endpoint for new transfers

Figure 15.22. Bulk IN Stall



15.4.4.2.3.6 Incomplete Isochronous IN Data Transfers

This section describes what the application must do on an incomplete isochronous IN data transfer.

Internal Data Flow

1. An isochronous IN transfer is treated as incomplete in one of the following conditions.

a. The core receives a corrupted isochronous IN token on at least one isochronous IN endpoint. In this case, the application detects a USB_GINTSTS.INCOMPISOIN (Incomplete Isochronous IN Transfer) interrupt.
b. The application or DMA is slow to write the complete data payload to the transmit FIFO and an IN token is received before the complete data payload is written to the FIFO. In this case, the application detects a USB_DIEPx_INT.INTKNTXFEMP (IN Token Received When TxFIFO Empty) interrupt. The application can ignore this interrupt, as it eventually results in a USB_GINTSTS.INCOMPISOIN (Incomplete Isochronous IN Transfer) interrupt at the end of periodic frame.
    i. The core transmits a zero-length data packet on the USB in response to the received IN token.
2. In either of the aforementioned cases, in Slave mode, the application must stop writing the data payload to the transmit FIFO as soon as possible.
3. The application must set the NAK bit and the disable bit for the endpoint. In DMA mode, the core automatically stops fetching the data payload when the endpoint disable bit is set.
4. The core disables the endpoint, clears the disable bit, and asserts the Endpoint Disable interrupt for the endpoint.

Application Programming Sequence

1. The application can ignore the USB_DIEPx_INT.INTKNTXFEMP (IN Token Received When TxFIFO empty) interrupt on any isochronous IN endpoint, as it eventually results in a USB_GINTSTS.INCOMPISOIN (Incomplete Isochronous IN Transfer) interrupt.
2. Assertion of the USB_GINTSTS.INCOMPISOIN (Incomplete Isochronous IN Transfer) interrupt indicates an incomplete isochronous IN transfer on at least one of the isochronous IN endpoints.
3. The application must read the Endpoint Control register for all isochronous IN endpoints to detect endpoints with incomplete IN data transfers.
4. In Slave mode, the application must stop writing data to the Periodic Transmit FIFOs associated with these endpoints on the AHB.
5. In both modes of operation, program the following fields in the USB_DIEPx_CTL register to disable the endpoint.
    - USB_DIEPx_CTL.SNAK = 1
    - USB_DIEPx_CTL.EPDIS (Endpoint Disable) = 1
6. The USB_DIEPx_INT.EPDISBLD (Endpoint Disabled) interrupt's assertion indicates that the core has disabled the endpoint.
    - At this point, the application must flush the data in the associated transmit FIFO or overwrite the existing data in the FIFO by enabling the endpoint for a new transfer in the next frame. To flush the data, the application must use the USB_GRSTCTL register.

15.4.4.2.3.7 Stalling Non-Isochronous IN Endpoints

This section describes how the application can stall a non-isochronous endpoint.

Application Programming Sequence

1. Disable the IN endpoint to be stalled. Set the Stall bit as well.
2. USB_DIEPx_CTL.EPDIS (Endpoint Disable) = 1, when the endpoint is already enabled
    - USB_DIEPx_CTL.STALL = 1
    - The Stall bit always takes precedence over the NAK bit
3. Assertion of the USB_DIEPx_INT.EPDISBLD (Endpoint Disabled) interrupt indicates to the application that the core has disabled the specified endpoint.
4. The application must flush the Non-periodic or Periodic Transmit FIFO, depending on the endpoint type. In case of a non-periodic endpoint, the application must re-enable the other non-periodic endpoints, which do not need to be stalled, to transmit data.
5. Whenever the application is ready to end the STALL handshake for the endpoint, the USB_DIEPx_CTL.STALL bit must be cleared.

6. If the application sets or clears a STALL for an endpoint due to a SetFeature.Endpoint Halt command or ClearFeature.Endpoint Halt command, the Stall bit must be set or cleared before the application sets up the Status stage transfer on the control endpoint.

Special Case: Stalling the Control IN/OUT Endpoint

The core must stall IN/OUT tokens if, during the Data stage of a control transfer, the host sends more IN/OUT tokens than are specified in the SETUP packet. In this case, the application must to enable USB_DIEPx_INT.INTKNTXFEMP and USB_DOEPx_INT.OUTTKNEPDIS interrupts during the Data stage of the control transfer, after the core has transferred the amount of data specified in the SETUP packet. Then, when the application receives this interrupt, it must set the STALL bit in the corresponding endpoint control register, and clear this interrupt.

15.4.4.2.3.8 Worst-Case Response Time

When the acts as a device, there is a worst case response time for any tokens that follow an isochronous OUT. This worst case response time depends on the AHB clock frequency.

The core registers are in the AHB domain, and the core does not accept another token before updating these register values. The worst case is for any token following an isochronous OUT, because for an isochronous transaction, there is no handshake and the next token could come sooner. This worst case value is 7 PHY clocks in FS mode.

If this worst case condition occurs, the core responds to bulk/interrupt tokens with a NAK and drops isochronous and SETUP tokens. The host interprets this as a timeout condition for SETUP and retries the SETUP packet. For isochronous transfers, the INCOMPISOIN and INCOMPLP interrupts inform the application that isochronous IN/OUT packets were dropped.

15.4.4.2.3.9 Choosing the Value of USB_GUSBCFG.USBTRDTIM

The value in USB_GUSBCFG.USBTRDTIM is the time it takes for the MAC, in terms of PHY clocks after it has received an IN token, to get the FIFO status, and thus the first data from PFC (Packet FIFO Controller) block. This time involves the synchronization delay between the PHY and AHB clocks. This delay is 5 clocks.

Once the MAC receives an IN token, this information (token received) is synchronized to the AHB clock by the PFC (the PFC runs on the AHB clock). The PFC then reads the data from the SPRAM and writes it into the dual clock source buffer. The MAC then reads the data out of the source buffer (4 deep).

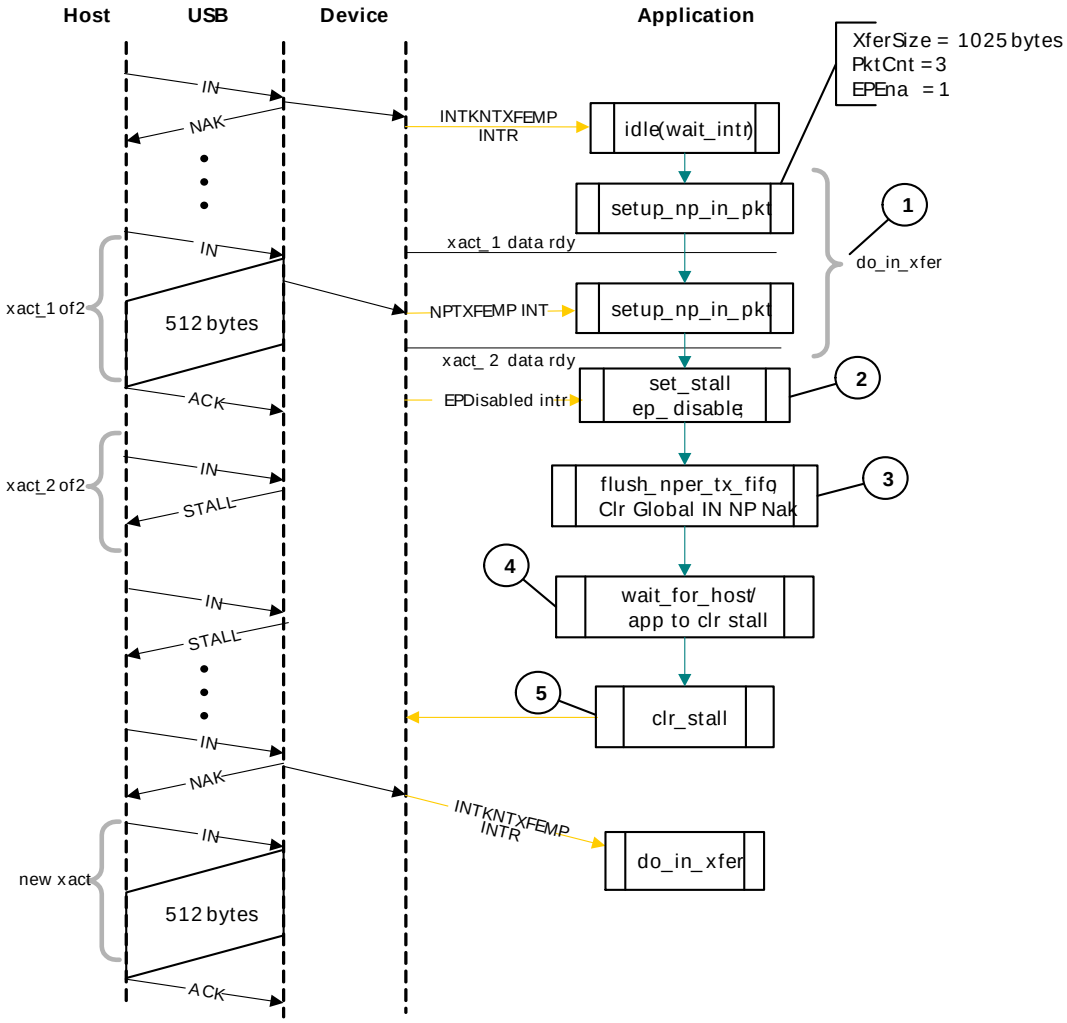
If the AHB is running at a higher frequency than the PHY (in Low-speed mode), the application can use a smaller value for USB_GUSBCFG.USBTRDTIM. Figure 15.23 (p. 255) explains the 5-clock delay. This diagram has the following signals:

- tkn_rcvd: Token received information from MAC to PFC
- dynced_tkn_rcvd: Doubled sync tkn_rcvd, from pclk to hclk domain
- spr_read: Read to SPRAM
- spr_addr: Address to SPRAM
- spr_rdata: Read data from SPRAM
- srcbuf_push: Push to the source buffer
- srcbuf_rdata: Read data from the source buffer. Data seen by MAC

The application can use the following formula to calculate the value of USB_GUSBCFG.USBTRDTIM:

4 * AHB Clock + 1 PHY Clock = (2 clock sync + 1 clock memory address + 1 clock memory data from sync RAM) + (1 PHY Clock (next PHY clock MAC can sample the 2-clock FIFO output)

Figure 15.23. USBTRDTIM Max Timing Case ERROR wrong image



15.4.4.2.3.10 Handling Babble Conditions

If receives a packet that is larger than the maximum packet size for that endpoint, the core stops writing data to the Rx buffer and waits for the end of packet (EOP). When the core detects the EOP, it flushes the packet in the Rx buffer and does not send any response to the host.

If the core continues to receive data at the EOF2 (the end of frame 2, which is very close to SOF), the core generates an early_suspend interrupt (USB_GINTSTS.ERLYSUSP). On receiving this interrupt, the application must check the erratic_error status bit (USB_DSTS.ERRTICERR). If this bit is set, the application must take it as a long babble and perform a soft reset.

15.4.4.2.3.11 Generic Non-Periodic (Bulk and Control) IN Data Transfers in DMA and Slave Mode

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193) . Before it can communicate with the host, it must initialize an endpoint as described in Endpoint Initialization (p. 228) . For packet writes in Slave mode, see: Packet Write in Slave Mode (p. 250) .

Application Requirements

1. Before setting up an IN transfer, the application must ensure that all data to be transmitted as part of the IN transfer is part of a single buffer, and must program the size of that buffer and its start address (in DMA mode) to the endpoint-specific registers.
2. For IN transfers, the Transfer Size field in the Endpoint Transfer Size register denotes a payload that constitutes multiple maximum-packet-size packets and a single short packet. This short packet is transmitted at the end of the transfer.

- To transmit a few maximum-packet-size packets and a short packet at the end of the transfer:
- Transfer size[epnum] = n * mps[epnum] + sp

(where n is an integer >= 0, and 0 <= sp < mps[epnum])

- If (sp > 0), then packet count[epnum] = n + 1. Otherwise, packet count[epnum] = n

a. To transmit a single zero-length data packet:
   - Transfer size[epnum] = 0
   - Packet count[epnum] = 1

b. To transmit a few maximum-packet-size packets and a zero-length data packet at the end of the transfer, the application must split the transfer in two parts. The first sends maximum-packet-size data packets and the second sends the zero-length data packet alone.

c. First transfer: transfer size[epnum] = n * mps[epnum]; packet count = n;

d. Second transfer: transfer size[epnum] = 0; packet count = 1;

3. In DMA mode, the core fetches an IN data packet from the memory, always starting at a DWORD boundary. If the maximum packet size of the IN endpoint is not a multiple of 4, the application must arrange the data in the memory with pads inserted at the end of a maximum-packet-size packet so that a new packet always starts on a DWORD boundary.
4. Once an endpoint is enabled for data transfers, the core updates the Transfer Size register. At the end of IN transfer, which ended with a Endpoint Disabled interrupt, the application must read the Transfer Size register to determine how much data posted in the transmit FIFO was already sent on the USB.
5. Data fetched into transmit FIFO = Application-programmed initial transfer size – core-updated final transfer size
   - Data transmitted on USB = (application-programmed initial packet count – Core updated final packet count) * mps[epnum]
   - Data yet to be transmitted on USB = (Application-programmed initial transfer size – data transmitted on USB)

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers and enable the endpoint to transmit the data.
2. In Slave mode, the application must also write the required data to the transmit FIFO for the endpoint. In DMA mode, the core fetches the data from memory according to the application setting for the endpoint.
3. Every time a packet is written into the transmit FIFO, either by the core's internal DMA (in DMA mode) or the application (in Slave Mode), the transfer size for that endpoint is decremented by the packet size. The data is fetched from the memory (DMA/Application), until the transfer size for the endpoint becomes 0. After writing the data into the FIFO, the "number of packets in FIFO" count is incremented (this is a 3-bit count, internally maintained by the core for each IN endpoint transmit FIFO. The maximum number of packets maintained by the core at any time in an IN endpoint FIFO is eight). For zero-length packets, a separate flag is set for each FIFO, without any data in the FIFO.
4. Once the data is written to the transmit FIFO, the core reads it out upon receiving an IN token. For every non-isochronous IN data packet transmitted with an ACK handshake, the packet count for the endpoint is decremented by one, until the packet count is zero. The packet count is not decremented on a TIMEOUT.
5. For zero length packets (indicated by an internal zero length flag), the core sends out a zero-length packet for the IN token and decrements the Packet Count field.
6. If there is no data in the FIFO for a received IN token and the packet count field for that endpoint is zero, the core generates a IN Tkn Rcvd When FIFO Empty Interrupt for the endpoint, provided the endpoint NAK bit is not set. The core responds with a NAK handshake for non-isochronous endpoints on the USB.
7. For Control IN endpoint, if there is a TIMEOUT condition, the USB_DIEPx_INT.TIMEOUT interrupt is generated.
8. When the transfer size is 0 and the packet count is 0, the transfer complete interrupt for the endpoint is generated and the endpoint enable is cleared.

Application Programming Sequence

1. Program the USB_DIEPx_TSIZ register with the transfer size and corresponding packet count. In DMA mode, also program the USB_DIEPx_DMAADDR register.
2. Program the USB_DIEPx_CTL register with the endpoint characteristics and set the CNAK and Endpoint Enable bits.
3. In slave mode when transmitting non-zero length data packet, the application must poll the USB_DIEPx_TXFSTS register (where x is the FIFO number associated with that endpoint) to determine whether there is enough space in the data FIFO. The application can optionally use USB_DIEPx_INT.TXFEMP before writing the data.

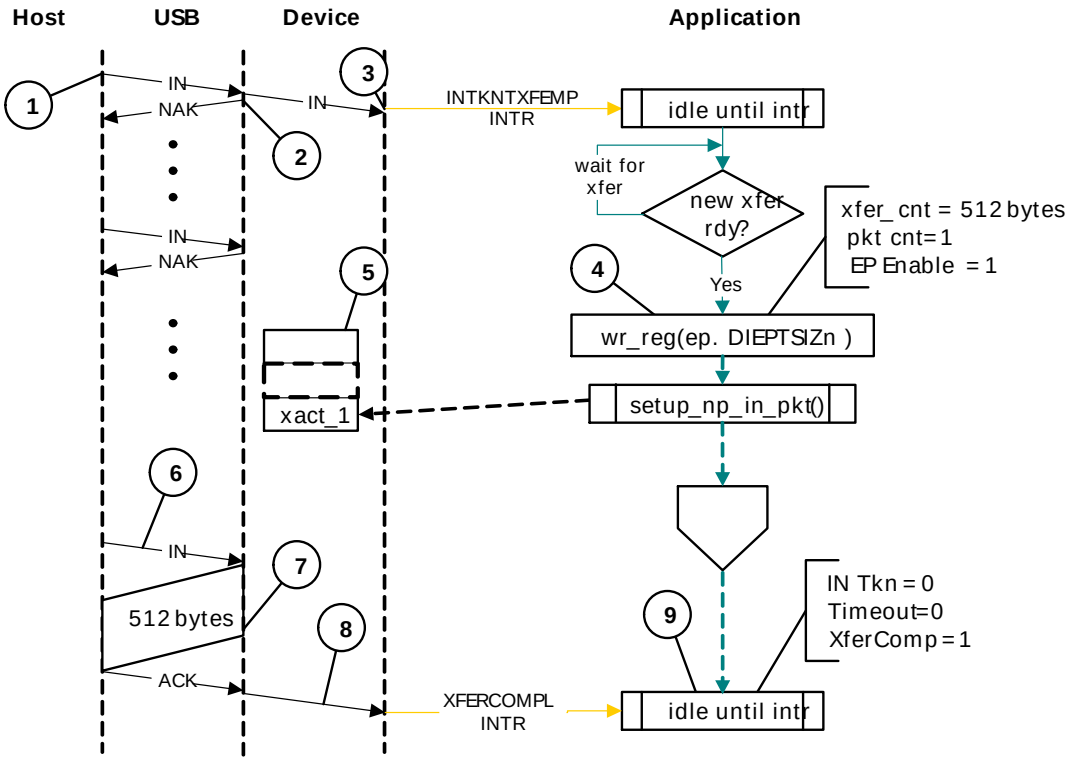
15.4.4.2.3.12 Examples

Slave Mode Bulk IN Transaction

These notes refer to Figure 15.24 (p. 257) .

1. The host attempts to read data (IN token) from an endpoint.
2. On receiving the IN token on the USB, the core returns a NAK handshake, because no data is available in the transmit FIFO.
3. To indicate to the application that there was no data to send, the core generates a USB_DIEPx_INT.INTKNTXFEMP (IN Token Received When TxFIFO Empty) interrupt.
4. When data is ready, the application sets up the USB_DIEPx_TSIZ register with the Transfer Size and Packet Count fields.
5. The application writes one maximum packet size or less of data to the Non-periodic TxFIFO.
6. The host reattempts the IN token.
7. Because data is now ready in the FIFO, the core now responds with the data and the host ACKs it.
8. Because the XFERSIZE is now zero, the intended transfer is complete. The device core generates a USB_DIEPx_INT.XFERCOMPL interrupt.
9. The application processes the interrupt and uses the setting of the USB_DIEPx_INT.XFERCOMPL interrupt bit to determine that the intended transfer is complete.

Figure 15.24. Slave Mode Bulk IN Transaction

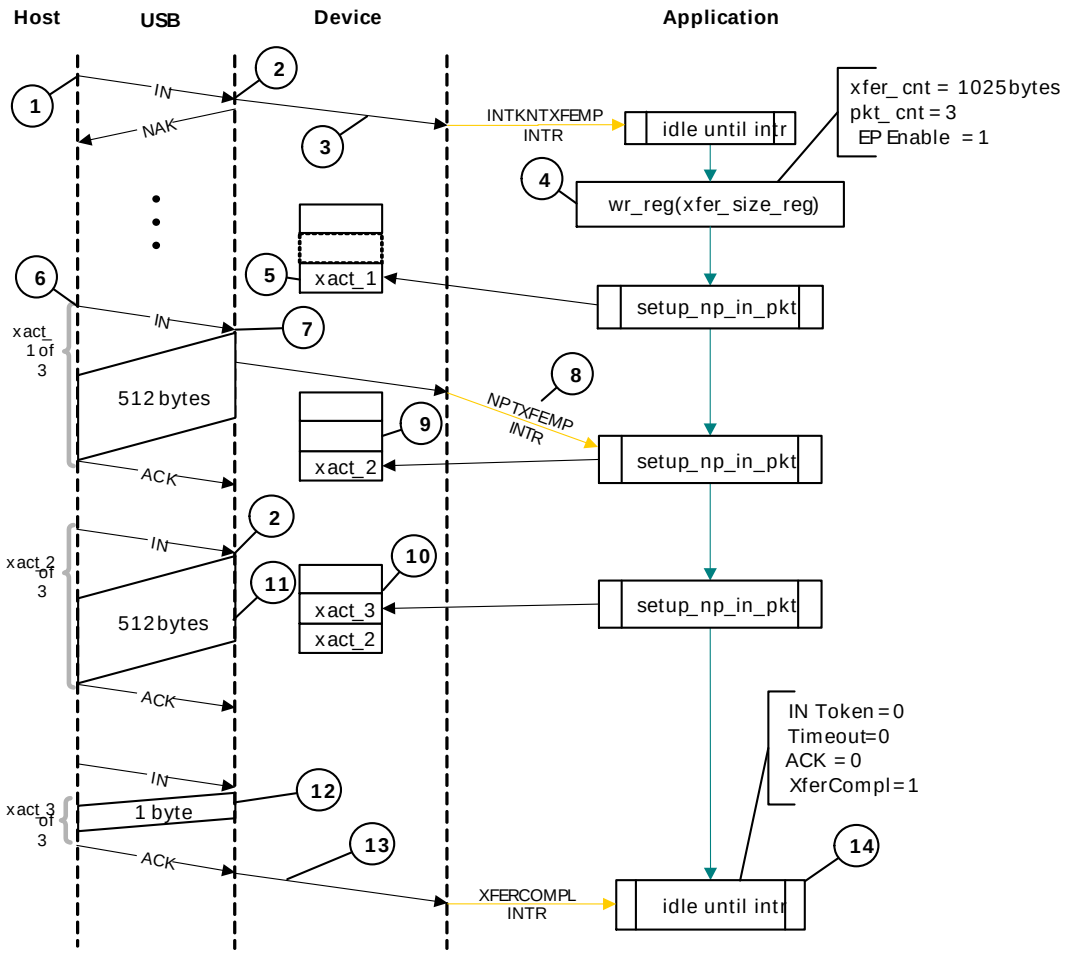


Slave Mode Bulk IN Transfer (Pipelined Transaction)

These notes refer to Figure 15.25 (p. 259)

1. The host attempts to read data (IN token) from an endpoint.
2. On receiving the IN token on the USB, the core returns a NAK handshake, because no data is available in the transmit FIFO.
3. To indicate that there was no data to send, the core generates an USB_DIEPx_INT.INTKNTXFEMP (In Token Received When TxFIFO Empty) interrupt.
4. When data is ready, the application sets up the USB_DIEPx_TSIZ register with the transfer size and packet count.
5. The application writes one maximum packet size or less of data to the Non-periodic TxFIFO.
6. The host reattempts the IN token.
7. Because data is now ready in the FIFO, the core responds with the data, and the host ACKs it.
8. When the TxFIFO level falls below the halfway mark, the core generates a USB_GINTSTS.NPTXFEMP (NonPeriodic TxFIFO Empty) interrupt. This triggers the application to start writing additional data packets to the FIFO.
9. A data packet for the second transaction is ready in the TxFIFO.
10. A data packet for third transaction is ready in the TxFIFO while the data for the second packet is being sent on the bus.
11. The second data packet is sent to the host.
12. The last short packet is sent to the host.
13. Because the last packet is sent and XFERSIZE is now zero, the intended transfer is complete. The core generates a USB_DIEPx_INT.XFERCOMPL interrupt.
14. The application processes the interrupt and uses the setting of the USB_DIEPx_INT.XFERCOMPL interrupt bit to determine that the intended transfer is complete

Figure 15.25. Slave Mode Bulk IN Transfer (Pipelined Transaction)



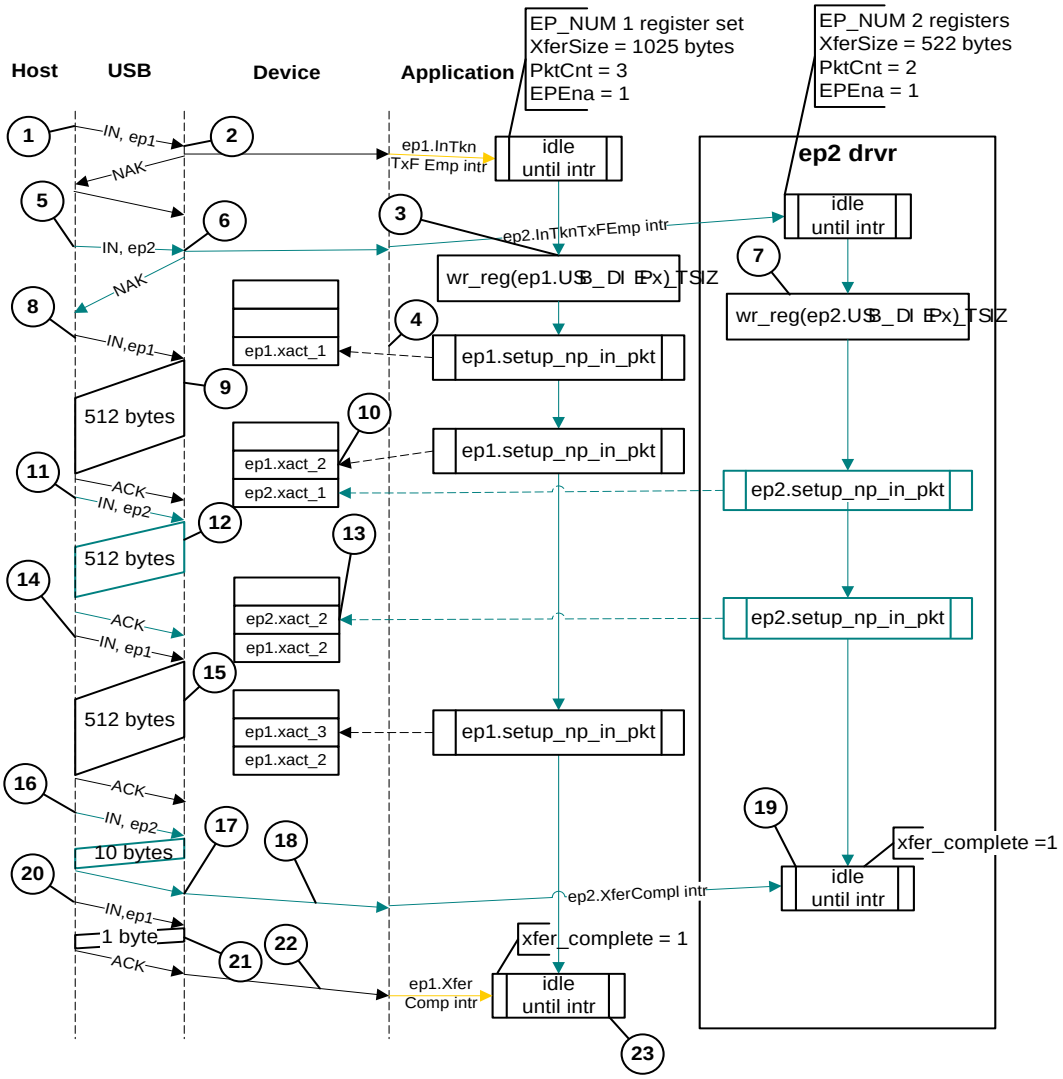
Slave Mode Bulk IN Two-Endpoint Transfer

These notes refer to Figure 15.26 (p. 260)

1. The host attempts to read data (IN token) from endpoint 1.
2. On receiving the IN token on the USB, the core returns a NAK handshake, because no data is available in the transmit FIFO for endpoint 1, and generates a USB_DIEP1_INT.INTKNTXFEMP (In Token Received When TxFIFO Empty) interrupt.
3. The application processes the interrupt and initializes USB_DIEP1_TSIZ register with the Transfer Size and Packet Count fields. The application starts writing the transaction data to the transmit FIFO.
4. The application writes one maximum packet size or less of data for endpoint 1 to the Non-periodic TxFIFO.
5. Meanwhile, the host attempts to read data (IN token) from endpoint 2.
6. On receiving the IN token on the USB, the core returns a NAK handshake, because no data is available in the transmit FIFO for endpoint 2, and the core generates a USB_DIEP2_INT.INTKNTXFEMP (In Token Received When TxFIFO Empty) interrupt.
7. Because the application has completed writing the packet for endpoint 1, it initializes the USB_DIEP2_TSIZ register with the Transfer Size and Packet Count fields. The application starts writing the transaction data into the transmit FIFO for endpoint 2.
8. The host repeats its attempt to read data (IN token) from endpoint 1.
9. Because data is now ready in the TxFIFO, the core returns the data, which the host ACKs.
10. Meanwhile, the application has initialized the data for the next two packets in the TxFIFO (ep2.xact1 and ep1.xact2, in order).
11. The host repeats its attempt to read data (IN token) from endpoint 2.
12. Because endpoint 2's data is ready, the core responds with the data (ep2.xact_1), which the host ACKs.

13. Meanwhile, the application has initialized the data for the next two packets in the TxFIFO (ep2.xact2 and ep1.xact3, in order). The application has finished initializing data for the two endpoints involved in this scenario.
14. The host repeats its attempt to read data (IN token) from endpoint 1.
15. Because data is now ready in the FIFO, the core responds with the data, which the host ACKs.
16. The host repeats its attempt to read data (IN token) from endpoint 2.
17. With data now ready in the FIFO, the core responds with the data, which the host ACKs.
18. With the last packet for endpoint 2 sent and its XFERSIZE now zero, the intended transfer is complete. The core generates a USB_DIEP2_INT.XFERCOMPL interrupt for this endpoint.
19. The application processes the interrupt and uses the setting of the USB_DIEP2_INT.XFERCOMPL interrupt bit to determine that the intended transfer on endpoint 2 is complete.
20. The host repeats its attempt to read data (IN token) from endpoint 1 (last transaction).
21. With data now ready in the FIFO, the core responds with the data, which the host ACKs.
22. Because the last endpoint one packet has been sent and XFERSIZE is now zero, the intended transfer is complete. The core generates a USB_DIEP1_INT.XFERCOMPL interrupt for this endpoint.
23. The application processes the interrupt and uses the setting of the USB_DIEP1_INT.XFERCOMPL interrupt bit to determine that the intended transfer on endpoint 1 is complete.

Figure 15.26. Slave Mode Bulk IN Two-Endpoint Transfer



15.4.4.2.3.13 Generic Periodic IN (Interrupt and Isochronous) Data Transfers

To initialize the core after power-on reset, the application must follow the sequence in Overview: Programming the Core (p. 193). Before it can communicate with the host, it must initialize an endpoint

as described in Endpoint Initialization (p. 228) . For packet writes in Slave mode, see: Packet Write in Slave Mode (p. 250) .

Application Requirements

1. Application requirements 1, 2, 3, and 4 of Generic Non-Periodic (Bulk and Control) IN Data Transfers Without Thresholding in DMA and Slave Mode(p. 255) also apply to periodic IN data transfers, except for a slight modification of Requirement 2.
   - The application can only transmit multiples of maximum-packet-size data packets or multiples of maximum-packet-size packets, plus a short packet at the end. To transmit a few maximum-packet-size packets and a short packet at the end of the transfer, the following conditions must be met.
     - transfer size[epnum] = n * mps[epnum] + sp(where n is an integer # 0, and 0 >= sp < mps[epnum])
     - If (sp > 0), packet count[epnum] = n + 1Otherwise, packet count[epnum] = n;
     - mc[epnum] = packet count[epnum]
   - The application cannot transmit a zero-length data packet at the end of transfer. It can transmit a single zero-length data packet by it self. To transmit a single zero-length data packet,
   - transfer size[epnum] = 0
     - packet count[epnum] = 1
     - mc[epnum] = packet count[epnum]
2. The application can only schedule data transfers 1 frame at a time.
   - (USB_DIEPx_TSIZ.MC – 1) * USB_DIEPx_CTL.MPS =< USB_DIEPx_TSIZ.XFERSIZE =< USB_DIEPx_TSIZ.MC * USB_DIEPx_CTL.MPS
   - USB_DIEPx_TSIZ.PKTCNT = USB_DIEPx_TSIZ.MC
   - If USB_DIEPx_TSIZ.XFERSIZE < USB_DIEPx_TSIZ.MC * USB_DIEPx_CTL.MPS, the last data packet of the transfer is a short packet.
3. This step is not applicable for isochronous data transfers, only for interrupt transfers.

   The application can schedule data transfers for multiple frames, only if multiples of max packet sizes (up to 3 packets), must be transmitted every frame. This is can be done, only when the core is operating in DMA mode. This is not a recommended mode though.
   - ((n*USB_DIEPx_TSIZ.MC) - 1)*USB_DIEPx_CTL.MPS <= USB_DIEPx_TSIZ.XFERSIZE <= n*USB_DIEPx_TSIZ.MC*USB_DIEPx_CTL.MPS
   - USB_DIEPx_TSIZ.PKTCNT = n*USB_DIEPx_TSIZ.MC
   - n is the number of frames for which the data transfers are scheduled

   Data Transmitted per frame in this case would be USB_DIEPx_TSIZ.MC*USB_DIEPx_CTL.MPS, in all the frames except the last one. In the frame “n”, the data transmitted would be (USB_DIEPx_TSIZ.XFERSIZE - (n-1)*USB_DIEPx_TSIZ.MC*USB_DIEPx_CTL.MPS)
4. For Periodic IN endpoints, the data must always be prefetched 1 frame ahead for transmission in the next frame. This can be done, by enabling the Periodic IN endpoint 1 frame ahead of the frame in which the data transfer is scheduled.
5. The complete data to be transmitted in the frame must be written into the transmit FIFO (either by the application or the DMA), before the Periodic IN token is received. Even when 1 DWORD of the data to be transmitted per frame is missing in the transmit FIFO when the Periodic IN token is received, the core behaves as when the FIFO was empty. When the transmit FIFO is empty,
6. A zero data length packet would be transmitted on the USB for ISO IN endpoints
   - A NAK handshake would be transmitted on the USB for INTR IN endpoints
7. For a High Bandwidth IN endpoint with three packets in a frame, the application can program the endpoint FIFO size to be 2*max_pkt_size and have the third packet load in after the first packet has been transmitted on the USB.

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers and enable the endpoint to transmit the data.

2. In Slave mode, the application must also write the required data to the associated transmit FIFO for the endpoint. In DMA mode, the core fetches the data for the endpoint from memory, according to the application setting.
3. Every time either the core's internal DMA (in DMA mode) or the application (in Slave mode) writes a packet to the transmit FIFO, the transfer size for that endpoint is decremented by the packet size. The data is fetched from DMA or application memory until the transfer size for the endpoint becomes 0.
4. When an IN token is received for an periodic endpoint, the core transmits the data in the FIFO, if available. If the complete data payload (complete packet) for the frame is not present in the FIFO, then the core generates an IN Token Received When TxFIFO Empty Interrupt for the endpoint.
   - A zero-length data packet is transmitted on the USB for isochronous IN endpoints
   - A NAK handshake is transmitted on the USB for interrupt IN endpoints
5. The packet count for the endpoint is decremented by 1 under the following conditions:
   - For isochronous endpoints, when a zero- or non-zero-length data packet is transmitted
   - For interrupt endpoints, when an ACK handshake is transmitted
   - When the transfer size and packet count are both 0, the Transfer Completed interrupt for the endpoint is generated and the endpoint enable is cleared.
6. At the "Periodic frame Interval" (controlled by USB_DCFG.PERFRINT), when the core finds non-empty any of the isochronous IN endpoint FIFOs scheduled for the current frame non-empty, the core generates a USB_GINTSTS.INCOMPISOIN interrupt.

Application Programming Sequence (Transfer Per Frame)

1. Program the USB_DIEPx_TSIZ register. In DMA mode, also program the USB_DIEPx_DMAADDR register.
2. Program the USB_DIEPx_CTL register with the endpoint characteristics and set the CNAK and Endpoint Enable bits.
3. In Slave mode, write the data to be transmitted in the next frame to the transmit FIFO.
4. Asserting the USB_DIEPx_INT.INTKNTXFEMP (In Token Received When TxFifo Empty) interrupt indicates that either the DMA or application has not yet written all data to be transmitted to the transmit FIFO.
5. If the interrupt endpoint is already enabled when this interrupt is detected, ignore the interrupt. If it is not enabled, enable the endpoint so that the data can be transmitted on the next IN token attempt.
   - If the isochronous endpoint is already enabled when this interrupt is detected, see Incomplete Isochronous IN Data Transfers (p. 252) for more details.
6. The core handles timeouts internally on interrupt IN endpoints programmed as periodic endpoints without application intervention. The application, thus, never detects a USB_DIEPx_INT.TIMEOUT interrupt for periodic interrupt IN endpoints.
7. Asserting the USB_DIEPx_INT.XFERCOMPL interrupt with no USB_DIEPx_INT.INTKNTXFEMP (In Token Received When TxFifo Empty) interrupt indicates the successful completion of an isochronous IN transfer. A read to the USB_DIEPx_TSIZ register must indicate transfer size = 0 and packet count = 0, indicating all data is transmitted on the USB.
8. Asserting the USB_DIEPx_INT.XFERCOMPL interrupt, with or without the USB_DIEPx_INT.INTKNTXFEMP (In Token Received When TxFifo Empty) interrupt, indicates the successful completion of an interrupt IN transfer. A read to the USB_DIEPx_TSIZ register must indicate transfer size = 0 and packet count = 0, indicating all data is transmitted on the USB.
9. Asserting the USB_GINTSTS.INCOMPISOIN (Incomplete Isochronous IN Transfer) interrupt with none of the aforementioned interrupts indicates the core did not receive at least 1 periodic IN token in the current frame.
10. For isochronous IN endpoints, see Incomplete Isochronous IN Data Transfers (p. 252) , for more details.

15.4.4.2.3.14 Generic Periodic IN Data Transfers Using the Periodic Transfer Interrupt Feature

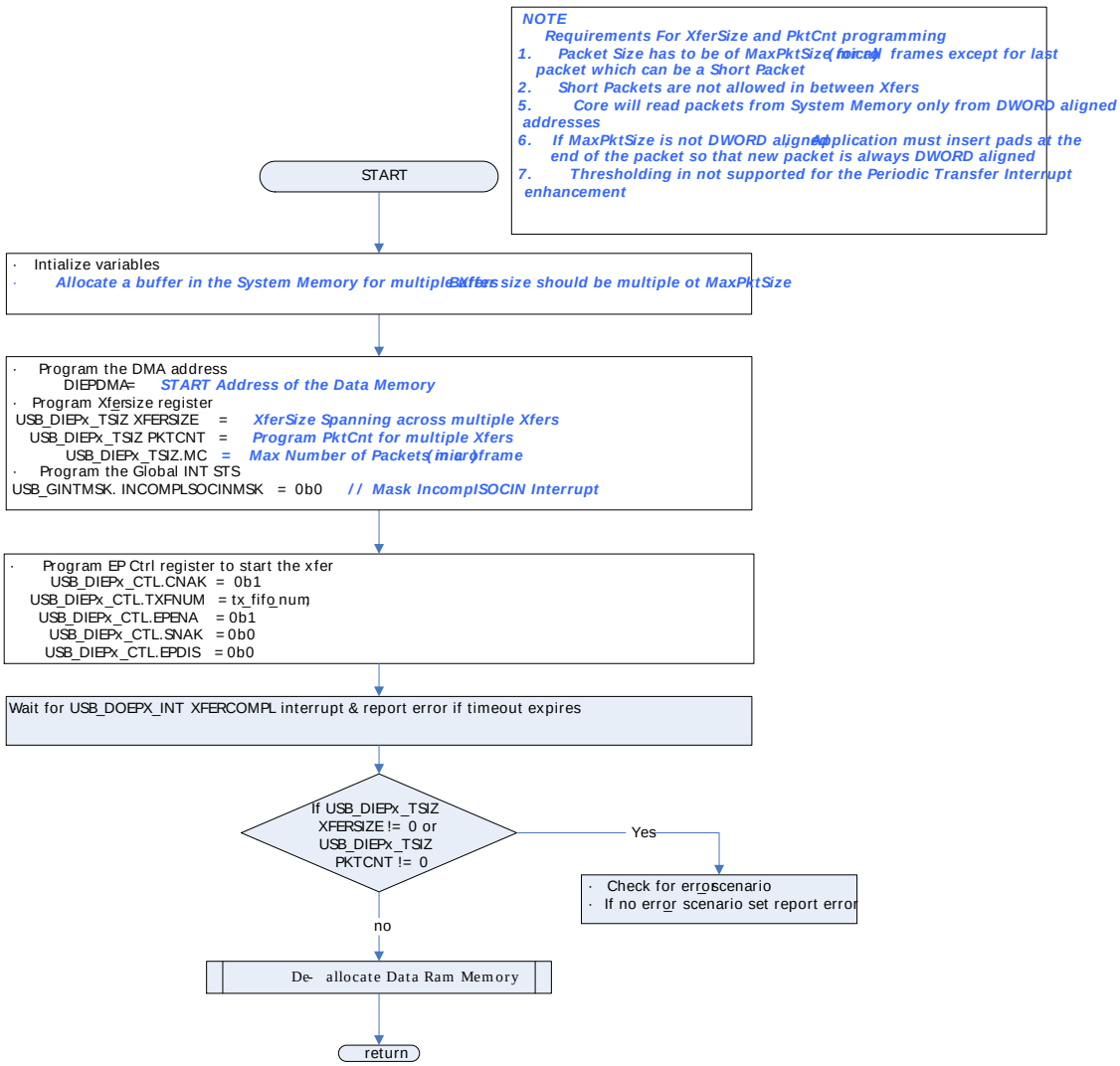
This section describes a typical Periodic IN (ISOC / INTR) data transfer with the Periodic Transfer Interrupt feature.

1. Before setting up an IN transfer, the application must ensure that all data to be transmitted as part of the IN transfer is part of a single buffer, and must program the size of that buffer and its start address (in DMA mode) to the endpoint-specific registers.
2. For IN transfers, the Transfer Size field in the Endpoint Transfer Size register denotes a payload that constitutes multiple maximum-packet-size packets and a single short packet. This short packet is transmitted at the end of the transfer.
   a. To transmit a few maximum-packet-size packets and a short packet at the end of the transfer:
      - Transfer size[epnum] = n * mps[epnum] + sp

      (where n is an integer > 0, and 0 < sp < mps[epnum]. A higher value of n reduces the periodicity of the USB_DOEPx_INT.XFERCOMPL interrupt)
      - If (sp > 0), then packet count[epnum] = n + 1. Otherwise, packet count[epnum] = n
   b. To transmit a single zero-length data packet:
      - Transfer size[epnum] = 0
      - Packet count[epnum] = 1
   c. To transmit a few maximum-packet-size packets and a zero-length data packet at the end of the transfer, the application must split the transfer in two parts. The first sends maximum-packet-size data packets and the second sends the zero-length data packet alone.
      - First transfer: transfer size[epnum] = n * mps[epnum]; packet count = n;
      - Second transfer: transfer size[epnum] = 0; packet count = 1;
   d. The application can only transmit multiples of maximum-packet-size data packets or multiples of maximum-packet-size packets, plus a short packet at the end. To transmit a few maximum-packet-size packets and a short packet at the end of the transfer, the following conditions must be met.
      - transfer size[epnum] = n * mps[epnum] + sp (where n is an integer > 0, and 0 < sp < mps[epnum])
      - If (sp > 0), packet count[epnum] = n + 1 Otherwise, packet count[epnum] = n;
      - mc[epnum] = number of packets to be sent out in a frame.
   e. The application cannot transmit a zero-length data packet at the end of transfer. It can transmit a single zero-length data packet by itself. To transmit a single zero-length data packet,
      - transfer size[epnum] = 0
      - packet count[epnum] = 1
      - mc[epnum] = packet count[epnum]
3. In DMA mode, the core fetches an IN data packet from the memory, always starting at a DWORD boundary. If the maximum packet size of the IN endpoint is not a multiple of 4, the application must arrange the data in the memory with pads inserted at the end of a maximum-packet-size packet so that a new packet always starts on a DWORD boundary.
4. Once an endpoint is enabled for data transfers, the core updates the Transfer Size register. At the end of IN transfer, which ended with a Endpoint Disabled interrupt, the application must read the Transfer Size register to determine how much data posted in the transmit FIFO was already sent on the USB.
   - Data fetched into transmit FIFO = Application-programmed initial transfer size - core-updated final transfer size
   - Data transmitted on USB = (application-programmed initial packet count - Core updated final packet count) * mps[epnum]
   - Data yet to be transmitted on USB = (Application-programmed initial transfer size - data transmitted on USB)
5. The application can schedule data transfers for multiple frames, only if multiples of max packet sizes (up to 3 packets), must be transmitted every frame. This is can be done, only when the core is operating in DMA mode.
   - ((n*USB_DIEPx_TSIZ.MC) - 1)*USB_DIEPx_CTL.MPS <= USB_DIEPx_TSIZ.XFERSIZE <= n*USB_DIEPx_TSIZ.MC*USB_DIEPx_CTL.MPS
   - USB_DIEPx_TSIZ.PKTCNT = n*USB_DIEPx_TSIZ.MC
   - n is the number of frames for which the data transfers are scheduled. Data Transmitted per frame in this case is USB_DIEPx_TSIZ.MC*USB_DIEPx_CTL.MPS in all frames except the last one. In frame n, the data transmitted is (USB_DIEPx_TSIZ.XFERSIZE – (n – 1) * USB_DIEPx_TSIZ.MC * USB_DIEPx_CTL.MPS)

6. For Periodic IN endpoints, the data must always be prefetched 1 frame ahead for transmission in the next frame. This can be done, by enabling the Periodic IN endpoint 1 frame ahead of the frame in which the data transfer is scheduled.
7. The complete data to be transmitted in the frame must be written into the transmit FIFO, before the Periodic IN token is received. Even when 1 DWORD of the data to be transmitted per frame is missing in the transmit FIFO when the Periodic IN token is received, the core behaves as when the FIFO was empty. When the transmit FIFO is empty,
   - A zero data length packet would be transmitted on the USB for ISOC IN endpoints
   - A NAK handshake would be transmitted on the USB for INTR IN endpoints
   - USB_DIEPx_TSIZ.PKTCNT is not decremented in this case.
8. For a High Bandwidth IN endpoint with three packets in a frame, the application can program the endpoint FIFO size to be 2 * max_pkt_size and have the third packet load in after the first packet has been transmitted on the USB.

Figure 15.27. Periodic IN Application Flow for Periodic Transfer Interrupt Feature

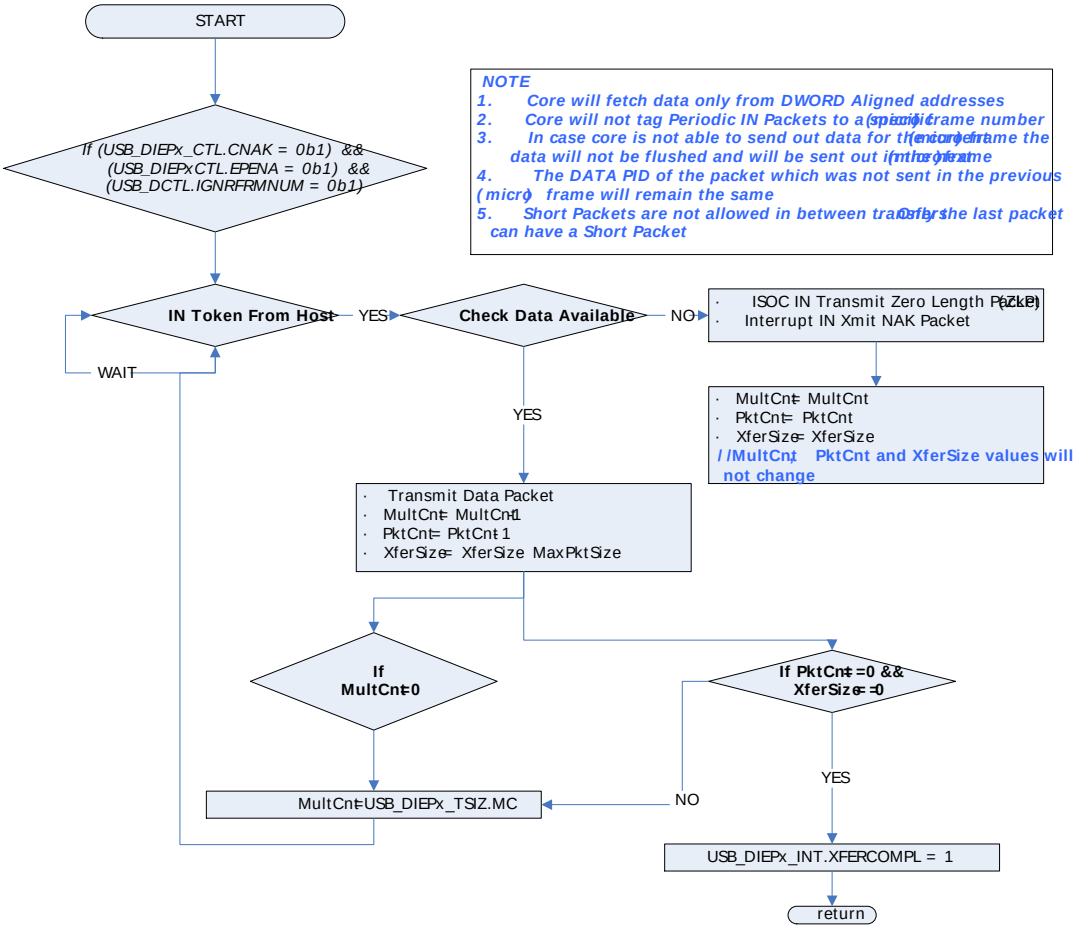


Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the endpoint-specific registers and enable the endpoint to transmit the data.
   - The application must enable the USB_DCTL.IGNRFRMNUM
2. When an isochronous OUT endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame will be ignored by the core.
   - Subsequently the core updates the Even / Odd bit on its own

3. Every time either the core's internal DMA writes a packet to the transmit FIFO, the transfer size for that endpoint is decremented by the packet size. The data is fetched from DMA or application memory until the transfer size for the endpoint becomes 0.
4. When an IN token is received for a periodic endpoint, the core transmits the data in the FIFO, if available. If the complete data payload (complete packet) for the frame is not present in the FIFO, then the core generates an IN Token Received When TxFifo Empty Interrupt for the endpoint.
   - A zero-length data packet is transmitted on the USB for isochronous IN endpoints
   - A NAK handshake is transmitted on the USB for interrupt IN endpoints
5. If an IN token comes for an endpoint on the bus, and if the corresponding TxFIFO for that endpoint has at least 1 packet available, and if the USB_DIEPx_CTL.NAK bit is not set, and if the internally maintained even/odd bit match with the bit 0 of the current frame number, then the core will send this data out on the USB. The core will also decrement the packet count. Core also toggles the MultCount in USB_DIEPx_CTL register and based on the value of MultCount the next PID value is sent.
   - If the IN token results in a timeout (core did not receive the handshake or handshake error), core rewind the FIFO pointers. Core does not decrement packet count. It does not toggle PID. USB_DIEPx_INT.TIMEOUT interrupt will be set which the application could check.
   - At the end of periodic frame interval (Based on the value programmed in the USB_DCFG.PERFRINT register, core will internally set the even/odd internal bit to match the next frame.
6. The packet count for the endpoint is decremented by 1 under the following conditions:
   - For isochronous endpoints, when a zero- or non-zero-length data packet is transmitted
   - For interrupt endpoints, when an ACK handshake is transmitted
7. The data PID of the transmitted data packet is based on the value of USB_DIEPx_TSIZ.MC programmed by the application. In case the USB_DIEPx_TSIZ.MC value is set to 3 then, for a particular frame the core expects to receive 3 Isochronous IN token for the respective endpoint. The data PIDs transmitted will be D2 followed by D1 and D0 respectively for the tokens.
   - If any of the tokens responded with a zero-length packet due to non-availability of data in the TxFIFO, the packet is sent in the next frame with the pending data PID. For example, in a frame, the first received token is responded to with data and data PID value D2. If the second token is responded to with a zero-length packet, the host is expected not to send any more tokens for the respective endpoint in the current frame. When a token arrives in the next frame it will be responded to with the pending data PID value of D1.
   - Similarly the second token of the current frame gets responded with D0 PID. The host is expected to send only two tokens for this frame as the first token got responded with D1 PID.
8. When the transfer size and packet count are both 0, the Transfer Completed interrupt for the endpoint is generated and the endpoint enable is cleared.
9. The USB_GINTSTS.INCOMPISOIN will be masked by the application hence at the Periodic Frame interval (controlled by USB_DCFG.PERFRINT), even though the core finds non-empty any of the isochronous IN endpoint FIFOs, USB_GINTSTS.INCOMPISOIN interrupt will not be generated.

Figure 15.28. Periodic IN Core Internal Flow for Periodic Transfer Interrupt Feature



15.4.5 OTG Revision 1.3 Programming Model

This section describes the OTG programming model when the core is configured to support OTG Revision 1.3 of the specification.

The core is an OTG device supporting HNP and SRP. When the core is connected to an “A” plug, it is referred to as an A-device. When the core is connected to a “B” plug it is referred to as a B-device. In Host mode, the core turns off Vbus to conserve power. SRP is a method by which the B-device signals the A-device to turn on Vbus power. A device must perform both data-line pulsing and Vbus pulsing, but a host can detect either data-line pulsing or Vbus pulsing for SRP. HNP is a method by which the B-device negotiates and switches to host role. In Negotiated mode after HNP, the B-device suspends the bus and reverts to the device role.

15.4.5.1 A-Device Session Request Protocol

The application must set the SRP-Capable bit in the Core USB Configuration register. This enables the core to detect SRP as an A-device.

1. To save power, the application suspends and turns off port power when the bus is idle by writing the Port Suspend and Port Power bits in the Host Port Control and Status register.
2. PHY indicates port power off by detecting that VBUS voltage level is no longer valid.
3. The device must detect SE0 for at least 2 ms to start SRP when Vbus power is off.
4. To initiate SRP, the device turns on its data line pull-up resistor for 5 to 10 ms. The core detects data-line pulsing.
5. The device drives Vbus above the A-device session valid (2.0 V minimum) for Vbus pulsing.

   The core interrupts the application on detecting SRP. The Session Request Detected bit is set in Global Interrupt Status register (USB_GINTSTS.SESSREQINT).

6. The application must service the Session Request Detected interrupt and turn on the Port Power bit by writing the Port Power bit in the Host Port Control and Status register. The PHY indicates port power-on by detecting a valid VBUS level.
7. When the USB is powered, the device connects, completing the SRP process.

15.4.5.2 B-Device Session Request Protocol

The application must set the SRP-Capable bit in the Core USB Configuration register. This enables the core to initiate SRP as a B-device. SRP is a means by which the core can request a new session from the host.

1. To save power, the host suspends and turns off port power when the bus is idle. PHY indicates port power off by detecting a not valid VBUS level.

   The core sets the Early Suspend bit in the Core Interrupt register after 3 ms of bus idleness. Following this, the core sets the USB Suspend bit in the Core Interrupt register.

   The PHY indicates the end of the B-device session by detecting a VBUS level below session valid.
2. PHY to enables the VBUS discharge function to speed up Vbus discharge.
3. The PHY indicates the session's end by detecting a session end voltage level on VBUS. This is the initial condition for SRP. The core requires 2 ms of SE0 before initiating SRP.

   The application must wait until Vbus discharges to 0.2 V after USB_GOTGCTL.BSESVLD is deasserted. This discharge time can be obtained from the datasheet.
4. The application initiates SRP by writing the Session Request bit in the OTG Control and Status register. The core perform data-line pulsing followed by Vbus pulsing.
5. The host detects SRP from either the data-line or Vbus pulsing, and turns on Vbus. The PHY indicates Vbus power-on by detecting a valid VBUS level.
6. The core performs Vbus pulsing.

   The host starts a new session by turning on Vbus, indicating SRP success. The core interrupts the application by setting the Session Request Success Status Change bit in the OTG Interrupt Status register. The application reads the Session Request Success bit in the OTG Control and Status register.
7. When the USB is powered, the core connects, completing the SRP process.

15.4.5.3 A-Device Host Negotiation Protocol

HNP switches the USB host role from the A-device to the B-device. The application must set the HNP-Capable bit in the Core USB Configuration register to enable the core to perform HNP as an A#device.

1. The core sends the B-device a SetFeature b_hnp_enable descriptor to enable HNP support. The B-device's ACK response indicates that the B-device supports HNP. The application must set Host Set HNP Enable bit in the OTG Control and Status register to indicate to the core that the B-device supports HNP.
2. When it has finished using the bus, the application suspends by writing the Port Suspend bit in the Host Port Control and Status register.
3. When the B-device observes a USB suspend, it disconnects, indicating the initial condition for HNP. The B-device initiates HNP only when it must switch to the host role; otherwise, the bus continues to be suspended.

   The core sets the Host Negotiation Detected interrupt in the OTG Interrupt Status register, indicating the start of HNP.

   The PHY turns off the D+ and D- pulldown resistors to indicate a device role. The PHY enable the D + pull-up resistor indicates a connect for B-device.

The application must read the Current Mode bit in the OTG Control and Status register to determine Device mode operation.

4. The B-device detects the connection, issues a USB reset, and enumerates the core for data traffic.
5. The B-device continues the host role, initiating traffic, and suspends the bus when done.

   The core sets the Early Suspend bit in the Core Interrupt register after 3 ms of bus idleness. Following this, the core sets the USB Suspend bit in the Core Interrupt register.
6. In Negotiated mode, the core detects the suspend, disconnects, and switches back to the host role. The core turns on the D+ and D- pulldown resistors to indicate its assumption of the host role.
7. The core sets the Connector ID Status Change interrupt in the OTG Interrupt Status register. The application must read the connector ID status in the OTG Control and Status register to determine the core's operation as an A-device. This indicates the completion of HNP to the application. The application must read the Current Mode bit in the OTG Control and Status register to determine Host mode operation.
8. The B-device connects, completing the HNP process.

15.4.5.4 B-Device Host Negotiation Protocol

HNP switches the USB host role from B-device to A-device. The application must set the HNP-Capable bit in the Core USB Configuration register to enable the core to perform HNP as a B-device.

1. The A-device sends the SetFeature b_hnp_enable descriptor to enable HNP support. The core's ACK response indicates that it supports HNP. The application must set the Device HNP Enable bit in the OTG Control and Status register to indicate HNP support.

   The application sets the HNP Request bit in the OTG Control and Status register to indicate to the core to initiate HNP.
2. When it has finished using the bus, the A-device suspends by writing the Port Suspend bit in the Host Port Control and Status register.

   The core sets the Early Suspend bit in the Core Interrupt register after 3 ms of bus idleness. Following this, the core sets the USB Suspend bit in the Core Interrupt register.

   The core disconnects and the A-device detects SE0 on the bus, indicating HNP. The core enables the D+ and D- pulldown resistors to indicate its assumption of the host role.

   The A-device responds by activating its D+ pull-up resistor within 3 ms of detecting SE0. The core detects this as a connect.

   The core sets the Host Negotiation Success Status Change interrupt in the OTG Interrupt Status register, indicating the HNP status. The application must read the Host Negotiation Success bit in the OTG Control and Status register to determine host negotiation success. The application must read the Current Mode bit in the Core Interrupt register (USB_GINTSTS) to determine Host mode operation.
3. The application sets the reset bit (USB_HPRT.PRTRST) and the core issues a USB reset and enumerates the A-device for data traffic
4. The core continues the host role of initiating traffic, and when done, suspends the bus by writing the Port Suspend bit in the Host Port Control and Status register.
5. In Negotiated mode, when the A-device detects a suspend, it disconnects and switches back to the host role. The core disables the D+ and D- pulldown resistors to indicate the assumption of the device role.
6. The application must read the Current Mode bit in the Core Interrupt (USB_GINTSTS) register to determine the Host mode operation.
7. The core connects, completing the HNP process.

15.4.6 OTG Revision 2.0 Programming Model

OTG Revision 2.0 supports the new Attach Detection Protocol (ADP). This protocol enables a local device (an OTG device or Embedded Host) to detect when a remote device is attached or detached.

Note
ADP is not supported by the core.

In addition to ADP, OTG Revision 2.0 also supports enhanced SRP and HNP, which are described in the following sections:

- OTG Revision 2.0 Session Request Protocol (p. 269)
- OTG Revision 2.0 Host Negotiation Protocol (p. 271)

Note
VBUS pulsing is not supported in OTG Revision 2.0 mode.

15.4.6.1 OTG Revision 2.0 Session Request Protocol

When the core is behaving as an A-device, it can power off VBUS when no session is active until the B-device initiates a SRP. The SRP detection is handled by the core.

Figure 15.29 (p. 270) illustrates the programming steps that need to be performed by A-device's application (core as A-device) when B-device initiates a SRP to establish a connection.

Figure 15.29. SRP Detection by Core When Operating as A-device

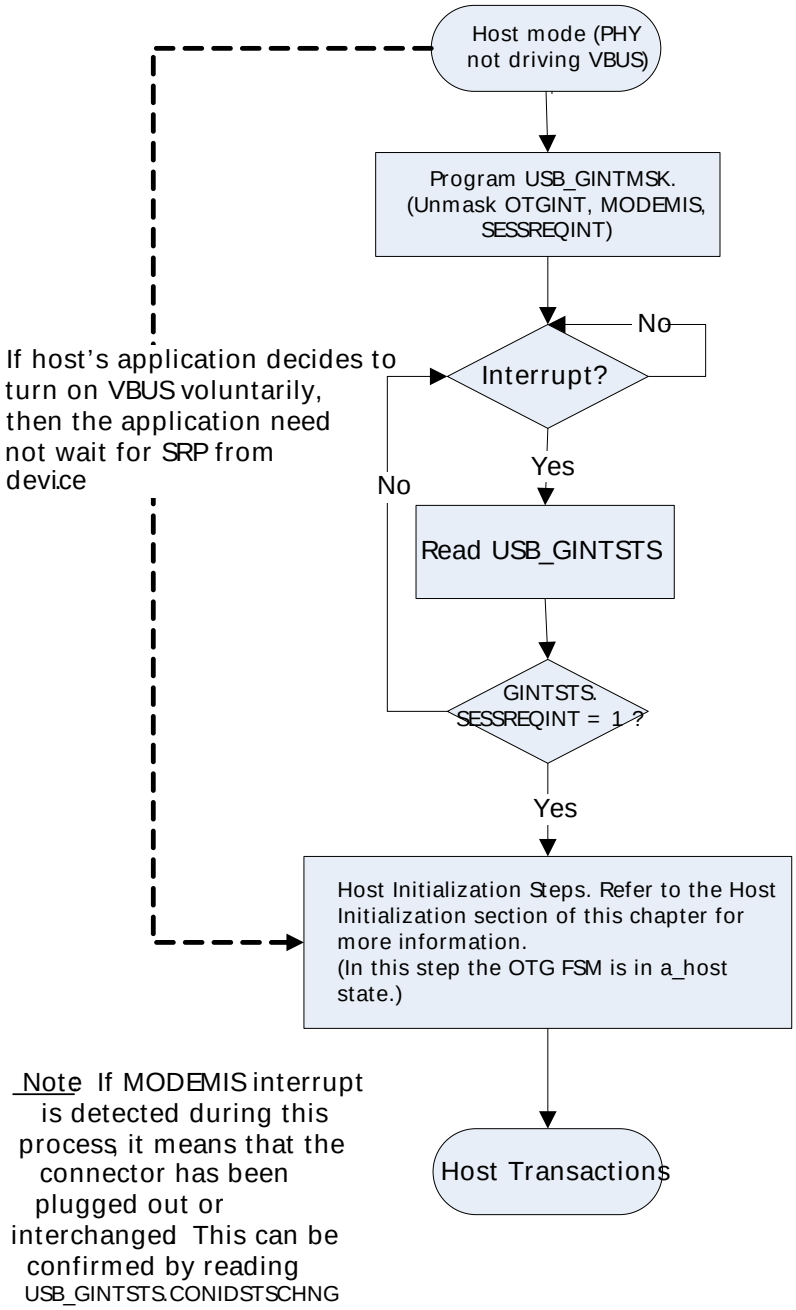
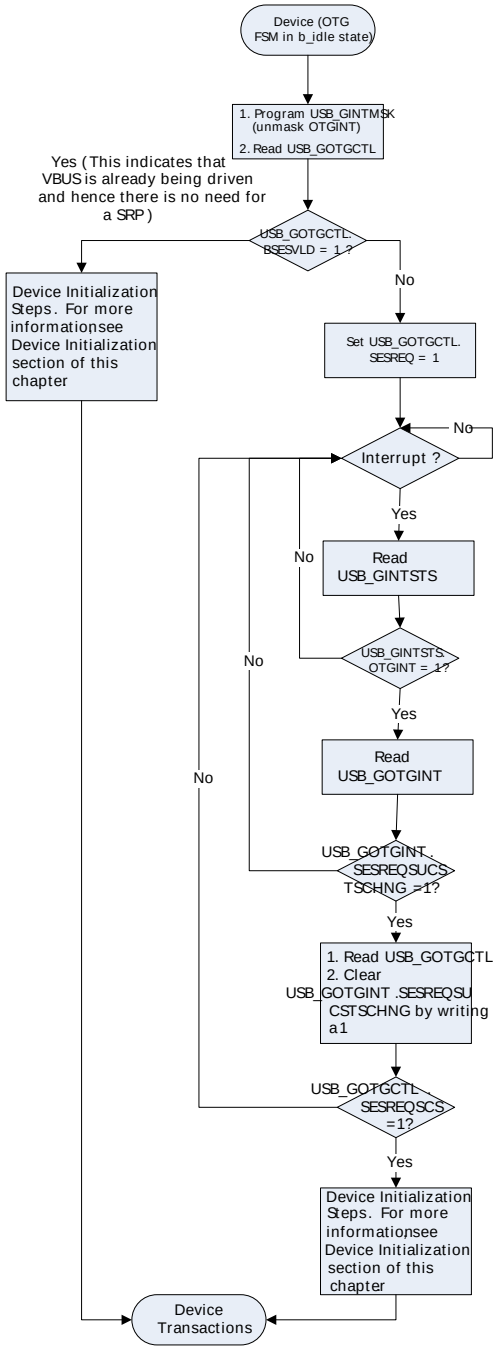


Figure 15.30 (p. 271) illustrates the steps that need to be performed by B-device's application (core as B-device) in order to establishing a connection with A-device by signaling a SRP.

Figure 15.30. SRP Initiation by the Core When Acting as a B-Device



Note

The programming flow illustrated in Figure 15.30 (p. 271) is similar to OTG revision 1.3. This is because the presence or absence of VBUS pulsing is transparent to the application.

15.4.6.2 OTG Revision 2.0 Host Negotiation Protocol

When the core is operating as A-device, the application must execute a GetStatus() operation to the B-device with a frequency of THOST_REQ_POLL to determine the state of the host request flag in the B-device. If the host request flag is set in B-device it must program the core to change its role within THOST_REQ_SUSP.

Figure 15.31 (p. 272) shows the programming steps that need to be performed by A-device's application (core as A-device) in order to change its role to device. In Figure 15.31 (p. 272) , the A-device performs a role change, becomes a B-device and then reverts back to host (A-device) mode of operation.

Figure 15.31. HNP When the Core is an A-Device

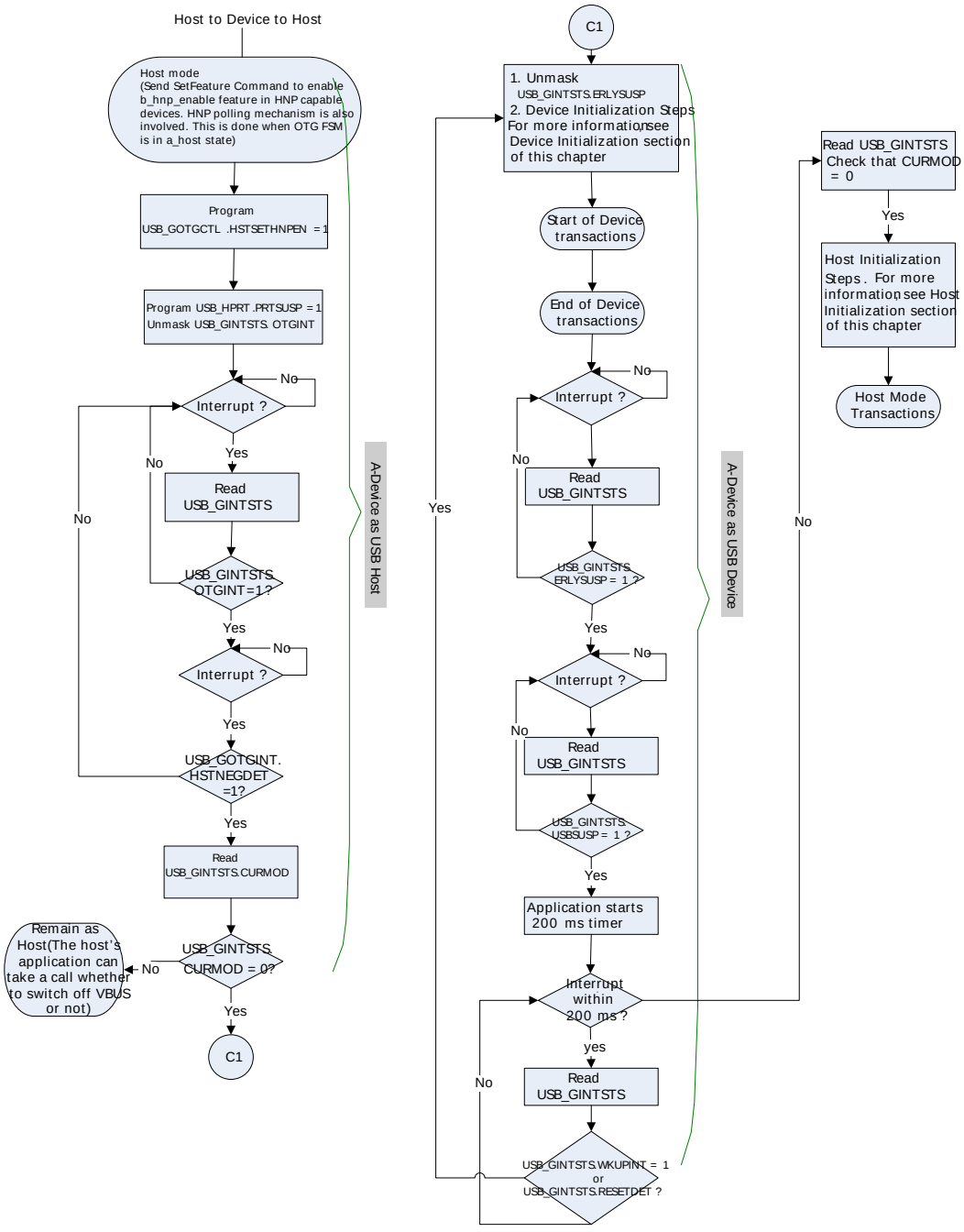
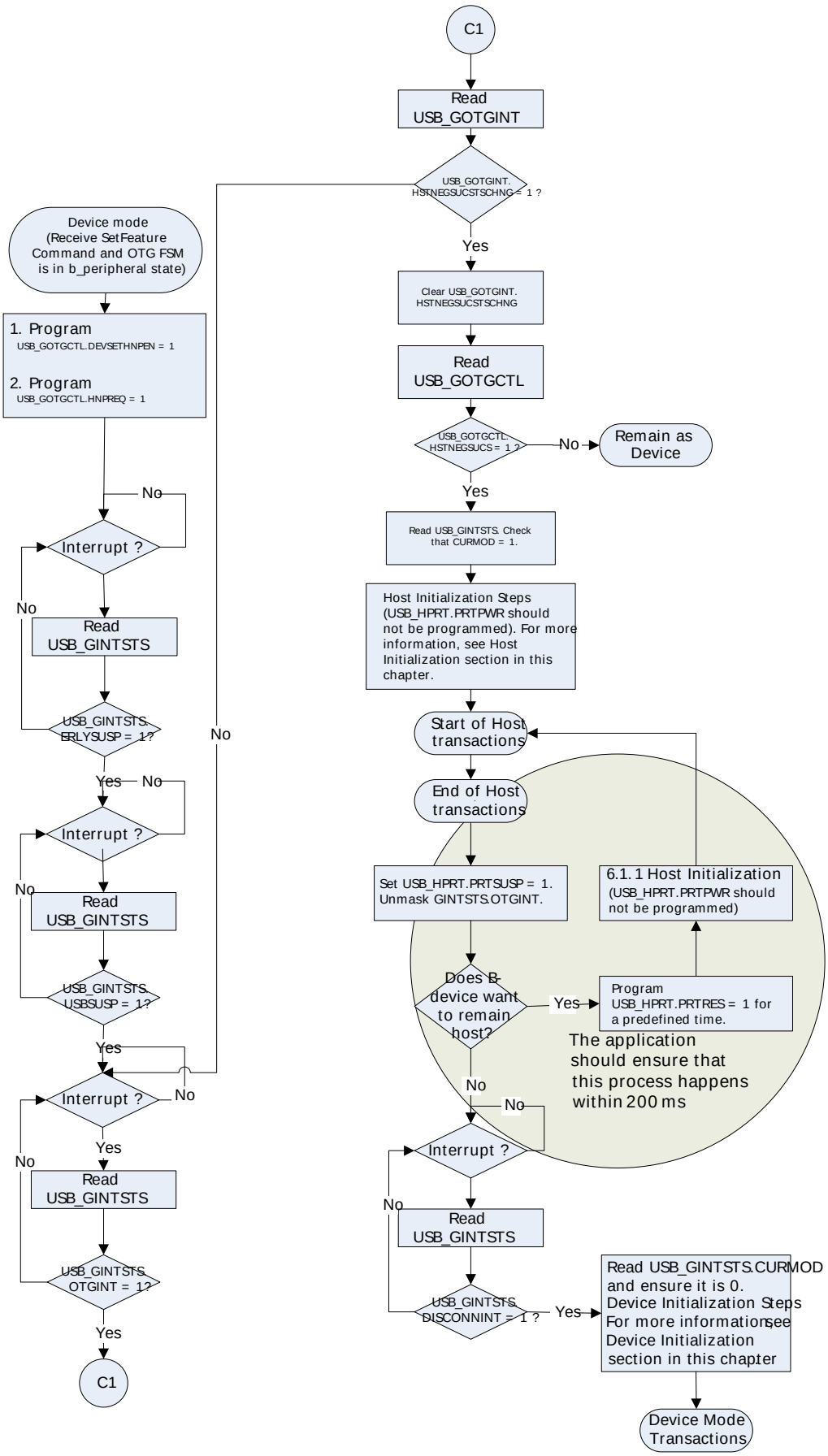


Figure 15.32 (p. 273) shows the programming steps that need to be performed by B-device's application (core as B-device) in order to change its role to Host. In Figure 15.32 (p. 273) , the B-device performs a role change, becomes a Host and then reverts back to Device mode of operation.

Figure 15.32. HNP When the Core is a B-Device



Note

During HNP process where the B-device is going to assume the role of a host, the B-device application needs to ensure that a USB reset process is programmed (in USB_HPRT

register) within 150 ms (TB_ACON_BSE0) of getting a USB_HPRT.PRTCONNDET interrupt.

15.4.7 FIFO RAM Allocation

15.4.7.1 Data FIFO RAM Allocation

External RAM must be allocated among different FIFOs in the core before any transactions can start. The application must follow this procedure every time it changes core FIFO RAM allocation.

The application must allocate data RAM per FIFO based on the AHB's operating frequency, the PHY Clock frequency, the available AHB bandwidth, and the performance required on the USB. Based on the above mentioned criteria, the application must provide a table as described below with RAM sizes for each FIFO in each mode.

The core shares a single FIFO RAM between transmit FIFO(s) and receive FIFO.

In DMA mode—The FIFO RAM is also used for storing the some register information.

The Device mode Endpoint DMA address registers (USB_DIEP0DMAADDR, USB_DOEP0DMAADDR, USB_DIEPx_DMAADDR, USB_DOEPx_DMAADDR) and Host mode Channel DMA registers (USB_HCx_DMAADDR) are stored in the FIFO RAM.

- These register information are stored at the end of the FIFO RAM after the space allocated for receive and Transmit FIFO. These register space must also be taken into account when calculating the total FIFO depth of the core as explained in the following sections.

The registers USB_DIEPx_DMAADDR/USB_DOEPx_DMAADDR are maintained in RAM.

The following rules apply while calculating how much RAM space must be allocated to store these registers.

Host Mode:

- Slave mode only: No space needed.
- DMA mode: One location per channel.

Device Mode:

- Slave mode only: No space needed.
- DMA mode: One location per end point direction.

15.4.7.1.1 Device Mode

15.4.7.1.1.1 Tx FIFO Operation

When allocating data RAM for FIFOs in Device mode keep in mind these factors:

1. Receive FIFO RAM allocation:
   - RAM for SETUP Packets: 4 * n + 6 locations must be Reserved in the receive FIFO to receive up to n SETUP packets on control endpoints, where n is the number of control endpoints the device core supports. The core does not use these locations, which are Reserved for SETUP packets, to write any other data.
   - One location for Global OUT NAK
   - Status information is written to the FIFO along with each received packet. Therefore, a minimum space of (Largest Packet Size / 4) + 1 must be allotted to receive packets. If a high-bandwidth endpoint is enabled, or multiple isochronous endpoints are enabled, then at least two (Largest

Packet Size / 4) + 1 spaces must be allotted to receive back-to-back packets. Typically, two (Largest Packet Size / 4) + 1 spaces are recommended so that when the previous packet is being transferred to AHB, the USB can receive the subsequent packet. If AHB latency is high, you must allocate enough space to receive multiple packets. This is critical to prevent dropping any isochronous packets.

- Along with each endpoint's last packet, transfer complete status information is also pushed to the FIFO. Typically, one location for each OUT endpoint is recommended.

2. Transmit FIFO RAM Allocation:

   The minimum RAM space required for each IN Endpoint Transmit FIFO is the maximum packet size for that particular IN endpoint.

   More space allocated in the transmit IN Endpoint FIFO results in a better performance on the USB and can hide latencies on the AHB.

Table 15.3.

FIFO Name	Data RAM Size
Receive data FIFO	rx_fifo_size. This must include RAM for setup packets, OUT endpoint control information and data OUT packets, as mentioned earlier.
Transmit FIFO 0	tx_fifo_size[0]
Transmit FIFO 1	tx_fifo_size[1]
Transmit FIFO 2	tx_fifo_size[2]
...	...
Transmit FIFO i	tx_fifo_size[i]

With this information, the following registers must be programmed as follows:

1. Receive FIFO Size Register (USB_GRXFSIZ)

   USB_GRXFSIZ.Receive FIFO Depth = rx_fifo_size;
2. Device IN Endpoint Transmit FIFO0 Size Register (USB_GNPTXFSIZ)

   USB_GNPTXFSIZ.non-periodic Transmit FIFO Depth = tx_fifo_size[0];

   USB_GNPTXFSIZ.non-periodic Transmit RAM Start Address = rx_fifo_size;
3. Device IN Endpoint Transmit FIFO#1 Size Register (USB_DIEPTXF1)

   USB_DIEPTXF1. Transmit RAM Start Address = USB_GNPTXFSIZ.FIFO0 Transmit RAM Start Address + tx_fifo_size[0];
4. Device IN Endpoint Transmit FIFO#2 Size Register (USB_DIEPTXF2)

   USB_DIEPTXF2.Transmit RAM Start Address = USB_DIEPTXF1.Transmit RAM Start Address + tx_fifo_size[1];
5. Device IN Endpoint Transmit FIFO#i Size Register (USB_DIEPTXFi)

   USB_DIEPTXFm.Transmit RAM Start Address = USB_DIEPTXFi-1.Transmit RAM Start Address + tx_fifo_size[i-1];
6. The transmit FIFOs and receive FIFO must be flushed after the RAM allocation is done, for the proper functioning of the FIFOs.
   - USB_GRSTCTL.TXFNUM = 0x10
   - USB_GRSTCTL.TXFFLSH = 1
   - USB_GRSTCTL.RXFFLSH = 1

The application must wait until the TXFFLSH bit and the RXFFLSH bits are cleared before performing any operation on the core.

15.4.7.1.2 Host Mode

Considerations for allocating data RAM for Host Mode FIFOs are listed here:

Receive FIFO RAM allocation:

Status information is written to the FIFO along with each received packet. Therefore, a minimum space of (Largest Packet Size / 4) + 2 must be allotted to receive packets. If a high-bandwidth channel is enabled, or multiple isochronous channels are enabled, then at least two (Largest Packet Size / 4) + 2 spaces must be allotted to receive back-to-back packets. Typically, two (Largest Packet Size / 4) + 2 spaces are recommended so that when the previous packet is being transferred to AHB, the USB can receive the subsequent packet. If AHB latency is high, you must allocate enough space to receive multiple packets.

Along with each host channel's last packet, information on transfer complete status and channel halted is also pushed to the FIFO. So two locations must be allocated for this.

For handling NAK in DMA mode, the application must determine the number of Control/Bulk OUT endpoint data that must fit into the TX_FIFO at the same instant. Based on this, one location each is required for Control/Bulk OUT endpoints.

For example, when the host addresses one Control OUT endpoint and three Bulk OUT endpoints, and all these must fit into the non-periodic TX_FIFO at the same time, then four extra locations are required in the RX FIFO to store the rewind status information for each of these endpoints.

Transmit FIFO RAM allocation

The minimum amount of RAM required for the Host Non-periodic Transmit FIFO is the largest maximum packet size among all supported non-periodic OUT channels.

More space allocated in the Transmit Non-periodic FIFO results in better performance on the USB and can hide AHB latencies. Typically, two Largest Packet Sizes' worth of space is recommended, so that when the current packet is under transfer to the USB, the AHB can get the next packet. If the AHB latency is large, then you must allocate enough space to buffer multiple packets.

The minimum amount of RAM required for Host periodic Transmit FIFO is the largest maximum packet size among all supported periodic OUT channels. If there is at lease one High Bandwidth Isochronous OUT endpoint, then the space must be at least two times the maximum packet size of that channel.

15.4.7.1.2.1 Internal Register Storage Space Allocation

When operating in DMA mode, the DMA address register for each host channel (USB_HCx_DMAADDR) is stored in the FIFO RAM. One location for each channel must be reserved for this.

Table 15.4.

FIFO Name	Data RAM Size
Receive Data FIFO	rx_fifo_size
Non-periodic Transmit FIFO	tx_fifo_size[0]
IN Endpoint Transmit FIFO	tx_fifo_size[1]

With this information, the following registers must be programmed:

1. Receive FIFO Size Register (USB_GRXFSIZ)
   - USB_GRXFSIZ.RXFDEP = rx_fifo_size;

2. Non-periodic Transmit FIFO Size Register (USB_GNPTXFSIZ)
   - USB_GNPTXFSIZ.NPTXFDEP = tx_fifo_size[0];
   - USB_GNPTXFSIZ.NPTXFSTADDR = rx_fifo_size;
3. Host Periodic Transmit FIFO Size Register (USB_HPTXFSIZ)
   - USB_HPTXFSIZ.PTXFSIZE = tx_fifo_size[1];
   - USB_HPTXFSIZ.PTXFSTADDR = USB_GNPTXFSIZ.NPTXFSTADDR + tx_fifo_size[0];
4. The transmit FIFOs and receive FIFO must be flushed after RAM allocation for proper FIFO function.
   - USB_GRSTCTL.TXFNUM = 0x10
   - USB_GRSTCTL.TXFFLSH = 1
   - USB_GRSTCTL.RXFFLSH = 1
   - The application must wait until the TXFFLSH bit and the RXFFLSH bits are cleared before performing any operation on the core.

15.4.7.1.3 Summary of Guidelines for Choosing Data FIFO RAM Depth in Host Mode

15.4.7.1.3.1 RX FIFO size

The RX FIFO size must be equal to at least twice the largest value of MPS size used. The recommended minimum RXFIFO depth = ((largest packet size/4)*2)+2. (+2) is required by the core for the status quadlets internally.

15.4.7.1.3.2 Non periodic TX FIFO size

This should be equal to at least twice the largest value of MPS size used. The recommended minimum non-periodic TXFIFO depth = ((largest packet size/4)*2).

15.4.7.1.3.3 Periodic TX FIFO size

The recommended size for Periodic TXFIFO is sum total of (MPS*MC)/4 for all the channels.

Note

Note: In the above recommendations, always round off the MPS value to the nearest multiple of 4. For example, if the largest value of MPS=125, use the rounded-off value, which is 128.

15.4.7.1.4 Calculating the Total FIFO Size

The RxFIFO is shared between the host and device. The Host TxFIFOs are also shared with Device IN endpoint TxFIFOs 0 through n.

There are three ways to calculate the total FIFO size.

Method 1

Use this method if you are using the following conditions:

- Minimum FIFO depth allocation
- The FIFO must equal at least one MaxPacketSize (MPS).

Device RxFIFO =

- (4 * number of control endpoints + 6) + ((largest USB packet used / 4) + 1 for status information) + (2 * number of OUT endpoints) + 1 for Global NAK

Note

Include the Control OUT endpoint in the number of OUT endpoints.

Host RxFIFO =

- Slave mode

  Minimum requirement: (largest USB packet used / 4) + 1 for status information + 1 transfer complete
- DMA mode

  (largest USB packet used / 4) + 1 for status information + 1 transfer complete + 1 location each bulk/ control out endpoint for handling NAK scenario

Host Non-Periodic TxFIFO =

- largest non-periodic USB packet used / 4

Host Periodic TxFIFO =

- Sum total of (MPS*MC)/4 of all periodic channels or 1500 locations, whichever is lower.

Device IN Endpoint TxFIFOs (a separate FIFO is allocated to each IN endpoint) =

- IN Endpoints Max packet Size / 4

Method 2

Use this method if you are using the recommended minimum FIFO depth allocation with support for high-bandwidth endpoints. This FIFO allocation enables the core to transfer a packet on the USB while the previous (next) packet is simultaneously transferred to the AHB. This FIFO allocation improves the core's performance.

Device RxFIFO =

- (4 * number of control endpoints + 6) + 2 * ((largest USB packet used / 4) + 1) +(2 * number of OUT endpoints) + 1

Host RxFIFO =

- Slave mode

  2 * ((largest USB packet used / 4) + 1 + 1)
- DMA mode

  2 * ((largest USB packet used / 4) + 1 + 1) + 1 location each bulk/control out endpoint for handling NAK scenario

Host Non-Periodic TxFIFO =

- 2 * (largest non-periodic USB packet used / 4)

Host Periodic TxFIFO =

- Sum total of (MPS*MC)/4 for all periodic channels or 1500 location, whichever is lower.

Device IN Endpoint-Specific TxFIFOs (a separate FIFO is allocated to each endpoint) =

- 2 * (max_pkt_size for the endpoint) / 4.

```
//DMA mode

      OTG Total RAM = (Device RxFIFO or Host RxFIFO; choose the largest one) +
```

```
      ((Host Non-Periodic TxFIFO + Host peiodic TxFIFO) or
       (Device IN Endpoint TxFIFO #0 + #1 + #2 + #n)); choose the largest one +
      (1 location per Host channel or 1 location per Device Endpoint direction; choose
       the largest one)

//Slave mode

    OTG Total RAM = (Device RxFIFO or Host RxFIFO; choose the largest one) +
    ((Host Non-Periodic TxFIFO + Host peiodic TxFIFO) or
     (Device IN Endpoint TxFIFO #0 + #1 + #2 + #n)); choose the largest one
```

Method 3

Use this method if you are using the recommended FIFO depth allocation that supports high-bandwidth endpoints and high AHB latency.

Note

- x = (AHB latency + time to transfer largest packet on AHB) / time to transfer largest packet on USB.
- The value of x is an integer. Any fractional value is rounded to the nearest integer. For example: x = 20 ms / 17,039 ms = 1.17 ms = 2 ms.

Device RxFIFO =

- (4 * number of control endpoints + 6) + (x + 1) * ((largest USB packet used / 4) + 1)+ (2 * number of OUT endpoints) + 1

Note

Include the Control OUT endpoint in the number of OUT endpoints.

Host RxFIFO =

- Slave mode

  (x + 1) * ((largest USB packet used / 4) + 1 + 1)
- DMA mode

  (x + 1) * ((largest USB packet used / 4) + 1 + 1) + 1 location each bulk/control out endpoint for handling NAK scenario

Host Non-Periodic TxFIFO =

- (x + 1) * (largest non-periodic USB packet used / 4)

Host Periodic TxFIFO =

- (x+1) * (Sum total of (MPS*MC)/4 of all periodic channels or 1500 locations, whichever is lower).

Device IN Endpoint-Specific TxFIFOs (a separate FIFO is allocated to each endpoint) =

- (x+1)*(max_pkt_size for the endpoint)/4

```
//DMA mode
      OTG Total RAM = (Device RxFIFO or Host RxFIFO; choose the largest one) +
      ((Host Non-Periodic TxFIFO + Host periodic TxFIFO) OR
       (Device IN Endpoint TxFIFO #0 + #1 + #2 + #n); choose the largest one) +
      (1 location per Host channel or 1 location per Device Endpoint direction; choose
```

```
      the largest one)

//Slave mode
    OTG Total RAM = (Device RxFIFO or Host RxFIFO; choose the largest one) +
    ((Host Non-Periodic TxFIFO + Host periodic TxFIFO) OR
     (Device IN Endpoint TxFIFO #0 + #1 + #2 + #n); choose the largest one)
```

15.4.7.2 Dynamic FIFO Allocation

The application can change the RAM allocation for each FIFO during the operation of the core.

15.4.7.2.1 Host Mode

In Host mode, before changing FIFO data RAM allocation, the application must determine the following.

All channels are disabled

- All FIFOs are empty

Once these conditions are met, the application can reallocate FIFO data RAM as explained in Data FIFO RAM Allocation (p. 274) .

After reallocating the FIFO data RAM, the application must flush all FIFOs in the core using the USB_GRSTCTL.TXFFLSH (TxFIFO Flush) and USB_GRSTCTL.RXFFLSH (RxFIFO Flush) fields. Flushing is required to reset the pointers in the FIFOs for proper FIFO operation after reallocation. For more information on flushing FIFOs, see Flushing TxFIFOs in the Core(p. 280) and Flushing RxFIFOs in the Core (p. 281) .

15.4.7.2.2 Device Mode

In Device mode, before changing FIFO data RAM allocation, the application must determine the following.

- All IN and OUT endpoints are disabled
- NAK mode is enabled in the core on all IN endpoints
- Global OUT NAK mode is enabled in the core
- All FIFOs are empty

Once these conditions are met, the application can reallocate FIFO data RAM as explained in Data FIFO RAM Allocation (p. 274) . When NAK mode is enabled in the core, the core responds with a NAK handshake on all tokens received on the USB, except for SETUP packets.

After the reallocating the FIFO data RAM, the application must flush all FIFOs in the core using the USB_GRSTCTL.TXFFLSH (TxFIFO Flush) and USB_GRSTCTL.RXFFLSH (RxFIFO Flush) fields. Flushing is required to reset the pointers in the FIFOs for proper FIFO operation after reallocation. For more information on flushing FIFOs, see Flushing TxFIFOs in the Core(p. 280) and Flushing RxFIFOs in the Core (p. 281) .

15.4.7.2.3 Flushing TxFIFOs in the Core

The application can flush all TxFIFOs in the core using USB_GRSTCTL.TXFFLSH as follows:

1. Check that USB_GINTSTS.GINNAKEFF=0. If this bit is cleared then set USB_DCTL.SGNPINNAK=1.
2. Wait for USB_GINTSTS.GINNAKEFF=1, which indicates the NAK setting has taken effect to all IN endpoints.
3. Poll USB_GRSTCTL.AHBIDLE until it is 1.

   AHBIdle = H indicates that the core is not writing anything to the FIFO.

4. Check that USB_GRSTCTL.TXFFLSH =0. If it is 0, then write the TxFIFO number you want to flush to USB_GRSTCTL.TXFNUM.
5. Set USB_GRSTCTL.TXFFLSH=1and wait for it to clear.
6. Set the USB_DCTL.GCNPINNAK bit.

15.4.7.2.4 Flushing RxFIFOs in the Core

The application can flush all RxFIFOs in the core using USB_GRSTCTL.RXFFLSH as follows:

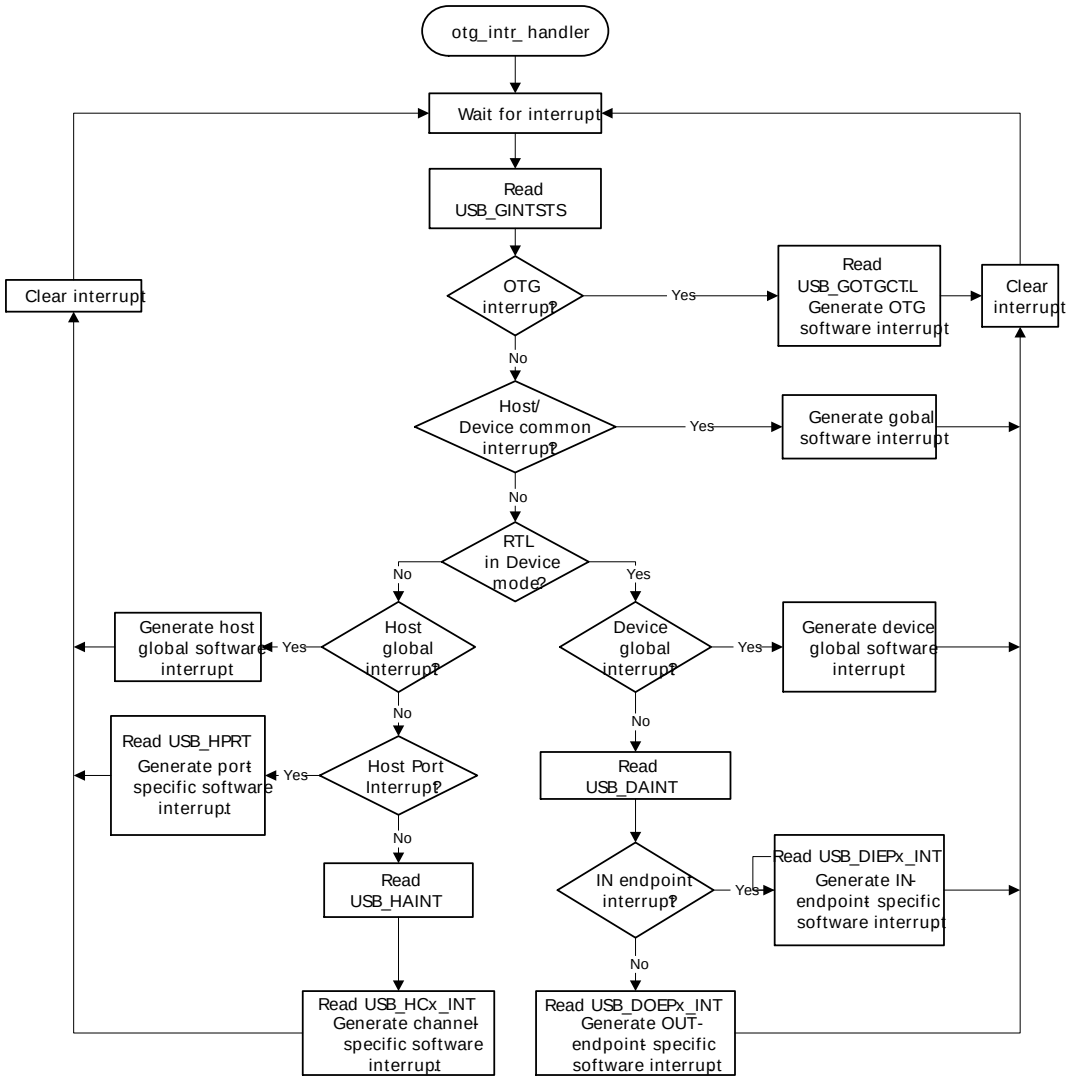
1. Check the status of the USB_GINTSTS.GOUTNAKEFF bit. If it has been cleared, then set USB_DCTL.SGOUTNAK=1. Else, clear USB_GINTSTS.GOUTNAKEFF.

   NAK Effective interrupt = 1 indicates that the core is not writing to FIFO.
2. Wait for USB_GINTSTS.GOUTNAKEFF=1, which indicates the NAK setting has taken effect to all OUT endpoints.
3. Poll the USB_GRSTCTL.AHBIDLE until it is 1.

   AHBIDLE = 1 indicates that the core is not reading anything from the FIFO.
4. Set USB_GRSTCTL.RXFFLSH=1 and wait for it to clear.
5. Set the USB_DCTL.GCOUTNAK bit.

The Core Interrupt Handler

Figure 15.33. Core Interrupt Handler



15.4.8 Suspend/Resume and SRP

This chapter describes different methods of saving power when the USB is suspended. This chapter discusses the following topics:

- Placing PHY in Low Power Mode Without Entering Suspend (p. 282)
  - When the Core is in Host Mode (p. 282)
  - When the Core is in Device Mode (p. 283)
- Suspend (p. 283)
  - Using EM2 (p. 283)
    - Overview of the EM2 Programming Model (p. 283)
    - Using EM2 when the Core is in Host Mode (p. 283)
    - EM2 when the Core is in Device Mode (p. 286)
- Clock Gating (EM0 and EM1) (p. 288)
  - Internal Clock Gating when the Core is in Host Mode (p. 288)
  - Internal Clock Gating when the Core is in Device Mode (p. 289)

15.4.8.1 Placing PHY in Low Power Mode Without Entering Suspend

The core can place the PHY in low power mode (the differential receiver is disabled) without entering suspend.

15.4.8.1.1 When the Core is in Host Mode

Programming flow for the Host Core to put PHY in low power mode

1. To turn off port power, perform write operation to set the following bits in the USB_HPRT register:
   - USB_HPRT.PRTPWR = 0;
   - USB_HPRT.PRTENA = 0;
2. To put PHY in low power mode, perform read-modify-write operation to set the following bits in the USB_PCGCCTL register:
   - USB_PCGCCTL.STOPPCLK = 1
   - USB_PCGCCTL.GATEHCLK = 0

Programming flow for the Host Core to make PHY exit low power mode

If your device is non-SRP capable, the host must implement polling to detect the device connection by turning on the port and exiting PHY low power mode periodically and checking for connect.

1. To turn on port power, perform write operation to set the following bits in the USB_HPRT register:
   - USB_HPRT.PRTPWR = 1
   - USB_HPRT.PRTENA = 0
2. To exit PHY low power mode, perform read-modify-write operation to set the following bits in the USB_PCGCCTL register:
   - USB_PCGCCTL.STOPPCLK = 0
   - USB_PCGCCTL.STOPHCLK = 0
3. Wait for the USB_HPRT Port Connect Detected (PRTCONNDET) bit to be set and do the enumeration of the device.

If your device is SRP-capable, when the device initiates SRP request, the Host core asynchronously detects SRP and the PHY exits low power mode.

1. Wait for Session Request from the device, or New Session Detected Interrupt (SESSREQINT) in the USB_GINTSTS register.
2. To turn on port power, perform write operation to set the following bits in the USB_HPRT register:

- USB_HPRT.PRTPWR = 1
- USB_HPRT.PRTENA = 0

3. Wait for the USB_HPRT Port Connect Detected (PRTCONNDET) bit to be set and do the enumeration of Device.

15.4.8.1.2 When the Core is in Device Mode

To make PHY enter low power mode, complete the following steps:

1. Ensure that the following signals are set as follows:
   - VBUS voltage level must be below the session valid level (VBUS is not active)
   - DP/DM must be SE0
2. From the application, perform read-modify-write operation to set USB_PCGCCTL.STOPPCLK = 1.

15.4.8.2 Suspend

When the core is in Suspend, the following power conservation options are available to use:

- Using EM2 (p. 283) : You can enter EM2, turning off power (and reseting) parts of the core
- Clock Gating (EM0 and EM1) (p. 288) : You can choose gate the AHB clock to some parts of the core Internal Clock Gating when the Core is in Host Mode (p. 288)

This section discusses methods of conserving power by using one of the above methods.

15.4.8.2.1 Using EM2

15.4.8.2.1.1 Overview of the EM2 Programming Model

When the USB is suspended or the session is not valid, the PHY is driven into Suspend mode, stopping the PHY clock to reduce power consumption in the PHY and the core. To further reduce power consumption, the core also supports AHB clock gating and using EM2.

The following sections show the procedures you must follow to use EM2 while in suspend/session-off.

During EM2, the clock to the core must be switched to one of the 32 kHz sources (LFRCO or LFXO). This core needs this clock to detect Resume and SRP events.

15.4.8.2.1.2 EM2 when the Core is in Host Mode

Host Mode Suspend in EM2

Sequence of operations:

1. Back up the essential registers of the core. Read and store the following core registers:

   - USB_GINTMSK
   - USB_GOTGCTL
   - USB_GAHBCFG
   - USB_GUSBCFG
   - USB_GRXFSIZ
   - USB_GNPTXFSIZ
   - USB_DCFG
   - USB_DCTL
   - USB_DAINTMSK
   - USB_DIEPMSK
   - USB_DOEPMSK
   - USB_DIEPx_CTL
   - USB_DIEPx_TSIZ
   - USB_DIEPx_DMAADDR
   - USB_PCGCCTL
   - USB_DIEPTXFn

2. The application sets the Port Suspend bit in the Host Port CSR, and the core drives a USB suspend.

3. The application sets the Power Clamp bit in the Power and Clock Gating Control register.
4. The application sets the Reset to Power-Down Modules bit in the Power and Clock Gating Control register.
5. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the core suspends the PHY and the PHY clock stops. If USB_HCFG.ENA32KHZS is set, switch the USBC clock to 32 kHz.
6. Enter EM2.

Host Mode Resume in EM2

Sequence of operations:

1. The resume event starts by the application waking up from EM2 (on an interrupt)
2. Switch USBC clock back to 48 MHz.
3. The application clears the Stop PHY Clock bit and the core takes the PHY back to normal mode. The PHY clock starts up.
4. The application clears the Power Clamp bit. The core starts driving Resume signaling on the USB.
5. The application clears the Reset to Power-Down Modules bit.
6. The application programs registers in the CSR and sets the Port Resume bit in Host Port CSR (Setting the Port Resume bit is required by the core, although Resume signaling starts earlier).
7. The application clears the Port Resume bit and the core stops driving Resume signaling.

The core is in normal operating mode.

Note

The application must insert delays of at least 2 PHY clocks between all steps in this sequence. This requirement applies to all USB EM2 programming sequences.

Host Mode Remote Wakeup in EM2

Sequence of operations:

1. The core detects Remote Wakeup signaling on the USB. The PHY exits suspend mode and the PHY clock restarts.
2. The core generates a Remote Wakeup Detected interrupt. The Remote Wakeup interrupt is generated using the 32 kHz clock depending on the USB_HCFG.RESVALID (ResumeValidPeriod) programmed. The Host Core starts resume signaling at this stage.
3. The USBC clock is switched back to normal 48 MHz clock.
4. The application clears the Stop PHY Clock bit.
5. The application clears the Power Clamp bit.
6. The application clears the Reset to Power-Down Modules bit
7. The application programs CSRs and sets the Port Resume bit. The core continues to drive Resume signaling on the USB.
8. The application clears the Port Resume bit and the core stops driving Resume signaling.

The core enters normal operating mode.

Host Mode Session End in EM2

Sequence of operations:

1. Back up the essential registers of the core. Read and store the following core registers:

   - USB_GINTMSK
   - USB_GOTGCTL
   - USB_DCTL
   - USB_DAINTMSK

- USB_GAHBCFG
- USB_GUSBCFG
- USB_GRXFSIZ
- USB_GNPTXFSIZ
- USB_DCFG
- USB_DIEPMSK
- USB_DOEPMSK
- USB_DIEPx_CTL
- USB_DIEPx_TSIZ
- USB_DIEPx_DMAADDR
- USB_PCGCCTL
- USB_DIEPTXFn

2. The application sets the Port Suspend bit in the Host Port CSR and the core drives a USB suspend.
3. The application clears the Port Power bit.
4. The application sets the Power Clamp bit in the Power and Clock Gating Control register, and the core clamps the signals between the internal modules on different power rails.
5. The application sets the Reset to Power-Down Modules bit in the Power and Clock Gating Control register.
6. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, and the core suspends the PHY, stopping the PHY clock.
7. Switch USBC clock to 32 kHz.
8. Enter EM2.

Host Mode Session Start (EM2 -> EM0)

Sequence of operations:

1. Exit EM2/Enter EM0).
2. Switch USBC clock back to 48 MHz.
3. The application clears the Stop PHY Clock bit.
4. The application clears the Power Clamp bit. The application clears the Reset to Power-Down Modules bit.
5. The application programs CSRs and sets the Port Power bit to turn on VBUS.
6. The core detects the connection and drives the USB reset.

The core enters normal operating mode.

Host Mode Session End (EM0 -> EM2)

Sequence of operations:

1. Back up the essential registers of the core. Read and store the following core registers:

- USB_GINTMSK
- USB_GOTGCTL
- USB_GAHBCFG
- USB_GUSBCFG
- USB_GRXFSIZ
- USB_GNPTXFSIZ
- USB_DCFG
- USB_DCTL
- USB_DAINTMSK
- USB_DIEPMSK
- USB_DOEPMSK
- USB_DIEPx_CTL
- USB_DIEPx_TSIZ
- USB_DIEPx_DMAADDR
- USB_PCGCCTL
- USB_DIEPTXFn

2. The application sets the Port Suspend bit in the Host Port CSR and the core drives a USB suspend.
3. The application clears the Port Power bit.
4. The application sets the Power Clamp bit in the Power and Clock Gating Control register, and the core clamps the signals between the internal modules on different power rails.
5. The application sets the Reset to Power-Down Modules bit in the Power and Clock Gating Control register.

6. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register.
7. Enter EM2.

Host Mode Sessions Start (SRP) (EM2 -> EM0)

Sequence of operations:

1. The core detects SRP (data line pulsing) on the bus. The core de-asserts the suspend_n signal to the PHY, generating the PHY clock. The SRP Detected interrupt is generated.
2. The application clears the Stop PHY Clock bit, the core deasserts the suspend_n signal to the PHY to generate the PHY clock.
3. The power (VDD_DN) is turned on and stabilizes.
4. The application clears the Power Clamp bit.
5. The application clears the Reset to Power-Down Modules bit.
6. The application programs the CSRs, and sets the Port Power bit to turn on VBUS.
7. The core detects device connection and drives a USB reset.

The core enters normal operating mode.

15.4.8.2.1.3 EM2 when the Core is in Device Mode

Device Mode Suspend With EM2

In Device mode, the device validates the host-driven Resume signal for a period of 1.5 µs (75 clock cycles at 48 MHz). With a 32-KHz clock, 2.34 ms is required (75 clock cycles at 32 KHz) to detect the resume. Hence, the application programs USB_DCFG.RESVALID with a value of 4 clock cycles (125 µs). If the core is in Suspend mode, the device thus detects the resume and the host signals a resume for a minimum of 125 µs.

If the device is being reset from suspend, it begins a high-speed detection handshake after detecting SE0 for no fewer than 2.5 µs. With a 48-MHz clock, detection occurs after 120 clock cycles (2.5 µs). With a 32-kHz clock, 120 clock cycles signifies 3.75 msec. Hence, a programmable value of 4 clock cycles (125 µs) is used to detect reset.

The 32-KHz Suspend feature incorporates switching to the 32-KHz clock during suspend and resume/ remote wakeup until the system comes up and starts driving 48 MHz.

Sequence of operations:

1. Detect Suspend state. Wait for an interrupt from the device core and check that USB_GINTSTS.USBSUSP is set to 1.
2. Back up the essential registers of the core. Read and store the following core registers:

   - USB_GINTMSK
   - USB_GOTGCTL
   - USB_GAHBCFG
   - USB_GUSBCFG
   - USB_GRXFSIZ
   - USB_GNPTXFSIZ
   - USB_DCFG
   - USB_DCTL
   - USB_DAINTMSK
   - USB_DIEPMSK
   - USB_DOEPMSK
   - USB_DIEPx_CTL
   - USB_DIEPx_TSIZ
   - USB_DIEPx_DMAADDR
   - USB_PCGCCTL
   - USB_DIEPTXFn
3. The application sets the PWRCLMP bit in the Power and Clock Gating Control (USB_PCGCCTL) register.
4. The application sets the USB_PCGCCTL.RSTPDWNMODULE bit.

5. The application sets the USB_PCGCCTL.STOPPCLK bit.
6. Switch USB Core Clock (USBC) to 32 kHz.
7. Enter EM2.

Device Mode Resume (EM2 -> EM0)

Sequence if operations:

1. The core detects Resume signaling on the USB. The core generates a Resume Detected interrupt.
2. Switch USB Core Clock (USBC) back to 48 MHz.
3. The application clears the STOPPCLK bit.
4. The application clears the USB_PCGCCTL.PWRCLMP and USB_PCGCCTL.RSTPDWNMODULE bits.
5. Restore the USB_GUSBCFG and USB_DCFG registers with the values stored during the Save operation before entering EM2.
6. Restore the following core registers with the values stored during the Save operation before entering EM2:

   - USB_GINTMSK
   - USB_GOTGCTL
   - USB_GUSBCFG
   - USB_GRXFSIZ
   - USB_GNPTXFSIZ
   - USB_DAINTMSK
   - USB_DIEPMSK
   - USB_DOEPMSK
   - USB_DIEPx_CTL
   - USB_DIEPx_TSIZ
   - USB_DIEPx_DMAADDR
   - USB_DIEPTXFn

7. The application programs CSRs, then sets the Power-On Programming Done bit in the Device Control register.

Device Mode Remote Wakeup (EM2 -> EM0)

Sequence if operations:

1. An interrupt wakes up the device from EM2.
2. Switch USB Core Clock (USBC) back to 48 MHz.
3. The application clears the STOPPCLK and GATEHCLK bits in the USB_PCGCCTL register.
4. The application clears the USB_PCGCCTL.PWRCLMP and USB_PCGCCTL.RSTPDWNMODULE bits.
5. Restore the USB_GUSBCFG and USB_DCFG registers with the values stored during the Save operation before entering EM2 .
6. Drive remote wakeup from the core. Program USB_DCTL by performing write-only operation with the following values:
   - USB_DCTL.RMTWKUPSIG = 1
   - Other Bits = Value stored during the Save operation before entering EM2
7. Clear all interrupt status. Wait for at least 1 millisecond of remote wakeup time and then program GINSTS register with 0xFFFFFFFF to clear all the status register fields.
8. Restore the following core registers with the values stored during the Save operation before entering EM2:

   - USB_GINTMSK
   - USB_GOTGCTL
   - USB_GUSBCFG
   - USB_GRXFSIZ
   - USB_GNPTXFSIZ
   - USB_DAINTMSK
   - USB_DIEPMSK
   - USB_DOEPMSK
   - USB_DIEPx_CTL
   - USB_DIEPx_TSIZ
   - USB_DIEPx_DMAADDR
   - USB_DIEPTXFn

9. Wait for remote wakeup time (1-15ms) and then program USB_DCTL by performing read-modify-write to set USB_DCTL.RMTWKUPSIG = 0.

Device Mode Session End (EM0 -> EM2)

Sequence of operations:

1. The core detects a USB suspend and generates a Suspend Detected interrupt. The host turns off VBUS.
2. The application sets the Power Clamp bit in the Power and Clock Gating Control register.
3. The application sets the Reset to Power-Down Modules bit in the Power and Clock Gating Control register.
4. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register.
5. Switch USB Core clock (USBC) to 32 kHz.
6. Enter EM2.

Device Mode Session Start (EM2 -> EM0)

Sequence of operations:

1. The core detects VBUS on (voltage level within session-valid). A New Session Detected interrupt is generated.
2. Switch USB Core clock (USBC) back to 48 MHz.
3. The application clears the Stop PHY Clock bit.
4. The application clears the Power Clamp bit.
5. The application clears the Reset to Power-Down Modules bit.
6. The application programs CSRs.
7. The cores detects a USB reset.

The core enters normal operating mode.

15.4.8.2.2 Using Clock Gating in EM0/EM1

The core supports HCLK gating to reduce dynamic power to internal modules to the core during Suspend/ session-off state in EM0 and EM1.

15.4.8.2.2.1 Internal Clock Gating when the Core is in Host Mode

The following sections show the procedures you must follow to use the clock gating feature.

Host Mode Suspend and Resume With Clock Gating

Sequence of operations:

1. The application sets the Port Suspend bit in the Host Port CSR, and the core drives a USB suspend.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register. The application sets the Gate hclk bit in the Power and Clock Gating Control register, the core gates the hclk internally.
3. The core remains in Suspend mode.
4. The application clears the Gate hclk and Stop PHY Clock bits, and the PHY clock is generated.
5. The application sets the Port Resume bit, and the core starts driving Resume signaling.
6. The application clears the Port Resume bit after at least 20 ms.
7. The core is in normal operating mode.

Host Mode Suspend and Remote Wakeup With Clock Gating

Sequence of operations:

1. The application sets the Port Suspend bit in the Host Port CSR, and the core drives a USB suspend.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates hclk internally.
3. The core remains in Suspend mode
4. The Remote Wakeup signaling from the device is detected. The core generates a Remote Wakeup Detected interrupt.
5. The application clears the Gate hclk and Stop PHY Clock bits. The core sets the Port Resume bit.
6. The application clears the Port Resume bit after at least 20 ms.
7. The core is in normal operating mode.

Host Mode Session End and Start With Clock Gating

Sequence of operations:

1. The application sets the Port Suspend bit in the Host Port CSR, and the core drives a USB suspend.
2. The application clears the Port Power bit. The core turns off VBUS.
3. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates hclk internally.
4. The core remains in Low-Power mode.
5. The application clears the Gate hclk bit and the application clears the Stop PHY Clock bit to start the PHY clock.
6. The application sets the Port Power bit to turn on VBUS.
7. The core detects device connection and drives a USB reset.
8. The core is in normal operating mode.

Host Mode Session End and SRP With Clock Gating

Sequence of operations:

1. The application sets the Port Suspend bit in the Host Port CSR, and the core drives a USB suspend.
2. The application clears the Port Power bit. The core turns off VBUS.
3. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates hclk internally.
4. The core remains in Low-Power mode.
5. SRP (data line pulsing) from the device is detected. An SRP Request Detected interrupt is generated.
6. The application clears the Gate hclk bit and the Stop PHY Clock bit.
7. The core sets the Port Power bit to turn on VBUS.
8. The core detects device connection and drives a USB reset.
9. The core is in normal operating mode.

15.4.8.2.2.2 Internal Clock Gating when the Core is in Device Mode

The following sections show the procedures you must follow to use the clock gating feature.

Device Mode Suspend and Resume With Clock Gating

Sequence of operations:

1. The core detects a USB suspend and generates a Suspend Detected interrupt.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates hclk.

3. The core remains in Suspend mode.
4. The Resume signaling from the host is detected. A Resume Detected interrupt is generated.
5. The application clears the Gate hclk bit and the Stop PHY Clock bit.
6. The host finishes Resume signaling.
7. The core is in normal operating mode.

Device Mode Suspend and Remote Wakeup With Clock Gating

Sequence of operations:

1. The core detects a USB suspend and generates a Suspend Detected interrupt.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register. The application sets the Gate hclk bit in the Power and Clock Gating Control register, the core gates hclk.
3. The core remains in Suspend mode.
4. The application clears the Gate hclk bit and the Stop PHY Clock bit.
5. The application sets the Remote Wakeup bit in the Device Control register, the core starts driving Remote Wakeup signaling.
6. The host drives Resume signaling.
7. The core is in normal operating mode.

Device Mode Session End and Start With Clock Gating

Sequence of operations:

1. The core detects a USB suspend, and generates a Suspend Detected interrupt. The host turns off VBUS.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates hclk.
3. The core remains in Low-Power mode.
4. The new session is detected (A session-valid voltage is detected). A New Session Detected interrupt is generated.
5. The application clears the Gate hclk and Stop PHY Clock bits.
6. The core detects USB reset.
7. The core is in normal operating mode

Device Mode Session End and SRP With Clock Gating

Sequence of operations:

1. The core detects a USB suspend, and generates a Suspend Detected interrupt. The host turns off VBUS.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the core gates hclk.
3. The core remains in Low-Power mode.
4. The application clears the Gate hclk and Stop PHY Clock bits.
5. The application sets the SRP Request bit, and the core drives data line and VBUS pulsing.
6. The host turns on VBUS, detects device connection, and drives a USB reset.
7. The core is in normal operating mode.

15.4.9 Register Usage

Only the Core Global, Power and Clock Gating, Data FIFO Access, and Host Port registers can be accessed in both Host and Device modes. When the core is operating in one mode, either Device or

Host, the application must not access registers from the other mode. If an illegal access occurs, a Mode Mismatch interrupt is generated and reflected in the Core Interrupt register (USB_GINTSTS.MODEMIS).

When the core switches from one mode to another, the registers in the new mode must be reprogrammed as they would be after a power-on reset.

The memory map for the core is as follows:

- Core Global Registers are located in the address offset-range [0x3C000, 0x3C3FF] and typically start with first letter G.
- Host Mode Registers are located in the address offset-range [0x3C400, 0x3C7FF] and start with first letter H.
- Device Mode Registers are located in the address offset-range [0x3C800, 0x3CDFF] and start with first letter D.
- The Power and Clock Gating register is located at offset 0x3CE00.
- The Device EP/Host Channel FIFOs start at address offset 0x3D000 with 4K spacing. These registers, available in both Host and Device modes, are used to read or write the FIFO space for a specific endpoint or a channel, in a given direction. If a host channel is of type IN, the FIFO can only be read on the channel. Similarly, if a host channel is of type OUT, the FIFO can only be written on the channel.
- The Direct RAM Access area start at address offset 0x5C000.

15.5 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	USB_CTRL	RW	System Control Register
0x004	USB_STATUS	R	System Status Register
0x008	USB_IF	R	Interrupt Flag Register
0x00C	USB_IFS	W1	Interrupt Flag Set Register
0x010	USB_IFC	W1	Interrupt Flag Clear Register
0x014	USB_IEN	RW	Interrupt Enable Register
0x018	USB_ROUTE	RW	I/O Routing Register
0x3C008	USB_GAHBCFG	RW	AHB Configuration Register
0x3C00C	USB_GUSBCFG	RWH	USB Configuration Register
0x3C010	USB_GRSTCTL	RWH	Reset Register
0x3C014	USB_GINTSTS	RWH	Interrupt Register
0x3C018	USB_GINTMSK	RW	Interrupt Mask Register
0x3C01C	USB_GRXSTSR	R	Receive Status Debug Read Register
0x3C020	USB_GRXSTSP	R	Receive Status Read and Pop Register
0x3C024	USB_GRXFSIZ	RW	Receive FIFO Size Register
0x3C028	USB_GNPTXFSIZ	RW	Non-periodic Transmit FIFO Size Register
0x3C05C	USB_GDFIFOCFG	RW	Global DFIFO Configuration Register
0x3C104	USB_DIEPTXF1	RW	Device IN Endpoint Transmit FIFO 1 Size Register
0x3C108	USB_DIEPTXF2	RW	Device IN Endpoint Transmit FIFO 2 Size Register
0x3C10C	USB_DIEPTXF3	RW	Device IN Endpoint Transmit FIFO 3 Size Register
0x3C800	USB_DCFG	RW	Device Configuration Register
0x3C804	USB_DCTL	RWH	Device Control Register
0x3C808	USB_DSTS	R	Device Status Register
0x3C810	USB_DIEPMSK	RW	Device IN Endpoint Common Interrupt Mask Register
0x3C814	USB_DOEPMSK	RW	Device OUT Endpoint Common Interrupt Mask Register
0x3C818	USB_DAINT	R	Device All Endpoints Interrupt Register
0x3C81C	USB_DAINTMSK	RW	Device All Endpoints Interrupt Mask Register
0x3C834	USB_DIEPEMPMSK	RW	Device IN Endpoint FIFO Empty Interrupt Mask Register
0x3C900	USB_DIEP0CTL	RWH	Device IN Endpoint 0 Control Register
0x3C908	USB_DIEP0INT	RWH	Device IN Endpoint 0 Interrupt Register
0x3C910	USB_DIEP0TSIZ	RW	Device IN Endpoint 0 Transfer Size Register
0x3C914	USB_DIEP0DMAADDR	RW	Device IN Endpoint 0 DMA Address Register
0x3C918	USB_DIEP0TXFSTS	R	Device IN Endpoint 0 Transmit FIFO Status Register
0x3C920	USB_DIEP0_CTL	RWH	Device IN Endpoint x+1 Control Register
0x3C928	USB_DIEP0_INT	RWH	Device IN Endpoint x+1 Interrupt Register
0x3C930	USB_DIEP0_TSIZ	RW	Device IN Endpoint x+1 Transfer Size Register
0x3C934	USB_DIEP0_DMAADDR	RW	Device IN Endpoint x+1 DMA Address Register
0x3C938	USB_DIEP0_TXFSTS	R	Device IN Endpoint x+1 Transmit FIFO Status Register
0x3C940	USB_DIEP1_CTL	RWH	Device IN Endpoint x+1 Control Register
0x3C948	USB_DIEP1_INT	RWH	Device IN Endpoint x+1 Interrupt Register

Offset	Name	Type	Description
0x3C950	USB_DIEP1_TSIZ	RW	Device IN Endpoint x+1 Transfer Size Register
0x3C954	USB_DIEP1_DMAADDR	RW	Device IN Endpoint x+1 DMA Address Register
0x3C958	USB_DIEP1_TXFSTS	R	Device IN Endpoint x+1 Transmit FIFO Status Register
0x3C960	USB_DIEP2_CTL	RWH	Device IN Endpoint x+1 Control Register
0x3C968	USB_DIEP2_INT	RWH	Device IN Endpoint x+1 Interrupt Register
0x3C970	USB_DIEP2_TSIZ	RW	Device IN Endpoint x+1 Transfer Size Register
0x3C974	USB_DIEP2_DMAADDR	RW	Device IN Endpoint x+1 DMA Address Register
0x3C978	USB_DIEP2_TXFSTS	R	Device IN Endpoint x+1 Transmit FIFO Status Register
0x3CB00	USB_DOEP0CTL	RWH	Device OUT Endpoint 0 Control Register
0x3CB08	USB_DOEP0INT	RWH	Device OUT Endpoint 0 Interrupt Register
0x3CB10	USB_DOEP0TSIZ	RW	Device OUT Endpoint 0 Transfer Size Register
0x3CB14	USB_DOEP0DMAADDR	RW	Device OUT Endpoint 0 DMA Address Register
0x3CB20	USB_DOEP0_CTL	RWH	Device OUT Endpoint x+1 Control Register
0x3CB28	USB_DOEP0_INT	RWH	Device OUT Endpoint x+1 Interrupt Register
0x3CB30	USB_DOEP0_TSIZ	RWH	Device OUT Endpoint x+1 Transfer Size Register
0x3CB34	USB_DOEP0_DMAADDR	RW	Device OUT Endpoint x+1 DMA Address Register
0x3CB40	USB_DOEP1_CTL	RWH	Device OUT Endpoint x+1 Control Register
0x3CB48	USB_DOEP1_INT	RWH	Device OUT Endpoint x+1 Interrupt Register
0x3CB50	USB_DOEP1_TSIZ	RWH	Device OUT Endpoint x+1 Transfer Size Register
0x3CB54	USB_DOEP1_DMAADDR	RW	Device OUT Endpoint x+1 DMA Address Register
0x3CB60	USB_DOEP2_CTL	RWH	Device OUT Endpoint x+1 Control Register
0x3CB68	USB_DOEP2_INT	RWH	Device OUT Endpoint x+1 Interrupt Register
0x3CB70	USB_DOEP2_TSIZ	RWH	Device OUT Endpoint x+1 Transfer Size Register
0x3CB74	USB_DOEP2_DMAADDR	RW	Device OUT Endpoint x+1 DMA Address Register
0x3CE00	USB_PCGCCTL	RWH	Power and Clock Gating Control Register
0x3D000	USB_FIFO0D0	RW	Device EP 0 FIFO
...	USB_FIFO0Dx	RW	Device EP 0 FIFO
0x3D5FC	USB_FIFO0D383	RW	Device EP 0 FIFO
0x3E000	USB_FIFO1D0	RW	Device EP 1 FIFO
...	USB_FIFO1Dx	RW	Device EP 1 FIFO
0x3E5FC	USB_FIFO1D383	RW	Device EP 1 FIFO
0x3F000	USB_FIFO2D0	RW	Device EP 2 FIFO
...	USB_FIFO2Dx	RW	Device EP 2 FIFO
0x3F5FC	USB_FIFO2D383	RW	Device EP 2 FIFO
0x40000	USB_FIFO3D0	RW	Device EP 3 FIFO
...	USB_FIFO3Dx	RW	Device EP 3 FIFO
0x405FC	USB_FIFO3D383	RW	Device EP 3 FIFO
0x5C000	USB_FIFORAM0	RW	Direct Access to Data FIFO RAM for Debugging (2 KB)
...	USB_FIFORAMx	RW	Direct Access to Data FIFO RAM for Debugging (2 KB)
0x5C7FC	USB_FIFORAM511	RW	Direct Access to Data FIFO RAM for Debugging (2 KB)

15.6 Register Description

15.6.1 USB_CTRL - System Control Register

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset							0x0				0x0				0	0					0	0	0		0		0x2				0	
Access							RW				RW				RW	RW					RW	RW	RW		RW		RW				RW	
Name							BIASPROGEM23				BIASPROGEM01				VREGOSEN	VREGDIS					LEMADDRMEN	LEMNAKEN	LEMIDLEEN		LEMPHYCTRL		LEMOSCCTRL				DMPUAP	

Bit	Name	Reset	Access	Description
31:26	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
25:24	BIASPROGEM23	0x0	RW	**Regulator Bias Programming Value in EM2/3**
	Regulator bias current setting in EM2/3 (i.e. while USB in suspend).			
23:22	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
21:20	BIASPROGEM01	0x0	RW	**Regulator Bias Programming Value in EM0/1**
	Regulator bias current setting in EM0/1 (i.e. while USB active).			
19:18	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
17	VREGOSEN	0	RW	**VREGO Sense Enable**
	Set this bit to enable USB_VREGO voltage level sensing.			
16	VREGDIS	0	RW	**Voltage Regulator Disable**
	Set this bit to disable the voltage regulator.			
15:12	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
11	LEMADDRMEN	0	RW	**Low Energy Mode on Device Address Mismatch Enable**
	Set this bit to enter low energy mode during device address mismatch.			
10	LEMNAKEN	0	RW	**Low Energy Mode on OUT NAK Enable**
	Set this bit to enter low energy mode during NAK'ed OUT packets.			
9	LEMIDLEEN	0	RW	**Low Energy Mode on Bus Idle Enable**
	Set this bit to enter low energy mode during bus idle.			
8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7	LEMPHYCTRL	0	RW	**Low Energy Mode USB PHY Control**
	Configuration for USB PHY control when Low Energy Mode is active			

Value	Mode	Description
0	NONE	The USB PHY is not affected by Low Energy Mode.
1	LEM	The USB PHY is put into an energy saving state when Low Energy Mode is active.

6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5:4	LEMOSCCTRL	0x2	RW	**Low Energy Mode Oscillator Control**
	Configuration for oscillator control when Low Energy Mode is active			

Value	Mode	Description
0	NONE	Low Energy Mode has no effect on neither USBC or USHFRCO.
1	GATE	The USBC clock is gated when Low Energy Mode is active.
2	SUSPEND	The USBC clock is gated, and USHFRCO is suspended (if not selected as HFCLK) when Low Energy Mode is active.

3:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
1	DMPUAP	0	RW	**DMPU Active Polarity**
	Use this bit to select the active polarity of the USB_DMPU pin.			

Value	Mode	Description
0	LOW	USB_DMPU is active low.
1	HIGH	USB_DMPU is active high.

0	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

15.6.2 USB_STATUS - System Status Register

Offset	Bit Position																														
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																														0		0
Access																														R		R
Name																														LEMACTIVE		VREGOS

Bit	Name	Reset	Access	Description
31:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	LEMACTIVE	0	R	**Low Energy Mode Active**
	This bit is set when Low Energy Mode is active.			
1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	VREGOS	0	R	**VREGO Sense Output**
	USB_VREGO Voltage Sense output. 0 when no USB_VREGO voltage, 1 when USB_VREGO above approximately 1.8 V. Always 0 when VREGOSEN in USB_CTRL is 0.			

15.6.3 USB_IF - Interrupt Flag Register

Offset	Bit Position																														
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															1	1
Access																															R	R
Name																															VREGOSL	VREGOSH

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	VREGOSL	1	R	**VREGO Sense Low Interrupt Flag**
	Set when USB_VREGO drops below approximately 1.8 V.			
0	VREGOSH	1	R	**VREGO Sense High Interrupt Flag**
	Set when USB_VREGO goes above approximately 1.8 V.			

15.6.4 USB_IFS - Interrupt Flag Set Register

Offset	Bit Position																														
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															W1	W1
Name																															VREGOSL	VREGOSH

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	VREGOSL	0	W1	**Set VREGO Sense Low Interrupt Flag**
	Write to 1 to set the VREGO Sense Low Interrupt Flag.			
0	VREGOSH	0	W1	**Set VREGO Sense High Interrupt Flag**
	Write to 1 to set the VREGO Sense High Interrupt Flag.			

15.6.5 USB_IFC - Interrupt Flag Clear Register

Offset	Bit Position																														
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															W1	W1
Name																															VREGOSL	VREGOSH

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	VREGOSL	0	W1	**Clear VREGO Sense Low Interrupt Flag**
	Write to 1 to clear the VREGO Sense Low Interrupt Flag.			
0	VREGOSH	0	W1	**Clear VREGO Sense High Interrupt Flag**
	Write to 1 to clear the VREGO Sense High Interrupt Flag.			

15.6.6 USB_IEN - Interrupt Enable Register

Offset	Bit Position																														
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															RW	RW
Name																															VREGOSL	VREGOSH

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	VREGOSL	0	RW	**VREGO Sense Low Interrupt Enable**
	Enable interrupt on VREGO Sense Low.			
0	VREGOSH	0	RW	**VREGO Sense High Interrupt Enable**
	Enable interrupt on VREGO Sense High.			

15.6.7 USB_ROUTE - I/O Routing Register

Offset	Bit Position																															
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																														0		0
Access																														RW		RW
Name																														DMPUPEN		PHYPEN

Bit	Name	Reset	Access	Description
31:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	DMPUPEN	0	RW	**DMPU Pin Enable**
	When set, the USB_DMPU pin is enabled.			
1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	PHYPEN	0	RW	**USB PHY Pin Enable**
	When set, the USB PHY and USB pins are enabled. The USB_DP and USB_DM are changed from regular GPIO pins to USB pins.			

15.6.8 USB_GAHBCFG - AHB Configuration Register

This register can be used to configure the core after power-on or a change in mode. This register mainly contains AHB system-related configuration parameters. Do not change this register after the initial programming. The application must program this register before starting any transactions on either the AHB or the USB.

Offset	Bit Position																															
0x3C008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset									0	0	0														0		0	0x0				0
Access									RW	RW	RW														RW		RW	RW				RW
Name									AHBSINGLE	NOTIALLDMAWRIT	REMMEMSUPP														NPTXFEMPLVL		DMAEN	HBSTLEN				GLBLINTRMSK

Bit	Name	Reset	Access	Description
31:24	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
23	AHBSINGLE	0	RW	**AHB Single Support**
	This bit is set to enable SINGLE burst transfers for the remaining data in a transfer when the USB core is operating in FMA mode. When cleared, the remaining data is set using INCR burst size transfers. When set the remaining data is sent using a SINGLE burst size transfer.			
22	NOTIALLDMAWRIT	0	RW	**Notify All DMA Writes**
	This bit is programmed to enable the System DMA Done functionality for all the DMA write Transactions corresponding to the Channel/Endpoint. This bit is valid only when USB_GAHBCFG.REMMEMSUPP is set to 1. When set, the core asserts int_dma_req for all the DMA write transactions on the AHB interface along with int_dma_done, chep_last_transact and chep_number signal informations. The core waits for sys_dma_done signal for all the DMA write transactions in order to complete the transfer of a particular Channel/Endpoint. When cleared, the core asserts int_dma_req signal only for the last transaction of DMA write transfer corresponding to a particular Channel/Endpoint. Similarly, the core waits for sys_dma_done signal only for that transaction of DMA write to complete the transfer of a particular Channel/Endpoint.			
21	REMMEMSUPP	0	RW	**Remote Memory Support**
	This bit is programmed to enable the functionality to wait for the system DMA Done Signal for the DMA Write Transfers. When set, the int_dma_req output signal is asserted when HSOTG DMA starts write transfer to the external memory. When the core is done with the Transfers it asserts int_dma_done signal to flag the completion of DMA writes from HSOTG. The core then waits for sys_dma_done signal from the system to proceed further and complete the Data Transfer corresponding to a particular Channel/Endpoint. When cleared, the int_dma_req and int_dma_done signals are not asserted and the core proceeds with the assertion of the XferComp interrupt as soon as the DMA write transfer is done at the HSOTG Core Boundary and it doesn't wait for the sys_dma_done signal to complete the DATA.			
20:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7	NPTXFEMPLVL	0	RW	**Non-Periodic TxFIFO Empty Level**
	This bit is used only in Slave mode. In host mode this bit indicates when the Non-Periodic TxFIFO Empty Interrupt bit in the Core Interrupt register (USB_GINTSTS.NPTXFEMP) is triggered. In device mode, this bit indicates when IN endpoint Transmit FIFO empty interrupt (USB_DIEP0INT/USB_DIEPx_INT.TXFEMP) is triggered.			

Value	Mode	Description
0	HALFEMPTY	Host Mode: USB_GINTSTS.NPTXFEMP interrupt indicates that the Non-Periodic TxFIFO is half empty. Device Mode: USB_DIEP0INT/USB_DIEPx_INT.TXFEMP interrupt indicates that the IN Endpoint TxFIFO is half empty.
1	EMPTY	Host Mode: USB_GINTSTS.NPTXFEMP interrupt indicates that the Non-Periodic TxFIFO is completely empty. Device Mode: USB_DIEP0INT/USB_DIEPx_INT.TXFEMP interrupt indicates that the IN Endpoint TxFIFO is completely empty.

6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	DMAEN	0	RW	**DMA Enable**
	When set to 0 the core operates in Slave mode. When set to 1 the core operates in a DMA mode.			
4:1	HBSTLEN	0x0	RW	**Burst Length/Type**
	This field is used in DMA mode.			

Value	Mode	Description
0	SINGLE	Single transfer.
1	INCR	Incrementing burst of unspecified length.
3	INCR4	4-beat incrementing burst.
5	INCR8	8-beat incrementing burst.
7	INCR16	16-beat incrementing burst.

0	GLBLINTRMSK	0	RW	**Global Interrupt Mask**
	The application uses this bit to mask or unmask the interrupt line assertion to itself. Irrespective of this bit's setting, the interrupt status registers are updated by the core. Set to unmask.			

15.6.9 USB_GUSBCFG - USB Configuration Register

This register can be used to configure the core after power-on or a changing to Host mode or Device mode. It contains USB and USB-PHY related configuration parameters. The application must program this register before starting any transactions on either the AHB or the USB. Do not make changes to this register after the initial programming.

Offset	Bit Position																														
0x3C00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0			0						0									0x5								0				0x0	
Access	W1			RW						RW									RW								RW				RW	
Name	CORRUPTTXPKT			TXENDDELAY						TERMSELDLPULSE									USBTRDTIM								FSINTF				TOUTCAL	

Bit	Name	Reset	Access	Description
31	CORRUPTTXPKT	0	W1	**Corrupt Tx packet**
	This bit is for debug purposes only. Never Set this bit to 1. The application should always write 0 to this bit.			
30:29	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
28	TXENDDELAY	0	RW	**Tx End Delay**
	Writing 1 to this bit enables the core to follow the TxEndDelay timings as per UTMI+ specification 1.05 section 4.1.5 for opmode signal during remote wakeup.			
27:23	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
22	TERMSELDLPULSE	0	RW	**TermSel DLine Pulsing Selection**
	This bit selects utmi_termselect to drive data line pulse during SRP.			

Value	Mode	Description
0	TXVALID	Data line pulsing using utmi_txvalid.
1	TERMSEL	Data line pulsing using utmi_termsel.

21:14	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
13:10	USBTRDTIM	0x5	RW	**USB Turnaround Time**
	Sets the turnaround time in PHY clocks. Specifies the response time For a MAC request to the Packet FIFO Controller (PFC) to fetch data from the DFIFO (SPRAM). Always write this field to 5.			
9:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	FSINTF	0	RW	**Full-Speed Serial Interface Select**
	Always write this bit to 0.			
4:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2:0	TOUTCAL	0x0	RW	**Timeout Calibration**
	Always write this field to 0.			

15.6.10 USB_GRSTCTL - Reset Register

The application uses this register to reset various hardware features inside the core.

Offset	Bit Position																														
0x3C010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	1	0																					0x00				0	0			0	0
Access	R	R																					RW				RW1H	RW1H			RW1H	RW1H
Name	AHBIDLE	DMAREQ																					TXFNUM				TXFFLSH	RXFFLSH			PIUFSSFTRST	CSFTRST

Bit	Name	Reset	Access	Description
31	AHBIDLE	1	R	**AHB Master Idle**
	Indicates that the AHB Master State Machine is in the IDLE condition.			
30	DMAREQ	0	R	**DMA Request Signal**
	Indicates that the DMA request is in progress. Used for debug.			
29:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:6	TXFNUM	0x00	RW	**TxFIFO Number**

This is the FIFO number that must be flushed using the TxFIFO Flush bit. This field must not be changed until the core clears the TxFIFO Flush bit.

Value	Mode	Description
0	F0	Host mode: Non-periodic TxFIFO flush. Device: Tx FIFO 0 flush
1	F1	Host mode: Periodic TxFIFO flush. Device: TXFIFO 1 flush.
2	F2	Device mode: TXFIFO 2 flush.
3	F3	Device mode: TXFIFO 3 flush.
4	F4	Device mode: TXFIFO 4 flush.
5	F5	Device mode: TXFIFO 5 flush.
6	F6	Device mode: TXFIFO 6 flush.
16	FALL	Flush all the transmit FIFOs in device or host mode.

5	TXFFLSH	0	RW1H	**TxFIFO Flush**

This bit selectively flushes a single or all transmit FIFOs, but cannot do so if the core is in the midst of a transaction. The application must write this bit only after checking that the core is neither writing to the TxFIFO nor reading from the TxFIFO. NAK Effective Interrupt ensures the core is not reading from the FIFO. USB_GRSTCTL.AHBIDLE ensures the core is not writing anything to the FIFO. Flushing is normally recommended when FIFOs are reconfigured. FIFO flushing is also recommended during device endpoint disable. The application must wait until the core clears this bit before performing any operations. This bit takes eight clocks to clear.

4	RXFFLSH	0	RW1H	**RxFIFO Flush**

The application can flush the entire RxFIFO using this bit, but must first ensure that the core is not in the middle of a transaction. The application must only write to this bit after checking that the core is neither reading from the RxFIFO nor writing to the RxFIFO. The application must wait until the bit is cleared before performing any other operations. This bit requires 8 clocks to clear.

3:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	PIUFSSFTRST	0	RW1H	**PIU FS Dedicated Controller Soft Reset**

Resets the PIU FS Dedicated Controller All module state machines in FS Dedicated Controller of PIU are reset to the IDLE state. Used to reset the FS Dedicated controller in PIU in case of any PHY Errors like Loss of activity or Babble Error resulting in the PHY remaining in RX state for more than one frame boundary

0	CSFTRST	0	RW1H	**Core Soft Reset**

Resets the core by clearing the interrupts and all the CSR registers except the following register bits: USB_PCGCCTL.RSTPDWNMODULE, USB_PCGCCTL.GATEHCLK, USB_PCGCCTL.PWRCLMP, USB_GUSBCFG.FSINTF, USB_HCFG.FSLSPCLKSEL, USB_DCFG.DEVSPD.

All module state machines (except the AHB Slave Unit) are reset to the IDLE state, and all the transmit FIFOs and the receive FIFO are flushed. Any transactions on the AHB Master are terminated as soon as possible, after gracefully completing the last data phase of an AHB transfer. Any transactions on the USB are terminated immediately. The application can write to this bit any time it wants to reset the core. This is a self-clearing bit and the core clears this bit after all the necessary logic is reset in the core, which can take several clocks, depending on the current state of the core. Once this bit is cleared software must wait at least 3 clock cycles before doing any access to the core. Software must also must check that bit 31 of this register is 1 (AHB Master is IDLE) before starting any operation.

15.6.11 USB_GINTSTS - Interrupt Register

This register interrupts the application for system-level events in the current mode (Device mode or Host mode). Some of the bits in this register are valid only in Host mode, while others are valid in Device mode only. This register also indicates the current mode. To clear the interrupt status bits of type RW1H, the application must write 1 into the bit.

The FIFO status interrupts are read only; once software reads from or writes to the FIFO while servicing these interrupts, FIFO interrupt conditions are cleared automatically.

The application must clear the USB_GINTSTS register at initialization before unmasking the interrupt bit to avoid any interrupts generated prior to initialization.

Offset	Bit Position																															
0x3C014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0								0	0	0	0	0	0			0	0	0	0	0	0			0	0		0	0			0
Access	RW1H								RW1H	RW1H	RW1H	RW1H	R	R			RW	RW1H	RW1H	RW1H	RW1H	RW1H			R	R		R	RW1H			R
Name	WKUPINT								RESETDET	FETSUSP	INCOMPLP	INCOMPISOIN	OEPINT	IEPINT			EOPF	ISOOUTDROP	ENUMDONE	USBRST	USBSUSP	ERLYSUSP			GOUTNAKEFF	GINNAKEFF		RXFLVL	SOF			CURMOD

Bit	Name	Reset	Access	Description
31	WKUPINT	0	RW1H	**Resume/Remote Wakeup Detected Interrupt**
	Wakeup Interrupt during Suspend state. In Device mode this interrupt is asserted only when Host Initiated Resume is detected on USB. In Host mode this interrupt is asserted only when Device Initiated Remote Wakeup is detected on USB. This bit can be set only by the core and the application should write 1 to clear.			
30:24	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
23	RESETDET	0	RW1H	**Reset detected Interrupt**
	In Device mode, this interrupt is asserted when a reset is detected on the USB in EM2 when the device is in Suspend. In Host mode, this interrupt is not asserted.			
22	FETSUSP	0	RW1H	**Data Fetch Suspended**
	This interrupt is valid only in DMA mode. This interrupt indicates that the core has stopped fetching data for IN endpoints due to the unavailability of TxFIFO space or Request Queue space. This interrupt is used by the application for an endpoint mismatch algorithm. For example, after detecting an endpoint mismatch, the application: Sets a Global non-periodic IN NAK handshake, Disables In endpoints, Flushes the FIFO, Determines the token sequence from the IN Token Sequence, Re-enables the endpoints, Clears the Global non-periodic IN NAK handshake. If the Global non-periodic IN NAK is cleared, the core has not yet fetched data for the IN endpoint, and the IN token is received: the core generates an IN Token Received when FIFO Empty interrupt. The OTG then sends the host a NAK response. To avoid this scenario, the application can check the USB_GINTSTS.FETSUSP interrupt, which ensures that the FIFO is full before clearing a Global NAK handshake. Alternatively, the application can mask the IN Token Received when FIFO Empty interrupt when clearing a Global IN NAK handshake.			
21	INCOMPLP	0	RW1H	**Incomplete Periodic Transfer**
	In Host mode, the core sets this interrupt bit when there are incomplete periodic transactions still pending which are scheduled for the current frame. In Device mode, the core sets this interrupt to indicate that there is at least one isochronous OUT endpoint on which the transfer is not completed in the current frame. This bit can be set only by the core and the application should write 1 to clear it.			
20	INCOMPISOIN	0	RW1H	**Incomplete Isochronous IN Transfer**
	The core sets this interrupt to indicate that there is at least one isochronous IN endpoint on which the transfer is not completed in the current frame.			
19	OEPINT	0	R	**OUT Endpoints Interrupt**
	The core sets this bit to indicate that an interrupt is pending on one of the OUT endpoints of the core (in Device mode). The application must read the Device All Endpoints Interrupt (USB_DAINT) register to determine the exact number of the OUT endpoint on which the interrupt occurred, and then read the corresponding Device OUT Endpoint-x Interrupt (USB_DOEP0INT/USB_DOEPx_INT) register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding USB_DOEP0INT/USB_DOEPx_INT register to clear this bit.			
18	IEPINT	0	R	**IN Endpoints Interrupt**
	The core sets this bit to indicate that an interrupt is pending on one of the IN endpoints of the core (in Device mode). The application must read the Device All Endpoints Interrupt (USB_DAINT) register to determine the exact number of the IN endpoint on Device IN Endpoint-x Interrupt (USB_DIEP0INT/USB_DIEPx_INT) register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding USB_DIEP0INT/USB_DIEPx_INT register to clear this bit.			
17:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15	EOPF	0	RW	**End of Periodic Frame Interrupt**
	Indicates that the period specified in the Periodic Frame Interval field of the Device Configuration register (DCFG_PERFRINT) has been reached in the current microframe.			

Bit	Name	Reset	Access	Description
14	ISOOUTDROP	0	RW1H	**Isochronous OUT Packet Dropped Interrupt**
	The core sets this bit when it fails to write an isochronous OUT packet into the RxFIFO because the RxFIFO does not have enough space to accommodate a maximum packet size packet for the isochronous OUT endpoint.			
13	ENUMDONE	0	RW1H	**Enumeration Done**
	The core sets this bit to indicate that speed enumeration is complete. The application must read the Device Status (USB_DSTS) register to obtain the enumerated speed.			
12	USBRST	0	RW1H	**USB Reset**
	The core sets this bit to indicate that a reset is detected on the USB.			
11	USBSUSP	0	RW1H	**USB Suspend**
	The core sets this bit to indicate that a suspend was detected on the USB. The core enters the Suspended state when there is no activity on the bus for an extended period of time.			
10	ERLYSUSP	0	RW1H	**Early Suspend**
	The core sets this bit to indicate that an Idle state has been detected on the USB for 3 ms.			
9:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7	GOUTNAKEFF	0	R	**Global OUT NAK Effective**
	Indicates that the Set Global OUT NAK bit in the Device Control register (USB_DCTL.SGOUTNAK), set by the application, has taken effect in the core. This bit can be cleared by writing the Clear Global OUT NAK bit in the Device Control register (USB_DCTL.CGOUTNAK).			
6	GINNAKEFF	0	R	**Global IN Non-periodic NAK Effective**
	Indicates that the Set Global Non-periodic IN NAK bit in the Device Control register (USB_DCTL.SGNPINNAK), set by the application, has taken effect in the core. That is, the core has sampled the Global IN NAK bit set by the application. This bit can be cleared by clearing the Clear Global Non-periodic IN NAK bit in the Device Control register (USB_DCTL.CGNPINNAK). This interrupt does not necessarily mean that a NAK handshake is sent out on the USB. The STALL bit takes precedence over the NAK bit.			
5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	RXFLVL	0	R	**RxFIFO Non-Empty**
	Indicates that there is at least one packet pending to be read from the RxFIFO.			
3	SOF	0	RW1H	**Start of Frame**
	In Host mode, the core sets this bit to indicate that an SOF (FS) or Keep-Alive (LS) is transmitted on the USB. The application must write a 1 to this bit to clear the interrupt. In Device mode, in the core sets this bit to indicate that an SOF token has been received on the USB. The application can read the Device Status register to get the current frame number. This interrupt is seen only when the core is operating at full-speed (FS). This bit can be set only by the core and the application should write 1 to clear it.			
2:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	CURMOD	0	R	**Current Mode of Operation**
	Indicates the current mode.			

Value	Mode	Description
0	DEVICE	Device mode.

15.6.12 USB_GINTMSK - Interrupt Mask Register

This register works with the Interrupt Register (USB_GINTSTS) to interrupt the application. When an interrupt bit is masked (bit is 0), the interrupt associated with that bit is not generated. However, the USB_GINTSTS register bit corresponding to that interrupt is still set.

Offset	Bit Position																															
0x3C018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0								0	0	0	0	0	0			0	0	0	0	0	0			0	0		0	0		0	
Access	RW								RW	RW	RW	RW	RW	RW			RW	RW	RW	RW	RW	RW			RW	RW		RW	RW		RW	
Name	WKUPINTMSK								RESETDETMSK	FETSUSPMSK	INCOMPLPMSK	INCOMPISOINMSK	OEPINTMSK	IEPINTMSK			EOPFMSK	ISOOUTDROPMSK	ENUMDONEMSK	USBRSTMSK	USBSUSPMSK	ERLYSUSPMSK			GOUTNAKEFFMSK	GINNAKEFFMSK		RXFLVLMSK	SOFMSK		MODEMISMSK	

Bit	Name	Reset	Access	Description
31	WKUPINTMSK	0	RW	**Resume/Remote Wakeup Detected Interrupt Mask**
	Set to 1 to unmask WKUPINT interrupt.			
30:24	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
23	RESETDETMSK	0	RW	**Reset detected Interrupt Mask**
	Set to 1 to unmask RESETDET interrupt.			
22	FETSUSPMSK	0	RW	**Data Fetch Suspended Mask**
	Set to 1 to unmask FETSUSP interrupt.			
21	INCOMPLPMSK	0	RW	**Incomplete Periodic Transfer Mask**
	Set to 1 to unmask INCOMPLP interrupt.			
20	INCOMPISOINMSK	0	RW	**Incomplete Isochronous IN Transfer Mask**
	Set to 1 to unmask INCOMPISOIN interrupt.			
19	OEPINTMSK	0	RW	**OUT Endpoints Interrupt Mask**
	Set to 1 to unmask OEPINT interrupt.			
18	IEPINTMSK	0	RW	**IN Endpoints Interrupt Mask**
	Set to 1 to unmask IEPINT interrupt.			
17:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15	EOPFMSK	0	RW	**End of Periodic Frame Interrupt Mask**
	Set to 1 to unmask EOPF interrupt.			
14	ISOOUTDROPMSK	0	RW	**Isochronous OUT Packet Dropped Interrupt Mask**
	Set to 1 to unmask ISOOUTDROP interrupt.			
13	ENUMDONEMSK	0	RW	**Enumeration Done Mask**
	Set to 1 to unmask ENUMDONE interrupt.			
12	USBRSTMSK	0	RW	**USB Reset Mask**
	Set to 1 to unmask USBRST interrupt.			
11	USBSUSPMSK	0	RW	**USB Suspend Mask**
	Set to 1 to unmask USBSUSP interrupt.			
10	ERLYSUSPMSK	0	RW	**Early Suspend Mask**
	Set to 1 to unmask ERLYSUSP interrupt.			
9:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7	GOUTNAKEFFMSK	0	RW	**Global OUT NAK Effective Mask**
	Set to 1 to unmask GOUTNAKEFF interrupt.			
6	GINNAKEFFMSK	0	RW	**Global Non-periodic IN NAK Effective Mask**
	Set to 1 to unmask GINNAKEFF interrupt.			
5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	RXFLVLMSK	0	RW	**Receive FIFO Non-Empty Mask**

Bit	Name	Reset	Access	Description
	Set to 1 to unmask RXFLVL interrupt.			
3	SOFMSK	0	RW	**Start of Frame Mask**
	Set to 1 to unmask SOF interrupt.			
2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	MODEMISMSK	0	RW	**Mode Mismatch Interrupt Mask**
	Set to 1 to unmask MODEMIS interrupt.			
0	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

15.6.13 USB_GRXSTSR - Receive Status Debug Read Register

A read to the Receive Status Debug Read register returns the contents of the top of the Receive FIFO. The receive status contents must be interpreted differently in Host and Device modes. The core ignores the receive status pop/read when the receive FIFO is empty and returns a value of 0x00000000. The application must only pop the Receive Status FIFO when the Receive FIFO Non-Empty bit of the Core Interrupt register (USB_GINTSTS.RXFLVL) is asserted.

Offset	Bit Position																															
0x3C01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset								0x0				0x0				0x0		0x000											0x0			
Access								R				R				R		R											R			
Name								FN				PKTSTS				DPID		BCNT											CHEPNUM			

Bit	Name	Reset	Access	Description
31:25	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
24:21	FN	0x0	R	**Frame Number**
	This is the least significant 4 bits of the Frame number in which the packet is received on the USB.			
20:17	PKTSTS	0x0	R	**Packet Status (host or device)**
	Indicates the status of the received packet.			

Value	Mode	Description
1	GOUTNAK	Device mode: Global OUT NAK (triggers an interrupt).
2	PKTRCV	Host mode: IN data packet received. Device mode: OUT data packet received.
3	XFERCOMPL	Host mode: IN transfer completed (triggers an interrupt). Device mode: OUT transfer completed (triggers an interrupt).
4	SETUPCOMPL	Device mode: SETUP transaction completed (triggers an interrupt).
5	TGLERR	Host mode: Data toggle error (triggers an interrupt).
6	SETUPRCV	Device mode: SETUP data packet received.
7	CHLT	Host mode: Channel halted (triggers an interrupt).

16:15	DPID	0x0	R	**Data PID (host or device)**
	Host mode: Indicates the Data PID of the received packet. Device mode: Indicates the Data PID of the received OUT data packet.			

Value	Mode	Description
0	DATA0	DATA0 PID.
1	DATA1	DATA1 PID.
2	DATA2	DATA2 PID.

Bit	Name	Reset	Access	Description

Value	Mode	Description
3	MDATA	MDATA PID.

Bit	Name	Reset	Access	Description
14:4	BCNT	0x000	R	**Byte Count (host or device)**

Host mode: Indicates the byte count of the received IN data packet.

Device mode: Indicates the byte count of the received data packet.

Bit	Name	Reset	Access	Description
3:0	CHEPNUM	0x0	R	**Channel Number host only / Endpoint Number**

Host mode: Indicates the channel number to which the current received packet belongs.

Device mode: Indicates the endpoint number to which the current received packet belongs.

15.6.14 USB_GRXSTSP - Receive Status Read and Pop Register

A read to the Receive Status Read and Pop register returns the contents of the top of the Receive FIFO and pops the top data entry out of the RxFIFO. The receive status contents must be interpreted differently in Host and Device modes. The core ignores the receive status pop/read when the receive FIFO is empty and returns a value of 0x00000000. The application must only pop the Receive Status FIFO when the Receive FIFO Non-Empty bit of the Core Interrupt register (USB_GINTSTS.RXFLVL) is asserted.

Offset	Bit Position																															
0x3C020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset								0x0				0x0				0x0		0x000											0x0			
Access								R				R				R		R											R			
Name								FN				PKTSTS				DPID		BCNT											CHEPNUM			

Bit	Name	Reset	Access	Description
31:25	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
24:21	FN	0x0	R	**Frame Number**

This is the least significant 4 bits of the Frame number in which the packet is received on the USB.

Bit	Name	Reset	Access	Description
20:17	PKTSTS	0x0	R	**Packet Status (host or device)**

Indicates the status of the received packet.

Value	Mode	Description
1	GOUTNAK	Device mode: Global OUT NAK (triggers an interrupt).
2	PKTRCV	Host mode: IN data packet received. Device mode: OUT data packet received.
3	XFERCOMPL	Host mode: IN transfer completed (triggers an interrupt). Device mode: OUT transfer completed (triggers an interrupt).
4	SETUPCOMPL	Device mode: SETUP transaction completed (triggers an interrupt).
5	TGLERR	Host mode: Data toggle error (triggers an interrupt).
6	SETUPRCV	Device mode: SETUP data packet received.
7	CHLT	Host mode: Channel halted (triggers an interrupt).

Bit	Name	Reset	Access	Description
16:15	DPID	0x0	R	**Data PID (host or device)**

Host mode: Indicates the Data PID of the received packet.

Device mode: Indicates the Data PID of the received OUT data packet.

Value	Mode	Description
0	DATA0	DATA0 PID.

Bit	Name	Reset	Access	Description
	Value	Mode	Description	
	1	DATA1	DATA1 PID.	
	2	DATA2	DATA2 PID.	
	3	MDATA	MDATA PID.	
14:4	BCNT	0x000	R	**Byte Count (host or device)**
	Host mode: Indicates the byte count of the received IN data packet.			
	Device mode: Indicates the byte count of the received data packet.			
3:0	CHEPNUM	0x0	R	**Channel Number host only / Endpoint Number**
	Host mode: Indicates the channel number to which the current received packet belongs.			
	Device mode: Indicates the endpoint number to which the current received packet belongs.			

15.6.15 USB_GRXFSIZ - Receive FIFO Size Register

The application can program the RAM size that must be allocated to the RxFIFO.

Offset	Bit Position																													
0x3C024	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0x200								
Access																							RW								
Name																							RXFDEP								

Bit	Name	Reset	Access	Description
31:10	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
9:0	RXFDEP	0x200	RW	**RxFIFO Depth**
	This value is in terms of 32-bit words. Minimum value is 16. Maximum value is 512.			

15.6.16 USB_GNPTXFSIZ - Non-periodic Transmit FIFO Size Register

The application can program the RAM size and the memory start address for the Non-periodic TxFIFO.

Offset	Bit Position																													
0x3C028	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset								0x0200																				0x200				
Access								RW																				RW				
Name								NPTXFINEPTXF0DEP																				NPTXFSTADDR				

Bit	Name	Reset	Access	Description
31:16	NPTXFINEPTXF0DEP	0x0200	RW	**Non-periodic TxFIFO Depth host only / IN Endpoint TxFIFO 0 Depth**
	This value is in terms of 32-bit words. Minimum value is 16. Maximum value is 512.			
15:10	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
9:0	NPTXFSTADDR	0x200	RW	**Non-periodic Transmit RAM Start Address host only**
	This field contains the memory start address for Non-periodic Transmit FIFO RAM. Programmed values must not exceed the reset value.			

15.6.17 USB_GDFIFOCFG - Global DFIFO Configuration Register

Offset	Bit Position																													
0x3C05C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x05F8																0x0600															
Access	RW																RW															
Name	EPINFOBASEADDR																GDFIFOCFG															

Bit	Name	Reset	Access	Description
31:16	EPINFOBASEADDR	0x05F8	RW	**Endpoint Info Base Address**
	This field provides the start address of the EP info controller.			
15:0	GDFIFOCFG	0x0600	RW	**DFIFO Config**
	This field is for dynamic programming of the DFIFO Size. This value takes effect only when the application programs a non zero value to this register. The core does not have any corrective logic if the FIFO sizes are programmed incorrectly.			

15.6.18 USB_DIEPTXF1 - Device IN Endpoint Transmit FIFO 1 Size Register

This register holds the size and memory start address of IN endpoint TxFIFO 1 in Device mode. For IN endpoint FIFO 0 use USB_GNPTXFSIZ register for programming the size and memory start address.

Offset	Bit Position																													
0x3C104	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset							0x200															0x400									
Access							RW															RW									
Name							INEPNTXFDEP															INEPNTXFSTADDR									

Bit	Name	Reset	Access	Description
31:26	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
25:16	INEPNTXFDEP	0x200	RW	**IN Endpoint TxFIFO Depth**
	This value is in terms of 32-bit words. Minimum value is 16. Maximum value is 512.			
15:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:0	INEPNTXFSTADDR	0x400	RW	**IN Endpoint FIFO 1 Transmit RAM Start Address**
	This field contains the memory start address for IN endpoint Transmit FIFO 1.			

15.6.19 USB_DIEPTXF2 - Device IN Endpoint Transmit FIFO 2 Size Register

This register holds the size and memory start address of IN endpoint TxFIFO 2 in Device mode. For IN endpoint FIFO 0 use USB_GNPTXFSIZ register for programming the size and memory start address.

Offset	Bit Position																													
0x3C108	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset							0x200															0x600									
Access							RW															RW									
Name							INEPNTXFDEP															INEPNTXFSTADDR									

Bit	Name	Reset	Access	Description
31:26	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
25:16	INEPNTXFDEP	0x200	RW	**IN Endpoint TxFIFO Depth**
	This value is in terms of 32-bit words. Minimum value is 16. Maximum value is 512.			
15:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:0	INEPNTXFSTADDR	0x600	RW	**IN Endpoint FIFO 2 Transmit RAM Start Address**
	This field contains the memory start address for IN endpoint Transmit FIFO 2.			

15.6.20 USB_DIEPTXF3 - Device IN Endpoint Transmit FIFO 3 Size Register

This register holds the size and memory start address of IN endpoint TxFIFO 3 in Device mode. For IN endpoint FIFO 0 use USB_GNPTXFSIZ register for programming the size and memory start address.

Offset	Bit Position																													
0x3C10C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset							0x200														0x800											
Access							RW														RW											
Name							INEPNTXFDEP														INEPNTXFSTADDR											

Bit	Name	Reset	Access	Description
31:26	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
25:16	INEPNTXFDEP	0x200	RW	**IN Endpoint TxFIFO Depth**
	This value is in terms of 32-bit words. Minimum value is 16. Maximum value is 512.			
15:12	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
11:0	INEPNTXFSTADDR	0x800	RW	**IN Endpoint FIFO 3 Transmit RAM Start Address**
	This field contains the memory start address for IN endpoint Transmit FIFO 3.			

15.6.21 USB_DCFG - Device Configuration Register

This register configures the core in Device mode after power-on or after certain control commands or enumeration. Do not make changes to this register after initial programming.

Offset	Bit Position																													
0x3C800	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x02																0			0x0		0x00							0	0	0x0	
Access	RW																RW			RW		RW							RW	RW	RW	
Name	RESVALID																ERRATICINTMSK			PERFRINT		DEVADDR							ENA32KHZSUSP	NZSTSOUTHSHK	DEVSPD	

Bit	Name	Reset	Access	Description
31:26	RESVALID	0x02	RW	**Resume Validation Period**
	This field is effective only when USB_DCFG.ENA32KHZSUSP is set. It will control the resume period when the core resumes from suspend. The core counts for RESVALID number of clock cycles to detect a valid resume when USB_DCFG.ENA32KHZSUSP is set.			
25:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15	ERRATICINTMSK	0	RW	
14:13	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
12:11	PERFRINT	0x0	RW	**Periodic Frame Interval**
	Indicates the time within a frame at which the application must be notified using the End Of Periodic Frame Interrupt. This can be used to determine if all the isochronous traffic for that frame is complete.			

Value	Mode	Description
0	80PCNT	80% of the frame interval.
1	85PCNT	85% of the frame interval.
2	90PCNT	90% of the frame interval.
3	95PCNT	95% of the frame interval.

10:4	DEVADDR	0x00	RW	**Device Address**
	The application must program this field after every SetAddress control command.			
3	ENA32KHZSUSP	0	RW	**Enable 32 KHz Suspend mode**
	When this bit is set, the core expects that the PHY clock during Suspend is switched from 48 MHz to 32 KHz.			
2	NZSTSOUTHSHK	0	RW	**Non-Zero-Length Status OUT Handshake**
	The application can use this field to select the handshake the core sends on receiving a nonzero-length data packet during the OUT transaction of a control transfer's Status stage. When set to 1 send a STALL handshake on a nonzero-length status OUT transaction and do not send the received OUT packet to the application. When set to 0 send the received OUT packet to the application (zerolength or nonzero-length) and send a handshake based on the NAK and STALL bits for the endpoint in the Device Endpoint Control register.			
1:0	DEVSPD	0x0	RW	**Device Speed**

Bit	Name	Reset	Access	Description

Indicates the speed at which the application requires the core to enumerate, or the maximum speed the application can support. However, the actual bus speed is determined only after the chirp sequence is completed, and is based on the speed of the USB host to which the core is connected.

Value	Mode	Description
2	LS	Low speed (PHY clock is 6 MHz). If you select 6 MHz LS mode, you must do a soft reset.
3	FS	Full speed (PHY clock is 48 MHz).

15.6.22 USB_DCTL - Device Control Register

Offset	Bit Position																														
0x3C804	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																0	0				0	0	0	0	0	0x0			0	0	1	0
Access																RW	RW				RW	W1	W1	W1	W1	RW			R	R	RW	RW
Name																NAKONBBLE	IGNRFRMNUM				PWRONPRGDONE	CGOUTNAK	SGOUTNAK	CGNPINNAK	SGNPINNAK	TSTCTL			GOUTNAKSTS	GNPINNAKSTS	SFTDISCON	RMTWKUPSIG

Bit	Name	Reset	Access	Description
31:17	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
16	NAKONBBLE	0	RW	**NAK on Babble Error**
	Set NAK automatically on babble. The core sets NAK automatically for the endpoint on which babble is received.			
15	IGNRFRMNUM	0	RW	**Ignore Frame number For Isochronous End points**
	When set to 0 the core transmits the packets only in the frame number in which they are intended to be transmitted. When set to 1 the core ignores the frame number, sending packets immediately as the packets are ready.			
14:12	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
11	PWRONPRGDONE	0	RW	**Power-On Programming Done**
	The application uses this bit to indicate that register programming is completed after a wake-up from Power Down mode.			
10	CGOUTNAK	0	W1	**Clear Global OUT NAK**
	A write to this field clears the Global OUT NAK.			
9	SGOUTNAK	0	W1	**Set Global OUT NAK**
	A write to this field sets the Global OUT NAK. The application uses this bit to send a NAK handshake on all OUT endpoints. The application must set this bit only after making sure that the Global OUT NAK Effective bit in the Core Interrupt Register (USB_GINTSTS.GOUTNAKEFF) is cleared.			
8	CGNPINNAK	0	W1	**Clear Global Non-periodic IN NAK**
	A write to this field clears the Global Non-periodic IN NAK.			
7	SGNPINNAK	0	W1	**Set Global Non-periodic IN NAK**
	A write to this field sets the Global Non-periodic IN NAK. The application uses this bit to send a NAK handshake on all non-periodic IN endpoints. The application must set this bit only after making sure that the Global IN NAK Effective bit in the Core Interrupt Register (USB_GINTSTS.GINNAKEFF) is cleared.			
6:4	TSTCTL	0x0	RW	**Test Control**
	Set to a non-zero value to enable test control.			

Value	Mode	Description
0	DISABLE	Test mode disabled.
1	J	Test_J mode.
2	K	Test_K mode.
3	SE0NAK	Test_SE0_NAK mode.
4	PACKET	Test_Packet mode.

Bit	Name	Reset	Access	Description

Value	Mode	Description
5	FORCE	Test_Force_Enable.

3	GOUTNAKSTS	0	R	**Global OUT NAK Status**

When this bit is 0 a handshake is sent based on the FIFO Status and the NAK and STALL bit settings. When this bit is 1 no data is written to the RxFIFO, irrespective of space availability. Sends a NAK handshake on all packets, except on SETUP transactions. All isochronous OUT packets are dropped.

2	GNPINNAKSTS	0	R	**Global Non-periodic IN NAK Status**

When this bit is 0 a handshake is sent out based on the data availability in the transmit FIFO. When this bit is 1 a NAK handshake is sent out on all non-periodic IN endpoints, irrespective of the data availability in the transmit FIFO.

1	SFTDISCON	1	RW	**Soft Disconnect**

The application uses this bit to signal the core to do a soft disconnect. As long as this bit is set, the host does not see that the device is connected, and the device does not receive signals on the USB. The core stays in the disconnected state until the application clears this bit. When suspended, the minimum duration for which the core must keep this bit set is 1 ms + 2.5 us. When IDLE or performing transactions, the minimum duration for which the core must keep this bit set is 2.5 us.

0	RMTWKUPSIG	0	RW	**Remote Wakeup Signaling**

When the application sets this bit, the core initiates remote signaling to wake up the USB host. The application must set this bit to instruct the core to exit the Suspend state. As specified in the USB 2.0 specification, the application must clear this bit 1-15 ms after setting it.

15.6.23 USB_DSTS - Device Status Register

This register indicates the status of the core with respect to USB-related events. It must be read on interrupts from Device All Interrupts (USB_DAINT) register.

Offset	Bit Position																															
0x3C808	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset									0x0		0x0000																		0	0x1		0
Access									R		R																		R	R		R
Name									DEVLNSTS		SOFFN																		ERRTICERR	ENUMSPD		SUSPSTS

Bit	Name	Reset	Access	Description
31:24	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
23:22	DEVLNSTS	0x0	R	**Device Line Status**

Indicates the current logic level USB data lines: DEVLNSTS[1]: Logic level of D+; DEVLNSTS[0]: Logic level of D-.

21:8	SOFFN	0x0000	R	**Frame Number of the Received SOF**

This field contains a Frame number. This field may return a non zero value if read immediately after power on reset. In case the register bits reads non zero immediately after power on reset it does not indicate that SOF has been received from the host. The read value of this interrupt is valid only after a valid connection between host and device is established.

7:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	ERRTICERR	0	R	**Erratic Error**

The core sets this bit to report any erratic errors (PHY error) Due to erratic errors, the core goes into Suspended state and an interrupt is generated to the application with Early Suspend bit of the Core Interrupt register (USB_GINTSTS.ERLYSUSP). If the early suspend is asserted due to an erratic error, the application can only perform a soft disconnect recover.

2:1	ENUMSPD	0x1	R	**Enumerated Speed**

Indicates the speed at which the core has come up after speed detection through a chirp sequence.

Value	Mode	Description
2	LS	Low speed (PHY clock is running at 6 MHz).

Bit	Name	Reset	Access	Description
	Value	Mode	Description	
	3	FS	Full speed (PHY clock is running at 48 MHz).	
0	SUSPSTS	0	R	**Suspend Status** In Device mode, this bit is set as long as a Suspend condition is detected on the USB. The core enters the Suspended state when there is no activity on the bus for an extended period of time. The core comes out of the suspend when there is any activity on the bus or when the application writes to the Remote Wakeup Signaling bit in the Device Control register (USB_DCTL.RMTWKUPSIG).

15.6.24 USB_DIEPMSK - Device IN Endpoint Common Interrupt Mask Register

This register works with each of the Device IN Endpoint Interrupt (USB_DIEP0INT/USB_DIEPx_INT) registers for all endpoints to generate an interrupt per IN endpoint. The IN endpoint interrupt for a specific status in the USB_DIEP0INT/USB_DIEPx_INT register can be masked by writing to the corresponding bit in this register. Status bits are masked by default.

Offset	Bit Position																														
0x3C810	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																			0					0		0		0	0	0	0	0
Access																			RW					RW		RW		RW	RW	RW	RW	RW
Name																			NAKMSK					TXFIFOUNDRNMSK		INEPNAKEFFMSK		INTKNTXFEMPMSK	TIMEOUTMSK	AHBERRMSK	EPDISBLDMSK	XFERCOMPLMSK

Bit	Name	Reset	Access	Description
31:14	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
13	NAKMSK	0	RW	**NAK interrupt Mask** Set to 1 to unmask NAK Interrupt.
12:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8	TXFIFOUNDRNMSK	0	RW	**Fifo Underrun Mask** Set to 1 to unmask TXFIFOUNDRN Interrupt.
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	INEPNAKEFFMSK	0	RW	**IN Endpoint NAK Effective Mask** Set to 1 to unmask INEPNAKEFF Interrupt.
5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	INTKNTXFEMPMSK	0	RW	**IN Token Received When TxFIFO Empty Mask** Set to 1 to unmask INTKNTXFEMP Interrupt.
3	TIMEOUTMSK	0	RW	**Timeout Condition Mask** Set to 1 to unmask Interrupt TIMEOUT. Applies to Non-isochronous endpoints.
2	AHBERRMSK	0	RW	**AHB Error Mask** Set to 1 to unmask AHBERR Interrupt.
1	EPDISBLDMSK	0	RW	**Endpoint Disabled Interrupt Mask** Set to 1 to unmask EPDISBLD Interrupt.
0	XFERCOMPLMSK	0	RW	**Transfer Completed Interrupt Mask** Set to 1 to unmask XFERCOMPL Interrupt.

15.6.25 USB_DOEPMSK - Device OUT Endpoint Common Interrupt Mask Register

This register works with each of the Device OUT Endpoint Interrupt (USB_DOEP0INT/ USB_DOEPx_INT) registers for all endpoints to generate an interrupt per OUT endpoint. The OUT endpoint interrupt for a specific status in the USB_DOEP0INT/USB_DOEPx_INT register can be masked by writing into the corresponding bit in this register. Status bits are masked by default.

Offset	Bit Position																													
0x3C814	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																			0	0				0		0	0	0	0	0	0	0
Access																			RW	RW				RW		RW	RW	RW	RW	RW	RW	RW
Name																			NAKMSK	BBLEERRMSK				OUTPKTERRMSK		BACK2BACKSETUP	STSPHSERCVDMSK	OUTTKNEPDISMSK	SETUPMSK	AHBERRMSK	EPDISBLDMSK	XFERCOMPLMSK

Bit	Name	Reset	Access	Description
31:14	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
13	NAKMSK	0	RW	**NAK interrupt Mask**
	Set to 1 to unmask NAK Interrupt.			
12	BBLEERRMSK	0	RW	**Babble Error interrupt Mask**
	Set to 1 to unmask BBLEERR Interrupt.			
11:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8	OUTPKTERRMSK	0	RW	**OUT Packet Error Mask**
	Set to 1 to unmask OUTPKTERR Interrupt.			
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	BACK2BACKSETUP	0	RW	**Back-to-Back SETUP Packets Received Mask**
	Set to 1 to unmask BACK2BACKSETUP Interrupt. Applies to control OUT endpoints only.			
5	STSPHSERCVDMSK	0	RW	**Status Phase Received Mask**
	Set to 1 to unmask STSPHSERCVD Interrupt. Applies to control OUT endpoints only.			
4	OUTTKNEPDISMSK	0	RW	**OUT Token Received when Endpoint Disabled Mask**
	Set to 1 to unmask OUTTKNEPDIS Interrupt. Applies to control OUT endpoints only.			
3	SETUPMSK	0	RW	**SETUP Phase Done Mask**
	Set to 1 to unmask SETUP Interrupt. Applies to control endpoints only.			
2	AHBERRMSK	0	RW	**AHB Error**
	Set to 1 to unmask AHBERR Interrupt.			
1	EPDISBLDMSK	0	RW	**Endpoint Disabled Interrupt Mask**
	Set to 1 to unmask EPDISBLD Interrupt.			
0	XFERCOMPLMSK	0	RW	**Transfer Completed Interrupt Mask**
	Set to 1 to unmask XFERCOMPL Interrupt.			

15.6.26 USB_DAINT - Device All Endpoints Interrupt Register

When a significant event occurs on an endpoint, a Device All Endpoints Interrupt register interrupts the application using the Device OUT Endpoints Interrupt bit or Device IN Endpoints Interrupt bit of

the Core Interrupt register (USB_GINTSTS.OEPINT or USB_GINTSTS.IEPINT, respectively). There is one interrupt bit per endpoint. For a bidirectional endpoint, the corresponding IN and OUT interrupt bits are used. Bits in this register are set and cleared when the application sets and clears bits in the corresponding Device Endpoint Interrupt register (USB_DIEP0INT/USB_DIEPx_INT, USB_DOEP0INT/ USB_DOEPx_INT).

Offset	Bit Position																															
0x3C818	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset													0	0	0	0													0	0	0	0
Access													R	R	R	R													R	R	R	R
Name													OUTEPINT3	OUTEPINT2	OUTEPINT1	OUTEPINT0													INEPINT3	INEPINT2	INEPINT1	INEPINT0

Bit	Name	Reset	Access	Description
31:20	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
19	OUTEPINT3	0	R	**OUT Endpoint 3 Interrupt Bit**
	This bit is set when one or more of the interrupt flags in USB_DOEP2_INT are set.			
18	OUTEPINT2	0	R	**OUT Endpoint 2 Interrupt Bit**
	This bit is set when one or more of the interrupt flags in USB_DOEP1_INT are set.			
17	OUTEPINT1	0	R	**OUT Endpoint 1 Interrupt Bit**
	This bit is set when one or more of the interrupt flags in USB_DOEP0_INT are set.			
16	OUTEPINT0	0	R	**OUT Endpoint 0 Interrupt Bit**
	This bit is set when one or more of the interrupt flags in USB_DOEP0INT are set.			
15:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	INEPINT3	0	R	**IN Endpoint 3 Interrupt Bit**
	This bit is set when one or more of the interrupt flags in USB_DIEP2_INT are set.			
2	INEPINT2	0	R	**IN Endpoint 2 Interrupt Bit**
	This bit is set when one or more of the interrupt flags in USB_DIEP1_INT are set.			
1	INEPINT1	0	R	**IN Endpoint 1 Interrupt Bit**
	This bit is set when one or more of the interrupt flags in USB_DIEP0_INT are set.			
0	INEPINT0	0	R	**IN Endpoint 0 Interrupt Bit**
	This bit is set when one or more of the interrupt flags in USB_DIEP0INT are set.			

15.6.27 USB_DAINTMSK - Device All Endpoints Interrupt Mask Register

The Device Endpoint Interrupt Mask register works with the Device Endpoint Interrupt register to interrupt the application when an event occurs on a device endpoint. However, the Device All Endpoints Interrupt (USB_DAINT) register bit corresponding to that interrupt is still set.

Offset	Bit Position																															
0x3C81C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset													0	0	0	0													0	0	0	0
Access													RW	RW	RW	RW													RW	RW	RW	RW
Name													OUTEPMSK3	OUTEPMSK2	OUTEPMSK1	OUTEPMSK0													INEPMSK3	INEPMSK2	INEPMSK1	INEPMSK0

Bit	Name	Reset	Access	Description
31:20	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
19	OUTEPMSK3	0	RW	**OUT Endpoint 3 Interrupt mask Bit**
	Set to 1 to unmask USB_DAINT.OUTEPINT3.			
18	OUTEPMSK2	0	RW	**OUT Endpoint 2 Interrupt mask Bit**
	Set to 1 to unmask USB_DAINT.OUTEPINT2.			
17	OUTEPMSK1	0	RW	**OUT Endpoint 1 Interrupt mask Bit**
	Set to 1 to unmask USB_DAINT.OUTEPINT1.			
16	OUTEPMSK0	0	RW	**OUT Endpoint 0 Interrupt mask Bit**
	Set to 1 to unmask USB_DAINT.OUTEPINT0.			
15:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	INEPMSK3	0	RW	**IN Endpoint 3 Interrupt mask Bit**
	Set to 1 to unmask USB_DAINT.INEPINT3.			
2	INEPMSK2	0	RW	**IN Endpoint 2 Interrupt mask Bit**
	Set to 1 to unmask USB_DAINT.INEPINT2.			
1	INEPMSK1	0	RW	**IN Endpoint 1 Interrupt mask Bit**
	Set to 1 to unmask USB_DAINT.INEPINT1.			
0	INEPMSK0	0	RW	**IN Endpoint 0 Interrupt mask Bit**
	Set to 1 to unmask USB_DAINT.INEPINT0.			

15.6.28 USB_DIEPEMPMSK - Device IN Endpoint FIFO Empty Interrupt Mask Register

This register is used to control the IN endpoint FIFO empty interrupt generation (USB_DIEP0INT/ USB_DIEPx_INT.TXFEMP).

Offset	Bit Position																													
0x3C834	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	DIEPEMPMSK															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	DIEPEMPMSK	0x0000	RW	**IN EP Tx FIFO Empty Interrupt Mask Bits**
	These bits acts as mask bits for USB_DIEP0INT.TXFEMP/USB_DIEPx_INT.TXFEMP interrupt. One bit per IN Endpoint: Bit 0 for IN EP 0, bit 6 for IN EP 6.			

15.6.29 USB_DIEP0CTL - Device IN Endpoint 0 Control Register

This section describes the Control IN Endpoint 0 Control register. Nonzero control endpoints use registers for endpoints 1 - 6.

Offset	Bit Position																															
0x3C900	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0			0	0	0x0				0		0x0		0		1														0x0	
Access	RW1H	RW1H			W1	W1	RW				RW1H		R		R		R														RW	
Name	EPENA	EPDIS			SNAK	CNAK	TXFNUM				STALL		EPTYPE		NAKSTS		USBACTEP														MPS	

Bit	Name	Reset	Access	Description
31	EPENA	0	RW1H	**Endpoint Enable**
	In DMA mode this bit indicates that data is ready to be transmitted on the endpoint. The core clears this bit before setting the following interrupts on this endpoint: Endpoint Disabled, Transfer Completed.			
30	EPDIS	0	RW1H	**Endpoint Disable**
	The application sets this bit to stop transmitting data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the Endpoint Disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the Endpoint Disabled Interrupt. The application must set this bit only if Endpoint Enable is already set for this endpoint.			
29:28	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
27	SNAK	0	W1	**Set NAK**
	A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for an endpoint after a SETUP packet is received on that endpoint.			
26	CNAK	0	W1	**Clear NAK**
	A write to this bit clears the NAK bit for the endpoint.			
25:22	TXFNUM	0x0	RW	**TxFIFO Number**
	This value is set to the FIFO number that is assigned to IN Endpoint 0.			
21	STALL	0	RW1H	**Handshake**
	The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit, Global Nonperiodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority.			
20	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
19:18	EPTYPE	0x0	R	**Endpoint Type**
	Hardcoded to 0. Endpoint 0 is always a control endpoint.			
17	NAKSTS	0	R	**NAK Status**
	When this bit is 0 the core is transmitting non-NAK handshakes based on the FIFO status. When this bit is 1 the core is transmitting NAK handshakes on this endpoint. When this bit is set, either by the application or core, the core stops transmitting data, even if there is data available in the TxFIFO. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.			
16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15	USBACTEP	1	R	**USB Active Endpoint**
	This bit is always 1, indicating that control endpoint 0 is always active in all configurations and interfaces.			
14:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1:0	MPS	0x0	RW	**Maximum Packet Size**
	The application must program this field with the maximum packet size for the current logical endpoint.			

Value	Mode	Description
0	64B	64 bytes.
1	32B	32 bytes.
2	16B	16 bytes.
3	8B	8 bytes.

15.6.30 USB_DIEP0INT - Device IN Endpoint 0 Interrupt Register

This register indicates the status of endpoint 0 with respect to USB- and AHB-related events. The application must read this register when the IN Endpoints Interrupt bit of the Core Interrupt register (USB_GINTSTS.IEPINT) is set. Before the application can read this register, it must first read the Device All Endpoints Interrupt (USB_DAINT) register to get the exact endpoint number for the Device Endpoint Interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the USB_DAINT and USB_GINTSTS registers.

Offset	Bit Position																														
0x3C908	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																			0	0	0				1	0		0	0	0	0	0
Access																			RW1H	RW1H	RW1H				R	RW1H		RW1H	RW1H	RW1H	RW1H	RW1H
Name																			NAKINTRPT	BBLEERR	PKTDRPSTS				TXFEMP	INEPNAKEFF		INTKNTXFEMP	TIMEOUT	AHBERR	EPDISBLD	XFERCOMPL

Bit	Name	Reset	Access	Description
31:14	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
13	NAKINTRPT	0	RW1H	**NAK Interrupt**
	The core generates this interrupt when a NAK is transmitted or received by the device. In case of isochronous IN endpoints the interrupt gets generated when a zero length packet is transmitted due to un-availability of data in the TXFifo.			
12	BBLEERR	0	RW1H	**NAK Interrupt**
	The core generates this interrupt when babble is received for the endpoint.			
11	PKTDRPSTS	0	RW1H	**Packet Drop Status**
	This bit indicates to the application that an ISO OUT packet has been dropped. This bit does not have an associated mask bit and does not generate an interrupt.			
10:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7	TXFEMP	1	R	**Transmit FIFO Empty**
	This interrupt is asserted when the TxFIFO for this endpoint is either half or completely empty. The half or completely empty status is determined by the TxFIFO Empty Level bit in the Core AHB Configuration register (USB_GAHBCFG.NPTXFEMPLVL).			
6	INEPNAKEFF	0	RW1H	**IN Endpoint NAK Effective**
	Applies to periodic IN endpoints only. This bit can be cleared when the application clears the IN endpoint NAK by writing to USB_DIEP0CTL.CNAK. This interrupt indicates that the core has sampled the NAK bit set (either by the application or by the core). The interrupt indicates that the IN endpoint NAK bit set by the application has taken effect in the core. This interrupt does not guarantee that a NAK handshake is sent on the USB. A STALL bit takes priority over a NAK bit.			
5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	INTKNTXFEMP	0	RW1H	**IN Token Received When TxFIFO is Empty**
	Indicates that an IN token was received when the associated TxFIFO (periodic/non-periodic) was empty. This interrupt is asserted on the endpoint for which the IN token was received.			
3	TIMEOUT	0	RW1H	**Timeout Condition**
	Indicates that the core has detected a timeout condition on the USB for the last IN token on this endpoint.			
2	AHBERR	0	RW1H	**AHB Error**
	This is generated in DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding endpoint DMA address register to get the error address.			
1	EPDISBLD	0	RW1H	**Endpoint Disabled Interrupt**
	This bit indicates that the endpoint is disabled per the application's request.			
0	XFERCOMPL	0	RW1H	**Transfer Completed Interrupt**
	This field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.			

15.6.31 USB_DIEP0TSIZ - Device IN Endpoint 0 Transfer Size Register

The application must modify this register before enabling endpoint 0. Once endpoint 0 is enabled using Endpoint Enable bit of the Device Control Endpoint 0 Control register (USB_DIEP0CTL.EPENA), the core modifies this register. The application can only read this register once the core has cleared the Endpoint Enable bit. Nonzero endpoints use the registers for endpoints 1-6.

Offset	Bit Position																														
0x3C910	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset												0x0																	0x00			
Access												RW																	RW			
Name												PKTCNT																	XFERSIZE			

Bit	Name	Reset	Access	Description
31:21	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
20:19	PKTCNT	0x0	RW	**Packet Count**
	Indicates the total number of USB packets that constitute the Transfer Size amount of data for endpoint 0. This field is decremented every time a packet (maximum size or short packet) is read from the TxFIFO.			
18:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:0	XFERSIZE	0x00	RW	**Transfer Size**
	Indicates the transfer size in bytes for endpoint 0. The core interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. The core decrements this field every time a packet from the external memory is written to the TxFIFO.			

15.6.32 USB_DIEP0DMAADDR - Device IN Endpoint 0 DMA Address Register

Offset	Bit Position																														
0x3C914	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																0xXXXXXXXX																
Access																RW																
Name																DIEP0DMAADDR																

Bit	Name	Reset	Access	Description
31:0	DIEP0DMAADDR	0xXXXXXXXX	RW	**DMA Address**
	Holds the start address of the external memory for fetching endpoint data. For control endpoints, this field stores control OUT data packets as well as SETUP transaction data packets. When more than three SETUP packets are received back-to-back, the SETUP data packet in the memory is overwritten. This register is incremented on every AHB transaction. The application can give only a DWORD-aligned address. The data for this register field is stored in RAM. Thus, the reset value is undefined (X).			

15.6.33 USB_DIEP0TXFSTS - Device IN Endpoint 0 Transmit FIFO Status Register

This read-only register contains the free space information for the Device IN endpoint 0 TxFIFO.

Offset	Bit Position																															
0x3C918	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0200															
Access																	R															
Name																	SPCAVAIL															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	SPCAVAIL	0x0200	R	**TxFIFO Space Available**
	Indicates the amount of free space available in the Endpoint TxFIFO. Values are in terms of 32-bit words.			

15.6.34 USB_DIEPx_CTL - Device IN Endpoint x+1 Control Register

The application uses this register to control the behavior of each logical endpoint other than endpoint 0.

Offset	Bit Position																															
0x3C920	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0x0				0		0x0		0	0	0					0x000										
Access	RW1H	RW1H	W1	W1	W1	W1	RW				RW1H		RW		R	R	RW					RW										
Name	EPENA	EPDIS	SETD1PIDOF	SETD0PIDEF	SNAK	CNAK	TXFNUM				STALL		EPTYPE		NAKSTS	DPIDEOF	USBACTEP					MPS										

Bit	Name	Reset	Access	Description
31	EPENA	0	RW1H	**Endpoint Enable**
	In DMA mode for IN endpoints, this bit indicates that data is ready to be transmitted on the endpoint. The core clears this bit before setting any of the following interrupts on this endpoint: SETUP Phase Done, Endpoint Disabled, Transfer Completed. For control endpoints in DMA mode, this bit must be set to be able to transfer SETUP data packets in memory.			
30	EPDIS	0	RW1H	**Endpoint Disable**
	The application sets this bit to stop transmitting/receiving data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the Endpoint Disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the Endpoint Disabled interrupt. The application must set this bit only if Endpoint Enable is already set for this endpoint.			
29	SETD1PIDOF	0	W1	**Set DATA1 PID / Odd Frame**
	For bulk and interrupt endpoints writing this field sets the Endpoint Data PID / Even or Odd Frame (DPIDEOF) field in this register to DATA1ODD.			
	For isochronous endpoints writing this field sets the Endpoint Data PID / Even or Odd Frame (DPIDEOF) field to odd (DATA1ODD).			
28	SETD0PIDEF	0	W1	**Set DATA0 PID / Even Frame**
	For bulk and interrupt endpoints writing this field sets the Endpoint Data PID / Even or Odd Frame (DPIDEOF) field in this register to DATA0EVEN.			

Bit	Name	Reset	Access	Description
	For isochronous endpoints writing this field sets the Endpoint Data PID / Even or Odd Frame (DPIDEOF) field to odd (DATA0EVEN).			
27	SNAK	0	W1	**Set NAK**
	A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for an endpoint after a SETUP packet is received on that endpoint.			
26	CNAK	0	W1	**Clear NAK**
	A write to this bit clears the NAK bit for the endpoint.			
25:22	TXFNUM	0x0	RW	**TxFIFO Number**
	These bits specify the FIFO number associated with this endpoint. Each active IN endpoint must be programmed to a separate FIFO number. This field is valid only for IN endpoints.			
21	STALL	0	RW1H	**Handshake**
	For bulk and interrupt endpoints: The application sets this bit to stall all tokens from the USB host to this endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. In this case only the application can clear this bit, never the core. When control endpoint: The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.			
20	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
19:18	EPTYPE	0x0	RW	**Endpoint Type**
	This is the transfer type supported by this logical endpoint.			

Value	Mode	Description
0	CONTROL	Control Endpoint.
1	ISO	Isochronous Endpoint.
2	BULK	Bulk Endpoint.
3	INT	Interrupt Endpoint.

Bit	Name	Reset	Access	Description
17	NAKSTS	0	R	**NAK Status**
	When this bit is 0 the core is transmitting non-NAK handshakes based on the FIFO status. When this bit is 1 the core is transmitting NAK handshakes on this endpoint. When either the application or the core sets this bit the core stops receiving any data on an OUT endpoint, even if there is space in the RxFIFO to accommodate the incoming packet. For non-isochronous IN endpoints the core stops transmitting any data on an IN endpoint, even if there data is available in the TxFIFO. For isochronous IN endpoints the core sends out a zero-length data packet, even if there data is available in the TxFIFO. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.			
16	DPIDEOF	0	R	**Endpoint Data PID / Even or Odd Frame**
	For interrupt/bulk endpoints this field contains the PID of the packet to be received or transmitted on this endpoint. The application must program the PID of the first packet to be received or transmitted on this endpoint, after the endpoint is activated. The applications use the SETD1PIDOF and SETD0PIDEF fields of this register to program either DATA0 or DATA1 PID. For isochronous endpoints, this field indicates the frame number in which the core transmits/receives isochronous data for this endpoint. The application must program the even/odd frame number in which it intends to transmit/receive isochronous data for this endpoint using the SETD0PIDEF and SETD1PIDOF fields in this register.			

Value	Mode	Description
0	DATA0EVEN	DATA0 PID / Even Frame.
1	DATA1ODD	DATA1 PID / Odd Frame.

Bit	Name	Reset	Access	Description
15	USBACTEP	0	RW	**USB Active Endpoint**
	Indicates whether this endpoint is active in the current configuration and interface. The core clears this bit for all endpoints after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program endpoint registers accordingly and set this bit.			
14:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:0	MPS	0x000	RW	**Maximum Packet Size**
	The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.			

15.6.35 USB_DIEPx_INT - Device IN Endpoint x+1 Interrupt Register

This register indicates the status of an endpoint with respect to USB- and AHB-related events. The application must read this register when the IN Endpoints Interrupt bit of the Core Interrupt register (USB_GINTSTS.IEPINT) is set. Before the application can read this register, it must first read the

Device All Endpoints Interrupt (USB_DAINT) register to get the exact endpoint number for the Device Endpoint x+1 Interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the USB_DAINT and USB_GINTSTS registers.

Offset	Bit Position																													
0x3C928	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																			0	0	0				1	0		0	0	0	0	0
Access																			RW1H	RW1H	RW1H				R	RW1H		RW1H	RW1H	RW1H	RW1H	RW1H
Name																			NAKINTRPT	BBLEERR	PKTDRPSTS				TXFEMP	INEPNAKEFF		INTKNTXFEMP	TIMEOUT	AHBERR	EPDISBLD	XFERCOMPL

Bit	Name	Reset	Access	Description
31:14	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
13	NAKINTRPT	0	RW1H	**NAK Interrupt**
	The core generates this interrupt when a NAK is transmitted or received by the device. In case of isochronous IN endpoints the interrupt gets generated when a zero length packet is transmitted due to un-availability of data in the TXFifo.			
12	BBLEERR	0	RW1H	**NAK Interrupt**
	The core generates this interrupt when babble is received for the endpoint.			
11	PKTDRPSTS	0	RW1H	**Packet Drop Status**
	This bit indicates to the application that an ISO OUT packet has been dropped. This bit does not have an associated mask bit and does not generate an interrupt.			
10:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7	TXFEMP	1	R	**Transmit FIFO Empty**
	This interrupt is asserted when the TxFIFO for this endpoint is either half or completely empty. The half or completely empty status is determined by the TxFIFO Empty Level bit in the Core AHB Configuration register (USB_GAHBCFG.NPTXFEMPLVL).			
6	INEPNAKEFF	0	RW1H	**IN Endpoint NAK Effective**
	Applies to periodic IN endpoints only. This bit can be cleared when the application clears the IN endpoint NAK by writing to USB_DIEPx_CTL.CNAK. This interrupt indicates that the core has sampled the NAK bit set (either by the application or by the core). The interrupt indicates that the IN endpoint NAK bit set by the application has taken effect in the core. This interrupt does not guarantee that a NAK handshake is sent on the USB. A STALL bit takes priority over a NAK bit.			
5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	INTKNTXFEMP	0	RW1H	**IN Token Received When TxFIFO is Empty**
	Applies to non-periodic IN endpoints only. Indicates that an IN token was received when the associated TxFIFO (periodic/non-periodic) was empty. This interrupt is asserted on the endpoint for which the IN token was received.			
3	TIMEOUT	0	RW1H	**Timeout Condition**
	Applies only to Control IN endpoints. Indicates that the core has detected a timeout condition on the USB for the last IN token on this endpoint.			
2	AHBERR	0	RW1H	**AHB Error**
	This is generated only in DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding endpoint DMA address register to get the error address.			
1	EPDISBLD	0	RW1H	**Endpoint Disabled Interrupt**
	This bit indicates that the endpoint is disabled per the application's request.			
0	XFERCOMPL	0	RW1H	**Transfer Completed Interrupt**
	This field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.			

15.6.36 USB_DIEPx_TSIZ - Device IN Endpoint x+1 Transfer Size Register

The application must modify this register before enabling the endpoint. Once the endpoint is enabled using Endpoint Enable bit of the Device Endpoint x+1 Control register (USB_DIEPx_CTL.EPENA), the core modifies this register. The application can only read this register once the core has cleared the Endpoint Enable bit.

Offset	Bit Position																														
0x3C930	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset		0x0						0x000															0x00000									
Access		RW						RW															RW									
Name		MC						PKTCNT															XFERSIZE									

Bit	Name	Reset	Access	Description
31	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
30:29	MC	0x0	RW	**Multi Count**
	For periodic IN endpoints, this field indicates the number of packets that must be transmitted per frame on the USB. The core uses this field to calculate the data PID for isochronous IN endpoints.			
28:19	PKTCNT	0x000	RW	**Packet Count**
	Indicates the total number of USB packets that constitute the Transfer Size amount of data. This field is decremented every time a packet (maximum size or short packet) is read from the TxFIFO.			
18:0	XFERSIZE	0x00000	RW	**Transfer Size**
	Indicates the transfer size in bytes. The core interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. The core decrements this field every time a packet from the external memory is written to the TxFIFO.			

15.6.37 USB_DIEPx_DMAADDR - Device IN Endpoint x+1 DMA Address Register

Offset	Bit Position																														
0x3C934	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0xXXXXXXXX																															
Access	RW																															
Name	DMAADDR																															

Bit	Name	Reset	Access	Description
31:0	DMAADDR	0xXXXXXXXX	RW	**DMA Address**
	Holds the start address of the external memory for fetching endpoint data. For control endpoints, this field stores control OUT data packets as well as SETUP transaction data packets. When more than three SETUP packets are received back-to-back, the SETUP			

Bit	Name	Reset	Access	Description
	data packet in the memory is overwritten. This register is incremented on every AHB transaction. The application can give only a DWORD-aligned address. The data for this register field is stored in RAM. Thus, the reset value is undefined (X).			

15.6.38 USB_DIEPx_TXFSTS - Device IN Endpoint x+1 Transmit FIFO Status Register

This read-only register contains the free space information for the Device IN endpoint TxFIFO.

Offset	Bit Position																													
0x3C938	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0200															
Access																	R															
Name																	SPCAVAIL															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	SPCAVAIL	0x0200	R	**TxFIFO Space Available**
	Indicates the amount of free space available in the Endpoint TxFIFO. Values are in terms of 32-bit words.			

15.6.39 USB_DOEP0CTL - Device OUT Endpoint 0 Control Register

The application uses this register to control the behavior of each logical endpoint other than endpoint 0.

Offset	Bit Position																													
0x3CB00	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0			0	0					0	0	0x0		0		1														0x0	
Access	RW1H	R			W1	W1					RW1H	RW	R		R		R														R	
Name	EPENA	EPDIS			SNAK	CNAK					STALL	SNP	EPTYPE		NAKSTS		USBACTEP														MPS	

Bit	Name	Reset	Access	Description
31	EPENA	0	RW1H	**Endpoint Enable**
	In DMA mode this bit indicates that the application has allocated the memory to start receiving data from the USB. The core clears this bit before setting any of the following interrupts on this endpoint: SETUP Phase Done, Endpoint Disabled, Transfer Completed. In DMA mode, this bit must be set for the core to transfer SETUP data packets into memory.			
30	EPDIS	0	R	**Endpoint Disable**
	This bit is always 0. The application cannot disable control OUT endpoint 0.			
29:28	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
27	SNAK	0	W1	**Set NAK**
	A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set bit on a Transfer Completed interrupt, or after a SETUP is received on the endpoint.			
26	CNAK	0	W1	**Clear NAK**
	A write to this bit clears the NAK bit for the endpoint.			

Bit	Name	Reset	Access	Description
25:22	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
21	STALL	0	RW1H	**Handshake**
	The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.			
20	SNP	0	RW	**Snoop Mode**
	This bit configures the endpoint to Snoop mode. In Snoop mode, the core does not check the correctness of OUT packets before transferring them to application memory.			
19:18	EPTYPE	0x0	R	**Endpoint Type**
	Hardcoded to 0. Endpoint 0 is always a control endpoint.			
17	NAKSTS	0	R	**NAK Status**
	When this bit is 0 the core is transmitting non-NAK handshakes based on the FIFO status. When this bit is 1 the core is transmitting NAK handshakes on this endpoint. When either the application or the core sets this bit, the core stops receiving data, even if there is space in the RxFIFO to accommodate the incoming packet. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.			
16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15	USBACTEP	1	R	**USB Active Endpoint**
	This bit is always 1, indicating that a control endpoint 0 is always active in all configurations and interfaces.			
14:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1:0	MPS	0x0	R	**Maximum Packet Size**
	The maximum packet size for control OUT endpoint 0 is the same as what is programmed in control IN Endpoint 0.			

Value	Mode	Description
0	64B	64 bytes.
1	32B	32 bytes.
2	16B	16 bytes.
3	8B	8 bytes.

15.6.40 USB_DOEP0INT - Device OUT Endpoint 0 Interrupt Register

This register indicates the status of endpoint 0 with respect to USB- and AHB-related events. The application must read this register when the OUT Endpoints Interrupt bit of the Core Interrupt register (USB_GINTSTS.OEPINT) is set. Before the application can read this register, it must first read the Device All Endpoints Interrupt (USB_DAINT) register to get the exact endpoint number for the Device Endpoint Interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the USB_DAINT and USB_GINTSTS registers.

Offset	Bit Position																														
0x3CB08	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0		0	0	0					0	0	0	0	0	0	0
Access																	RW		RW1H	RW1H	RW1H					RW1H	RW1H	RW1H	RW1H	RW1H	RW1H	RW1H
Name																	STUPPKTRCVD		NAKINTRPT	BBLEERR	PKTDRPSTS					BACK2BACKSETUP	STSPHSERCVD	OUTTKNEPDIS	SETUP	AHBERR	EPDISBLD	XFERCOMPL

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
15	STUPPKTRCVD	0	RW	**Setup Packet Received**
	The core generates this interrupt when a setup packet is received.			
14	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
13	NAKINTRPT	0	RW1H	**NAK Interrupt**
	The core generates this interrupt when a NAK is transmitted or received by the device. In case of isochronous IN endpoints the interrupt gets generated when a zero length packet is transmitted due to un-availability of data in the TXFifo.			
12	BBLEERR	0	RW1H	**NAK Interrupt**
	The core generates this interrupt when babble is received for the endpoint.			
11	PKTDRPSTS	0	RW1H	**Packet Drop Status**
	This bit indicates to the application that an ISO OUT packet has been dropped. This bit does not have an associated mask bit and does not generate an interrupt.			
10:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	BACK2BACKSETUP	0	RW1H	**Back-to-Back SETUP Packets Received**
	This bit indicates that the core has received more than three back-to-back SETUP packets for this particular endpoint.			
5	STSPHSERCVD	0	RW1H	**Status Phase Received For Control Write**
	This interrupt is valid only for Control OUT endpoints and only in Scatter Gather DMA mode. This interrupt is generated only after the core has transferred all the data that the host has sent during the data phase of a control write transfer, to the system memory buffer. The interrupt indicates to the application that the host has switched from data phase to the status phase of a Control Write transfer. The application can use this interrupt to ACK or STALL the Status phase, after it has decoded the data phase. This is applicable only in Case of Scatter Gather DMA mode.			
4	OUTTKNEPDIS	0	RW1H	**OUT Token Received When Endpoint Disabled**
	Indicates that an OUT token was received when the endpoint was not yet enabled. This interrupt is asserted on the endpoint for which the OUT token was received.			
3	SETUP	0	RW1H	**Setup Phase Done**
	Indicates that the SETUP phase for the control endpoint is complete and no more back-to-back SETUP packets were received for the current control transfer. On this interrupt, the application can decode the received SETUP data packet.			
2	AHBERR	0	RW1H	**AHB Error**
	This is generated only in DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding endpoint DMA address register to get the error address.			
1	EPDISBLD	0	RW1H	**Endpoint Disabled Interrupt**
	This bit indicates that the endpoint is disabled per the application's request.			
0	XFERCOMPL	0	RW1H	**Transfer Completed Interrupt**
	This field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.			

15.6.41 USB_DOEP0TSIZ - Device OUT Endpoint 0 Transfer Size Register

The application must modify this register before enabling the endpoint. Once the endpoint is enabled using Endpoint Enable bit of the Device Endpoint x+1 Control register (USB_DOEPx_CTL.EPENA), the core modifies this register. The application can only read this register once the core has cleared the Endpoint Enable bit.

Offset	Bit Position																														
0x3CB10	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset		0x0											0													0x00						
Access		RW											RW													RW						
Name		SUPCNT											PKTCNT													XFERSIZE						

Bit	Name	Reset	Access	Description
31	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
30:29	SUPCNT	0x0	RW	**SETUP Packet Count**
	This field specifies the number of back-to-back SETUP data packets the endpoint can receive.			
28:20	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
19	PKTCNT	0	RW	**Packet Count**
	This field is decremented to zero after a packet is written into the RxFIFO.			
18:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:0	XFERSIZE	0x00	RW	**Transfer Size**
	Indicates the transfer size in bytes for endpoint 0. The core interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. The core decrements this field every time a packet is read from the RxFIFO and written to the external memory.			

15.6.42 USB_DOEP0DMAADDR - Device OUT Endpoint 0 DMA Address Register

Offset	Bit Position																														
0x3CB14	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0xXXXXXXXX																															
Access	RW																															
Name	DOEP0DMAADDR																															

Bit	Name	Reset	Access	Description
31:0	DOEP0DMAADDR	0xXXXXXXXX	RW	**DMA Address**
	Holds the start address of the external memory for storing endpoint data. For control endpoints, this field stores control OUT data packets as well as SETUP transaction data packets. When more than three SETUP packets are received back-to-back, the SETUP data packet in the memory is overwritten. This register is incremented on every AHB transaction. The application can give only a DWORD-aligned address. The data for this register field is stored in RAM. Thus, the reset value is undefined (X).			

15.6.43 USB_DOEPx_CTL - Device OUT Endpoint x+1 Control Register

The application uses this register to control the behavior of each logical endpoint other than endpoint 0.

Offset	Bit Position																															
0x3CB20	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0					0	0	0x0		0	0	0					0x000										
Access	RW1H	RW1H	W1	W1	W1	W1					RW1H	RW	RW		R	R	RW					RW										
Name	EPENA	EPDIS	SETD1PIDOF	SETD0PIDEF	SNAK	CNAK					STALL	SNP	EPTYPE		NAKSTS	DPIDEOF	USBACTEP					MPS										

Bit	Name	Reset	Access	Description
31	EPENA	0	RW1H	**Endpoint Enable**
	In DMA mode this bit indicates that the application has allocated the memory to start receiving data from the USB. The core clears this bit before setting any of the following interrupts on this endpoint: SETUP Phase Done, Endpoint Disabled, Transfer Completed. For control endpoints in DMA mode, this bit must be set to be able to transfer SETUP data packets in memory.			
30	EPDIS	0	RW1H	**Endpoint Disable**
	The application sets this bit to stop transmitting/receiving data on an endpoint, even before the transfer for that endpoint is complete. The application must wait for the Endpoint Disabled interrupt before treating the endpoint as disabled. The core clears this bit before setting the Endpoint Disabled interrupt. The application must set this bit only if Endpoint Enable is already set for this endpoint.			
29	SETD1PIDOF	0	W1	**Set DATA1 PID / Odd Frame**
	For bulk and interrupt endpoints writing this field sets the Endpoint Data PID / Even or Odd Frame (DPIDEOF) field in this register to DATA1ODD. For isochronous endpoints writing this field sets the Endpoint Data PID / Even or Odd Frame (DPIDEOF) field to odd (DATA1ODD).			
28	SETD0PIDEF	0	W1	**Set DATA0 PID / Even Frame**
	For bulk and interrupt endpoints writing this field sets the Endpoint Data PID / Even or Odd Frame (DPIDEOF) field in this register to DATA0EVEN. For isochronous endpoints writing this field sets the Endpoint Data PID / Even or Odd Frame (DPIDEOF) field to odd (DATA0EVEN).			
27	SNAK	0	W1	**Set NAK**
	A write to this bit sets the NAK bit for the endpoint. Using this bit, the application can control the transmission of NAK handshakes on an endpoint. The core can also set this bit for an endpoint after a SETUP packet is received on that endpoint.			
26	CNAK	0	W1	**Clear NAK**
	A write to this bit clears the NAK bit for the endpoint.			
25:22	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
21	STALL	0	RW1H	**STALL Handshake**
	For non-control, non-isochronous endpoints: The application sets this bit to stall all tokens from the USB host to this endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Only the application can clear this bit, never the core. For control endpoints: The application can only set this bit, and the core clears it, when a SETUP token is received for this endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.			
20	SNP	0	RW	**Snoop Mode**
	This bit configures the endpoint to Snoop mode. In Snoop mode, the core does not check the correctness of OUT packets before transferring them to application memory.			
19:18	EPTYPE	0x0	RW	**Endpoint Type**
	This is the transfer type supported by this logical endpoint.			

Value	Mode	Description
0	CONTROL	Control Endpoint.
1	ISO	Isochronous Endpoint.
2	BULK	Bulk Endpoint.
3	INT	Interrupt Endpoint.

17	NAKSTS	0	R	**NAK Status**

Bit	Name	Reset	Access	Description
	When this bit is 0 the core is transmitting non-NAK handshakes based on the FIFO status. When this bit is 1 the core is transmitting NAK handshakes on this endpoint. When either the application or the core sets this bit the core stops receiving any data on an OUT endpoint, even if there is space in the RxFIFO to accommodate the incoming packet. Irrespective of this bit's setting, the core always responds to SETUP data packets with an ACK handshake.			
16	DPIDEOF	0	R	**Endpoint Data PID / Even-odd Frame**
	For interrupt/bulk endpoints: Contains the PID of the packet to be received or transmitted on this endpoint. The application must program the PID of the first packet to be received or transmitted on this endpoint, after the endpoint is activated. The application use the SETD1PIDOF and SETD0PIDEF fields of this register to program either DATA0 or DATA1 PID. For isochronous endpoints: Indicates the frame number in which the core transmits/receives isochronous data for this endpoint. The application must program the even/odd frame number in which it intends to transmit/receive isochronous data for this endpoint using the SETD1PIDOF and SETD0PIDEF fields in this register.			

Value	Mode	Description
0	DATA0EVEN	DATA0 PID / Even Frame.
1	DATA1ODD	DATA1 PID / Odd Frame.

15	USBACTEP	0	RW	**USB Active Endpoint**
	Indicates whether this endpoint is active in the current configuration and interface. The core clears this bit for all endpoints after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program endpoint registers accordingly and set this bit.			
14:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:0	MPS	0x000	RW	**Maximum Packet Size**
	The application must program this field with the maximum packet size for the current logical endpoint. This value is in bytes.			

15.6.44 USB_DOEPx_INT - Device OUT Endpoint x+1 Interrupt Register

This register indicates the status of an endpoint with respect to USB- and AHB-related events. The application must read this register when the OUT Endpoints Interrupt bit of the Core Interrupt register (USB_GINTSTS.OEPINT) is set. Before the application can read this register, it must first read the Device All Endpoints Interrupt (USB_DAINT) register to get the exact endpoint number for the Device Endpoint Interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the USB_DAINT and USB_GINTSTS registers.

Offset	Bit Position																													
0x3CB28	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0		0	0	0					0	0	0	0	0	0	0
Access																	RW		RW1H	RW1H	RW1H					RW1H	RW1H	RW1H	RW1H	RW1H	RW1H	RW1H
Name																	STUPPKTRCVD		NAKINTRPT	BBLEERR	PKTDRPSTS					BACK2BACKSETUP	STSPHSERCVD	OUTTKNEPDIS	SETUP	AHBERR	EPDISBLD	XFERCOMPL

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15	STUPPKTRCVD	0	RW	**Setup Packet Received**
	The core generates this interrupt when a setup packet is received.			
14	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
13	NAKINTRPT	0	RW1H	**NAK Interrupt**
	The core generates this interrupt when a NAK is transmitted or received by the device.			
12	BBLEERR	0	RW1H	**Babble Error**

Bit	Name	Reset	Access	Description
	The core generates this interrupt when babble is received for the endpoint.			
11	PKTDRPSTS	0	RW1H	**Packet Drop Status**
	This bit indicates to the application that an ISO OUT packet has been dropped. This bit does not have an associated mask bit and does not generate an interrupt.			
10:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	BACK2BACKSETUP	0	RW1H	**Back-to-Back SETUP Packets Received**
	Applies to Control OUT endpoints only. This bit indicates that the core has received more than three back-to-back SETUP packets for this particular endpoint.			
5	STSPHSERCVD	0	RW1H	**Status Phase Received For Control Write**
	This interrupt is valid only for Control OUT endpoints and only in Scatter Gather DMA mode. This interrupt is generated only after the core has transferred all the data that the host has sent during the data phase of a control write transfer, to the system memory buffer. The interrupt indicates to the application that the host has switched from data phase to the status phase of a Control Write transfer. The application can use this interrupt to ACK or STALL the Status phase, after it has decoded the data phase. This is applicable only in Case of Scatter Gather DMA mode.			
4	OUTTKNEPDIS	0	RW1H	**OUT Token Received When Endpoint Disabled**
	Applies only to control OUT endpoints. Indicates that an OUT token was received when the endpoint was not yet enabled. This interrupt is asserted on the endpoint for which the OUT token was received.			
3	SETUP	0	RW1H	**Setup Phase Done**
	Applies to control OUT endpoints only. Indicates that the SETUP phase for the control endpoint is complete and no more back-to-back SETUP packets were received for the current control transfer. On this interrupt, the application can decode the received SETUP data packet.			
2	AHBERR	0	RW1H	**AHB Error**
	This is generated only in DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding endpoint DMA address register to get the error address.			
1	EPDISBLD	0	RW1H	**Endpoint Disabled Interrupt**
	This bit indicates that the endpoint is disabled per the application's request.			
0	XFERCOMPL	0	RW1H	**Transfer Completed Interrupt**
	This field indicates that the programmed transfer is complete on the AHB as well as on the USB, for this endpoint.			

15.6.45 USB_DOEPx_TSIZ - Device OUT Endpoint x+1 Transfer Size Register

The application must modify this register before enabling the endpoint. Once the endpoint is enabled using Endpoint Enable bit of the Device Endpoint x+1 Control register (USB_DOEPx_CTL.EPENA), the core modifies this register. The application can only read this register once the core has cleared the Endpoint Enable bit.

Offset	Bit Position																															
0x3CB30	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset		0x0		0x000										0x00000																		
Access		R		RW										RW																		
Name		RXDPIDSUPCNT		PKTCNT										XFERSIZE																		

Bit	Name	Reset	Access	Description
31	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
30:29	RXDPIDSUPCNT	0x0	R	**Receive Data PID / SETUP Packet Count**

For isochronous OUT endpoints: This is the data PID received in the last packet for this endpoint.

For control OUT Endpoints: This field specifies the number of back-to-back SETUP data packets the endpoint can receive.

Value	Mode	Description
0	DATA0	DATA0 PID.
1	DATA2	DATA2 PID / 1 Packet.
2	DATA1	DATA1 PID / 2 Packets.
3	MDATA	MDATA PID / 3 Packets.

Bit	Name	Reset	Access	Description
28:19	PKTCNT	0x000	RW	**Packet Count**

This field is decremented to zero after a packet is written into the RxFIFO.

Bit	Name	Reset	Access	Description
18:0	XFERSIZE	0x00000	RW	**Transfer Size**

Indicates the transfer size in bytes. The core interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the endpoint, to be interrupted at the end of each packet. The core decrements this field every time a packet is read from the RxFIFO and written to the external memory.

15.6.46 USB_DOEPx_DMAADDR - Device OUT Endpoint x+1 DMA Address Register

Offset	Bit Position																													
0x3CB34	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0xXXXXXXXX																															
Access	RW																															
Name	DMAADDR																															

Bit	Name	Reset	Access	Description
31:0	DMAADDR	0xXXXXXXXX	RW	**DMA Address**

Holds the start address of the external memory for storing endpoint data. For control endpoints, this field stores control OUT data packets as well as SETUP transaction data packets. When more than three SETUP packets are received back-to-back, the SETUP data packet in the memory is overwritten. This register is incremented on every AHB transaction. The application can give only a DWORD-aligned address. The data for this register field is stored in RAM. Thus, the reset value is undefined (X).

15.6.47 USB_PCGCCTL - Power and Clock Gating Control Register

This register is available in Host and Device modes. The application use this register to control the core's power-down and clock gating features.

Offset	Bit Position																															
0x3CE00	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																										0			0	0	0	0
Access																										R			RW	RW	RW	RW
Name																										PHYSLEEP			RSTPDWNMODULE	PWRCLMP	GATEHCLK	STOPPCLK

Bit	Name	Reset	Access	Description
31:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	PHYSLEEP	0	R	**PHY In Sleep**
	Indicates that the PHY is in Sleep State.			
5:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	RSTPDWNMODULE	0	RW	**Reset Power-Down Modules**
	The application sets this bit to reset the part of the USB that is powered down during EM2. The application clears this bit to release reset after an waking up from EM2 when the PHY clock is back at 48/6 MHz. Accessing core registers is possible only when this bit is set to 0.			
2	PWRCLMP	0	RW	**Power Clamp**
	The application sets this bit before the power is turned off to clamp the signals between the power-on modules and the power-off modules of the USB core. The application clears the bit to disable the clamping.			
1	GATEHCLK	0	RW	**Gate HCLK**
	The application sets this bit to gate the clock (HCLK) to modules other than the AHB Slave and Master and wakeup logic when the USB is suspended or the session is not valid. The application clears this bit when the USB is resumed or a new session starts.			
0	STOPPCLK	0	RW	**Stop PHY clock**
	The application sets this bit to stop the PHY clock when the USB is suspended, the session is not valid, or the device is disconnected. The application clears this bit when the USB is resumed or a new session starts.			

15.6.48 USB_FIFO0Dx - Device EP 0 FIFO

This register is used to read or write the FIFO space for endpoint 0, in a given direction.

Offset	Bit Position																															
0x3D000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																0xXXXXXXXX																
Access																RW																
Name																FIFO0D																

Bit	Name	Reset	Access	Description
31:0	FIFO0D	0xXXXXXXXX	RW	**Device EP 0**
	FIFO 0 push/pop region. Used in slave mode.			

15.6.49 USB_FIFO1Dx - Device EP 1 FIFO

This register is used to read or write the FIFO space for endpoint 1, in a given direction.

Offset	Bit Position																													
0x3E000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0xXXXXXXXX																															
Access	RW																															
Name	FIFO1D																															

Bit	Name	Reset	Access	Description
31:0	FIFO1D	0xXXXXXXXX	RW	**Device EP 1 FIFO**
	FIFO 1 push/pop region. Used in slave mode.			

15.6.50 USB_FIFO2Dx - Device EP 2 FIFO

This register is used to read or write the FIFO space for endpoint 2, in a given direction.

Offset	Bit Position																													
0x3F000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0xXXXXXXXX																															
Access	RW																															
Name	FIFO2D																															

Bit	Name	Reset	Access	Description
31:0	FIFO2D	0xXXXXXXXX	RW	**Device EP 2 FIFO**
	FIFO 2 push/pop region. Used in slave mode.			

15.6.51 USB_FIFO3Dx - Device EP 3 FIFO

This register is used to read or write the FIFO space for endpoint 3, in a given direction.

Offset	Bit Position																														
0x40000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0xXXXXXXXX																															
Access	RW																															
Name	FIFO3D																															

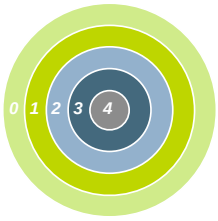
Bit	Name	Reset	Access	Description
31:0	FIFO3D	0xXXXXXXXX	RW	**Device EP 3 FIFO**
	FIFO 3 push/pop region. Used in slave mode.			

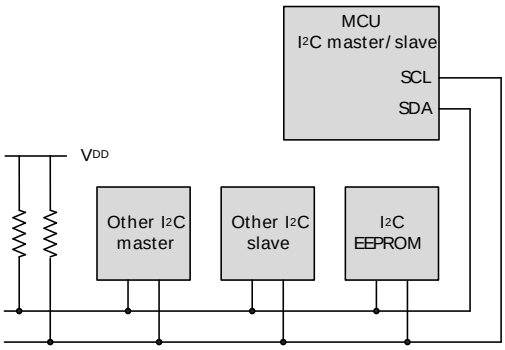
15.6.52 USB_FIFORAMx - Direct Access to Data FIFO RAM for Debugging (2 KB)

Offset	Bit Position																														
0x5C000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0xXXXXXXXX																															
Access	RW																															
Name	FIFORAM																															

Bit	Name	Reset	Access	Description
31:0	FIFORAM	0xXXXXXXXX	RW	**FIFO RAM**
	Direct Access to Data FIFO RAM for Debugging (2 KB)			

16 I²C - Inter-Integrated Circuit Interface





Quick Facts

What?

The I²C interface allows communication on I²C-buses with the lowest energy consumption possible.

Why?

I²C is a popular serial bus that enables communication with a number of external devices using only two I/O pins.

How?

With the help of DMA, the I²C interface allows I²C communication with minimal CPU intervention. Address recognition is available in all energy modes (except EM4), allowing the MCU to wait for data on the I²C-bus with sub-µA current consumption.

16.1 Introduction

The I²C module provides an interface between the MCU and a serial I²C-bus. It is capable of acting as both master and slave, and supports multi-master buses. Standard-mode, fast-mode and fast-mode plus speeds are supported, allowing transmission rates all the way from 10 kbit/s up to 1 Mbit/s. Slave arbitration and timeouts are also provided to allow implementation of an SMBus compliant system. The interface provided to software by the I²C module allows both fine-grained control of the transmission process and close to automatic transfers. Automatic recognition of slave addresses is provided in all energy modes (except EM4).

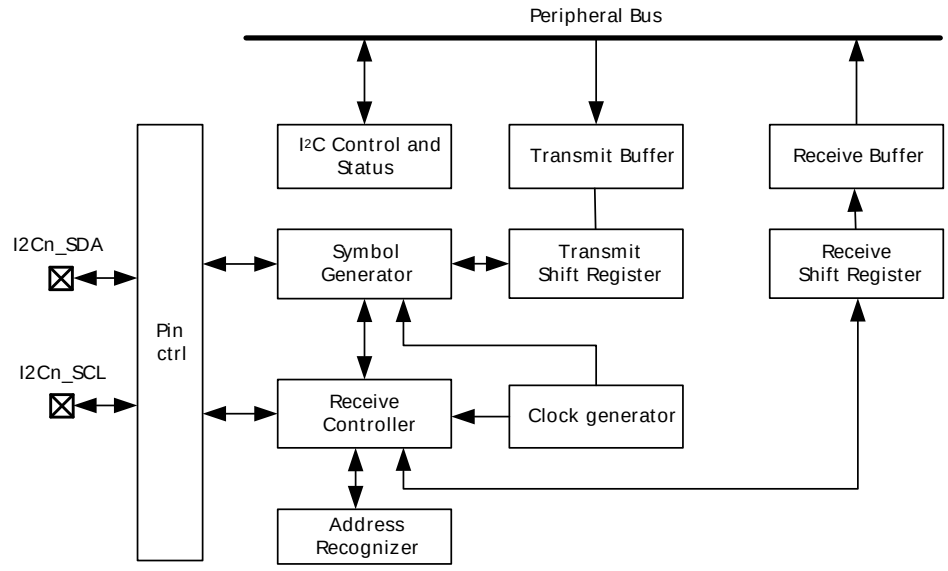
16.2 Features

- True multi-master capability
- Support for different bus speeds
  - Standard-mode (Sm) bit rate up to 100 kbit/s
  - Fast-mode (Fm) bit rate up to 400 kbit/s
  - Fast-mode Plus (Fm+) bit rate up to 1 Mbit/s
- Arbitration for both master and slave (allows SMBus ARP)
- Clock synchronization and clock stretching
- Hardware address recognition
  - 7-bit masked address
  - General call address
  - Active in all energy modes (except EM4)
- 10-bit address support
- Error handling
  - Clock low timeout
  - Clock high timeout
  - Arbitration lost
  - Bus error detection
- Double buffered data
- Full DMA support

16.3 Functional Description

An overview of the I2C module is shown in Figure 16.1 (p. 335) .

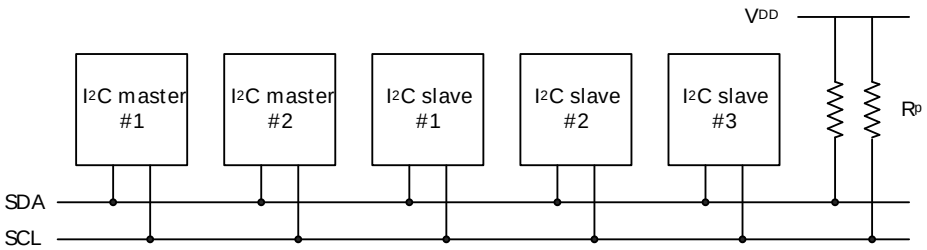
Figure 16.1. I2C Overview



16.3.1 I2C-Bus Overview

The I2C-bus uses two wires for communication; a serial data line (SDA) and a serial clock line (SCL) as shown in Figure 16.2 (p. 335) . As a true multi-master bus it includes collision detection and arbitration to resolve situations where multiple masters transmit data at the same time without data loss.

Figure 16.2. I2C-Bus Example



Each device on the bus is addressable by a unique address, and an I2C master can address all the devices on the bus, including other masters.

Both the bus lines are open-drain. The maximum value of the pull-up resistor can be calculated as a function of the maximal rise-time tr for the given bus speed, and the estimated bus capacitance Cb as shown in Equation 16.1 (p. 335) .

I2C Pull-up Resistor Equation

$$Rp(max) = (tr/0.8473) \times Cb. \quad (16.1)$$

The maximal rise times for 100 kHz, 400 kHz and 1 MHz I2C are 1 µs, 300 ns and 120 ns respectively.

Note

The GPIO drive strength can be used to control slew rate.

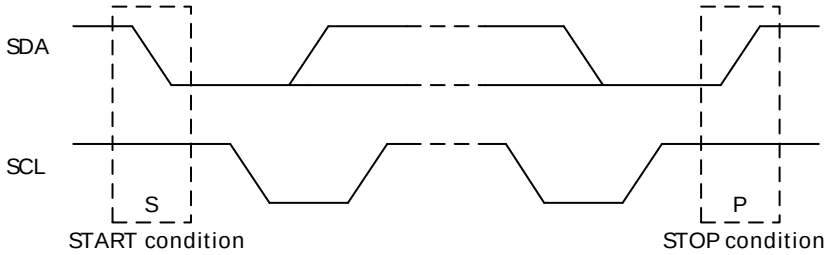
Note

If V_{dd} drops below the voltage on SCL and SDA lines, the MCU could become back powered and pull the SCL and SDA lines low.

16.3.1.1 START and STOP Conditions

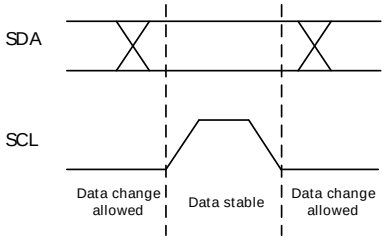
START and STOP conditions are used to initiate and stop transactions on the I^2C-bus. All transactions on the bus begin with a START condition (S) and end with a STOP condition (P). As shown in Figure 16.3 (p. 336) , a START condition is generated by pulling the SDA line low while SCL is high, and a STOP condition is generated by pulling the SDA line high while SCL is high.

Figure 16.3. I^2C START and STOP Conditions



The START and STOP conditions are easily identifiable bus events as they are the only conditions on the bus where a transition is allowed on SDA while SCL is high. During the actual data transmission, SDA is only allowed to change while SCL is low, and must be stable while SCL is high. One bit is transferred per clock pulse on the I^2C-bus as shown in Figure 16.2 (p. 335) .

Figure 16.4. I^2C Bit Transfer on I^2C-Bus



16.3.1.2 Bus Transfer

When a master wants to initiate a transfer on the bus, it waits until the bus is idle and transmits a START condition on the bus. The master then transmits the address of the slave it wishes to interact with and a single R/W bit telling whether it wishes to read from the slave (R/W bit set to 1) or write to the slave (R/W bit set to 0).

After the 7-bit address and the R/W bit, the master releases the bus, allowing the slave to acknowledge the request. During the next bit-period, the slave pulls SDA low (ACK) if it acknowledges the request, or keeps it high if it does not acknowledge it (NACK).

Following the address acknowledge, either the slave or master transmits data, depending on the value of the R/W bit. After every 8 bits (one byte) transmitted on the SDA line, the transmitter releases the line to allow the receiver to transmit an ACK or a NACK. Both the data and the address are transmitted with the most significant bit first.

The number of bytes in a bus transfer is unrestricted. The master ends the transmission after a (N)ACK by sending a STOP condition on the bus. After a STOP condition, any master wishing to initiate a transfer

on the bus can try to gain control of it. If the current master wishes to make another transfer immediately after the current, it can start a new transfer directly by transmitting a repeated START condition (Sr) instead of a STOP followed by a START.

Examples of I^{2}C transfers are shown in Figure 16.5 (p. 337), Figure 16.6 (p. 337), and Figure 16.7 (p. 337) . The identifiers used are:

- ADDR - Address
- DATA - Data
- S - Start bit
- Sr - Repeated start bit
- P - Stop bit
- W/R - Read(1)/Write(0)
- A - ACK
- N - NACK

Figure 16.5. I^{2}C Single Byte Write to Slave

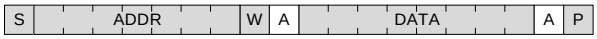
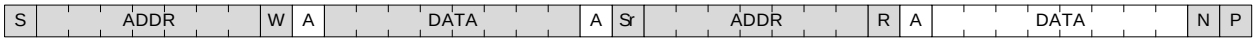


Figure 16.6. I^{2}C Double Byte Read from Slave

Figure 16.7. I^{2}C Single Byte Write, then Repeated Start and Single Byte Read



16.3.1.3 Addresses

I^{2}C supports both 7-bit and 10-bit addresses. When using 7-bit addresses, the first byte transmitted after the START-condition contains the address of the slave that the master wants to contact. In the 7-bit address space, several addresses are reserved. These addresses are summarized in Table 16.1 (p. 337) , and include a General Call address which can be used to broadcast a message to all slaves on the I^{2}C-bus.

Table 16.1. I^{2}C Reserved I^{2}C Addresses

I^{2}C Address	R/W	Description
0000-000	0	General Call address
0000-000	1	START byte
0000-001	X	Reserved for the C-Bus format
0000-010	X	Reserved for a different bus format
0000-011	X	Reserved for future purposes
0000-1XX	X	Reserved for future purposes
1111-1XX	X	Reserved for future purposes
1111-0XX	X	10 Bit slave addressing mode

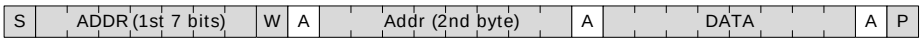
16.3.1.4 10-bit Addressing

To address a slave using a 10-bit address, two bytes are required to specify the address instead of one. The seven first bits of the first byte must then be 1111 0XX, where XX are the two most significant bits of the 10-bit address. As with 7-bit addresses, the eight bit of the first byte determines whether the master wishes to read from or write to the slave. The second byte contains the eight least significant bits of the slave address.

When a slave receives a 10-bit address, it must acknowledge both the address bytes if they match the address of the slave.

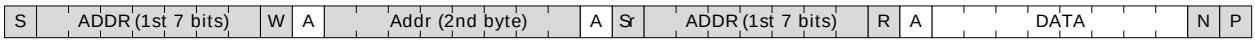
When performing a master transmitter operation, the master transmits the two address bytes and then the remaining data, as shown in Figure 16.8 (p. 338) .

Figure 16.8. I²C Master Transmitter/Slave Receiver with 10-bit Address



When performing a master receiver operation however, the master first transmits the two address bytes in a master transmitter operation, then sends a repeated START followed by the first address byte and then receives data from the addressed slave. The slave addressed by the 10-bit address in the first two address bytes must remember that it was addressed, and respond with data if the address transmitted after the repeated start matches its own address. An example of this (with one byte transmitted) is shown in Figure 16.9 (p. 338) .

Figure 16.9. I²C Master Receiver/Slave Transmitter with 10-bit Address



16.3.1.5 Arbitration, Clock Synchronization, Clock Stretching

Arbitration and clock synchronization are features aimed at allowing multi-master buses. Arbitration occurs when two devices try to drive the bus at the same time. If one device drives it low, while the other drives it high, the one attempting to drive it high will not be able to do so due to the open-drain bus configuration. Both devices sample the bus, and the one that was unable to drive the bus in the desired direction detects the collision and backs off, letting the other device continue communication on the bus undisturbed.

Clock synchronization is a means of synchronizing the clock outputs from several masters driving the bus at once, and is a requirement for effective arbitration.

Slaves on the bus are allowed to force the clock output on the bus low in order to pause the communication on the bus and give themselves time to process data or perform any real-time tasks they might have. This is called clock stretching.

Arbitration is supported by the I²C module for both masters and slaves. Clock synchronization and clock stretching is also supported.

16.3.2 Enable and Reset

The I²C is enabled by setting the EN bit in the I2Cn_CTRL register. Whenever this bit is cleared, the internal state of the I²C is reset, terminating any ongoing transfers.

Note

When enabling the I^2C, the ABORT command or the Bus Idle Timeout feature must be applied prior to use even if the BUSY flag is not set.

16.3.3 Safely Disabling and Changing Slave Configuration

The I^2C slave is partially asynchronous, and some precautions are necessary to always ensure a safe slave disable or slave configuration change. These measures should be taken, if (while the slave is enabled) the user cannot guarantee that an address match will not occur at the exact time of slave disable or slave configuration change.

Worst case consequences for an address match while disabling slave or changing configuration is that the slave may end up in an undefined state. To reset the slave back to a known state, the EN bit in I2Cn_CTRL must be reset. This should be done regardless of whether the slave is going to be re-enabled or not.

16.3.4 Clock Generation

The SCL signal generated by the I^2C master determines the maximum transmission rate on the bus. The clock is generated as a division of the peripheral clock, and is given by Equation 16.2 (p. 339) :

I^2C Maximum Transmission Rate

$$f_{SCL} = 1/(T_{low} + T_{high}), \quad (16.2)$$

where

T_{low} and T_{high} is the low and high periods of the clock signal respectively, given below. When the clock is not streched, the low and high periods of the clock signal are:

I^2C High and Low Cycles Equations

$$T_{high} = (N_{high} \times (CLKDIV + 1))/f_{HFPERCLK},$$
$$T_{low} = (N_{low} \times (CLKDIV + 1))/f_{HFPERCLK}. \quad (16.3)$$

Equation 16.3 (p. 339) and Equation 16.2 (p. 339) does not apply for low clock division factors (0, 1 and 2) because of synchronization. For these clock division factors, the formulas for computing high and low periods of the clock signal are given in Table 16.2 (p. 339) .

Table 16.2. I^2C High and Low Periods for Low CLKDIV

CLKDIV	Standard (4:4)		Asymmetric (6:3)		Fast (11:6)	
	T_{low}	T_{high}	T_{low}	T_{high}	T_{low}	T_{high}
0	$7/f_{HFPERCLK}$	$7/f_{HFPERCLK}$	$9/f_{HFPERCLK}$	$6/f_{HFPERCLK}$	$14/f_{HFPERCLK}$	$9/f_{HFPERCLK}$
1	$10/f_{HFPERCLK}$	$10/f_{HFPERCLK}$	$14/f_{HFPERCLK}$	$8/f_{HFPERCLK}$	$24/f_{HFPERCLK}$	$14/f_{HFPERCLK}$
2	$15/f_{HFPERCLK}$	$15/f_{HFPERCLK}$	$21/f_{HFPERCLK}$	$12/f_{HFPERCLK}$	$36/f_{HFPERCLK}$	$21/f_{HFPERCLK}$

The values of N_{low} and N_{high} and thus the ratio between the high and low parts of the clock signal is controlled by CLHR in the I2Cn_CTRL register. The available modes are summarized in Table 16.3 (p. 340) along with the highest I^2C-bus frequencies in the given modes that can be achieved without violating the timing specifications of the I^2C-bus. The maximum data hold time is dependent on the DIV and is given by:

Maximum Data Hold Time

$$t_{HD,DAT\text{-}max} = (4+DIV)/f_{HFPERCLK}. \quad (16.4)$$

Note

DIV must be set to 1 or higher during slave mode operation.

Table 16.3. I²C Clock Mode

HFPERCLK frequency (MHz)	Clock Low High Ratio (CLHR)	Sm max frequency (kHz)	Fm max frequency (kHz)	Fm+ max frequency (kHz)
28	0	92	400	1000
	1	81	400	848
	2	71	400	736
21	0	90	400	1000
	1	80	400	954
	2	72	368	552
14	0	92	400	1000
	1	81	400	636
	2	68	368	608
11	0	91	400	785
	1	81	333	733
	2	71	289	478
6.6	0	91	400	471
	1	81	299	439
	2	64	286	286
1.2	0	59	85	85
	1	54	79	79
	2	52	52	52

16.3.5 Arbitration

Arbitration is enabled by default, but can be disabled by setting the ARBDIS bit in I2Cn_CTRL. When arbitration is enabled, the value on SDA is sensed every time the I²C module attempts to change its value. If the sensed value is different than the value the I²C module tried to output, it is interpreted as a simultaneous transmission by another device, and that the I²C module has lost arbitration.

Whenever arbitration is lost, the ARBLOST interrupt flag in I2Cn_IF is set, any lines held are released, and the I²C device goes idle. If an I²C master loses arbitration during the transmission of an address, another master may be trying to address it. The master therefore receives the rest of the address, and if the address matches the slave address of the master, the master goes into either slave transmitter or slave receiver mode.

Note

Arbitration can be lost both when operating as a master and when operating as a slave.

16.3.6 Buffers

16.3.6.1 Transmit Buffer and Shift Register

The I²C transmitter is double buffered through the transmit buffer and transmit shift register as shown in Figure 16.1 (p. 335) . A byte is loaded into the transmit buffer by writing to I2Cn_TXDATA. When the

transmit shift register is empty and ready for new data, the byte from the transmit buffer is then loaded into the shift register. The byte is then kept in the shift register until it is transmitted. When a byte has been transmitted, a new byte is loaded into the shift register (if available in the transmit buffer). If the transmit buffer is empty, then the shift register also remains empty. The TXC flag in I2Cn_STATUS and the TXC interrupt flags in I2Cn_IF are then set, signaling that the transmit shift register is out of data. TXC is cleared when new data becomes available, but the TXC interrupt flag must be cleared by software.

Whenever a byte is loaded from the transmit buffer to the transmit shift register, the TXBL flag in I2Cn_STATUS and the TXBL interrupt flag in I2Cn_IF are set. This indicates that there is room in the buffer for more data. TXBL is cleared automatically when data is written to the buffer.

If a write is attempted to the transmit buffer while it is not empty, the TXOF interrupt flag in I2Cn_IF is set, indicating the overflow. The data already in the buffer remains preserved, and no new data is written.

The transmit buffer and the transmit shift register can be cleared by setting command bit CLEARTX in I2Cn_CMD. This will prevent the I^2C module from transmitting the data in the buffer and the shift register, and will make them available for new data. Any byte currently being transmitted will not be aborted. Transmission of this byte will be completed.

16.3.6.2 Receive Buffer and Shift Register

Like the transmitter, the I^2C receiver is double buffered. The receiver uses the receive buffer and receive shift register as shown in Figure 16.1 (p. 335) . When a byte has been fully received by the receive shift register, it is loaded into the receive buffer if there is room for it. Otherwise, the byte waits in the shift register until space becomes available in the buffer.

When a byte becomes available in the receive buffer, the RXDATAV in I2Cn_STATUS and RXDATAV interrupt flag in I2Cn_IF are set. The data can now be fetched from the buffer using I2Cn_RXDATA. Reading from this register will pull a byte out of the buffer, making room for a new byte and clearing RXDATAV in I2Cn_STATUS and RXDATAV in I2Cn_IF in the process.

If a read from the receive buffer is attempted through I2Cn_RXDATA while the buffer is empty, the RXUF interrupt flag in I2Cn_IF is set, and the data read from the buffer is undefined.

I2Cn_RXDATAP can be used to read data from the receive buffer without removing it from the buffer. The RXUF interrupt flag in I2Cn_IF will never be set as a result of reading from I2Cn_RXDATAP, but the data read through I2Cn_RXDATAP when the receive buffer is empty is still undefined.

Once a transaction is complete (STOP sent or received), the receive buffer needs to be flushed (all received data must be picked up) before starting a new transaction.

16.3.7 Master Operation

A bus transaction is initiated by transmitting a START condition (S) on the bus. This is done by setting the START bit in I2Cn_CMD. The command schedules a START condition, and makes the I^2C module generate a start condition whenever the bus becomes free.

The I^2C-bus is considered busy whenever another device on the bus transmits a START condition. Until a STOP condition is detected, the bus is owned by the master issuing the START condition. The bus is considered free when a STOP condition is transmitted on the bus. After a STOP is detected, all masters that have data to transmit send a START condition and begin transmitting data. Arbitration ensures that collisions are avoided.

When the START condition has been transmitted, the master must transmit a slave address (ADDR) with an R/W bit on the bus. If this address is available in the transmit buffer, the master transmits it immediately, but if the buffer is empty, the master holds the I^2C-bus while waiting for software to write the address to the transmit buffer.

After the address has been transmitted, a sequence of bytes can be read from or written to the slave, depending on the value of the R/W bit (bit 0 in the address byte). If the bit was cleared, the master has entered a master transmitter role, where it now transmits data to the slave. If the bit was set, it has entered a master receiver role, where it now should receive data from the slave. In either case, an unlimited number of bytes can be transferred in one direction during the transmission.

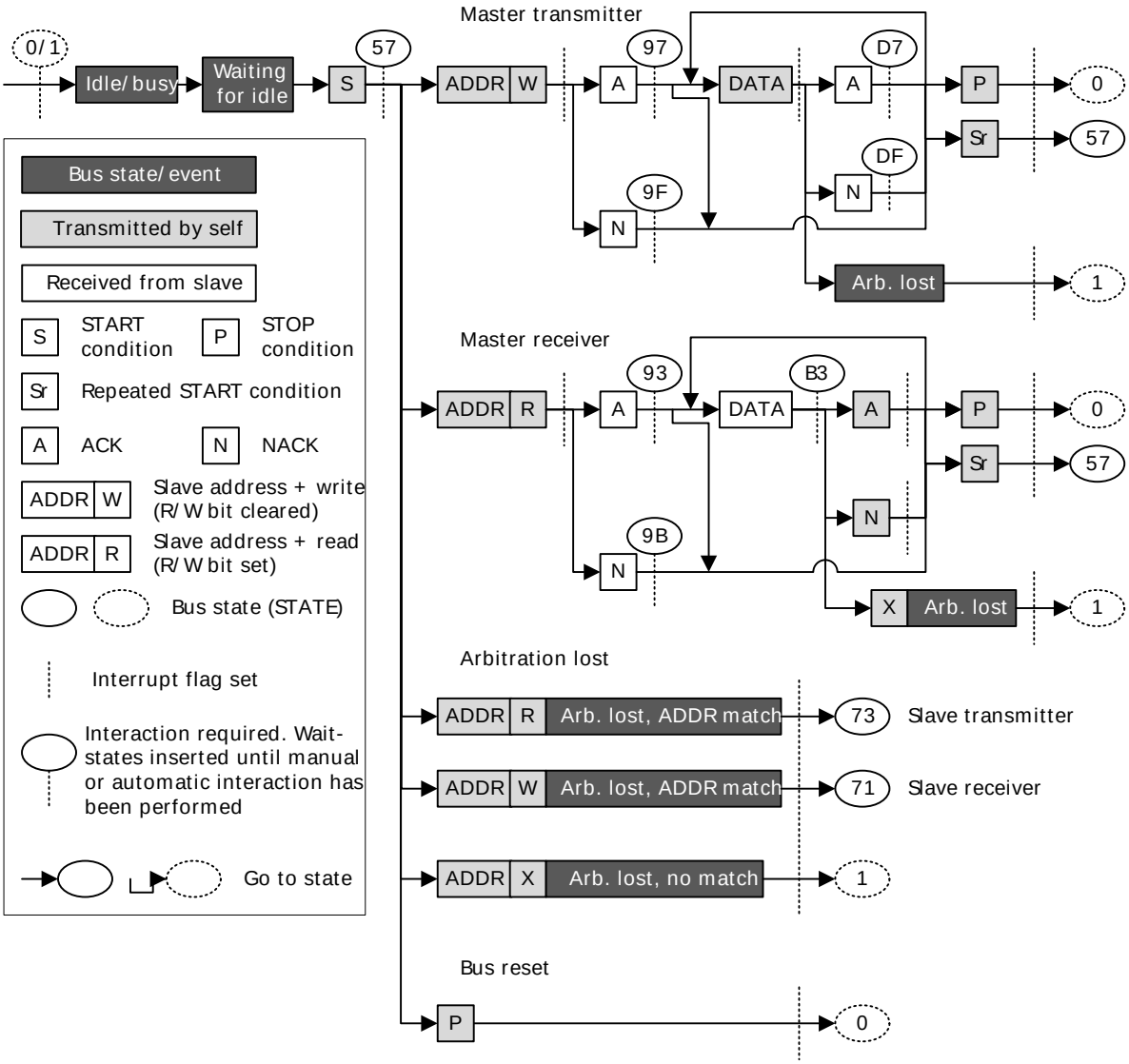
At the end of the transmission, the master either transmits a repeated START condition (Sr) if it wishes to continue with another transfer, or transmits a STOP condition (P) if it wishes to release the bus.

16.3.7.1 Master State Machine

The master state machine is shown in Figure 16.10 (p. 342) . A master operation starts in the far left of the state machine, and follows the solid lines through the state machine, ending the operation or continuing with a new operation when arriving at the right side of the state machine.

Branches in the path through the state machine are the results of bus events and choices made by software, either directly or indirectly. The dotted lines show where I^{2}C-specific interrupt flags are set along the path and the full-drawn circles show places where interaction may be required by software to let the transmission proceed.

Figure 16.10. I^{2}C Master State Machine



16.3.7.2 Interactions

Whenever the I²C module is waiting for interaction from software, it holds the bus clock SCL low, freezing all bus activities, and the BUSHOLD interrupt flag in I2Cn_IF is set. The action(s) required by software depends on the current state the of the I²C module. This state can be read from the I2Cn_STATE register.

As an example, Table 16.5 (p. 345) shows the different states the I²C goes through when operating as a Master Transmitter, i.e. a master that transmits data to a slave. As seen in the table, when a start condition has been transmitted, a requirement is that there is an address and an R/W bit in the transmit buffer. If the transmit buffer is empty, then the BUSHOLD interrupt flag is set, and the bus is held until data becomes available in the buffer. While waiting for the address, I2Cn_STATE has a value 0x57, which can be used to identify exactly what the I²C module is waiting for.

Note

The bus would never stop at state 0x57 if the address was available in the transmit buffer.

The different interactions used by the I²C module are listed in Table 16.4 (p. 343) in prioritized order. If a set of different courses of action are possible from a given state, the course of action using the highest priority interactions, that first has everything it is waiting for is the one that is taken.

Table 16.4. I²C Interactions in Prioritized Order

Interaction	Priority	Software action	Automatically continues if
STOP*	1	Set the STOP command bit in I2Cn_CMD	PSTOP is set (STOP pending) in I2Cn_STATUS
ABORT	2	Set the ABORT command bit in I2Cn_CMD	Never, the transmission is aborted
CONT*	3	Set the CONT command bit in I2Cn_CMD	PCONT is set in I2Cn_STATUS (CONT pending)
NACK*	4	Set the NACK command bit in I2Cn_CMD	PNACK is set in I2Cn_STATUS (NACK pending)
ACK*	5	Set the ACK command bit in I2Cn_CMD	AUTOACK is set in I2Cn_CTRL or PACK is set in I2Cn_STATUS (ACK pending)
ADDR+W -> TXDATA	6	Write an address to the transmit buffer with the R/W bit set	Address is available in transmit buffer with R/W bit set
ADDR+R -> TXDATA	7	Write an address to the transmit buffer with the R/W bit cleared	Address is available in transmit buffer with R/W bit cleared
START*	8	Set the START command bit in I2Cn_CMD	PSTART is set in I2Cn_STATUS (START pending)
TXDATA	9	Write data to the transmit buffer	Data is available in transmit buffer
RXDATA	10	Read data from receive buffer	Space is available in receive buffer
None	11	No interaction is required	

The commands marked with a * in Table 16.4 (p. 343) can be issued before an interaction is required. When such a command is issued before it can be used/consumed by the I²C module, the command is

set in a pending state, which can be read from the STATUS register. A pending START command can for instance be identified by PSTART having a high value.

Whenever the I²C module requires an interaction, it checks the pending commands. If one or a combination of these can fulfill an interaction, they are consumed by the module and the transmission continues without setting the BUSHOLD interrupt flag in I2Cn_IF to get an interaction from software. The pending status of a command goes low when it is consumed.

When several interactions are possible from a set of pending commands, the interaction with the highest priority, i.e. the interaction closest to the top of Table 16.4 (p. 343) is applied to the bus.

Pending commands can be cleared by setting the CLEARPC command bit in I2Cn_CMD.

16.3.7.2.1 Automatic ACK Interaction

When receiving addresses and data, an ACK command in I2Cn_CMD is normally required after each received byte. When AUTOACK is set in I2Cn_CTRL, an ACK is always pending, and the ACK-pending bit PACK in I2Cn_STATUS is thus always set, even after an ACK has been consumed. This can be used to reduce the amount of software interaction required during a transfer.

16.3.7.3 Reset State

After a reset, the state of the I²C-bus is unknown. To avoid interrupting transfers on the I²C-bus after a reset of the I²C module or the entire MCU, the I²C-bus is assumed to be busy when coming out of a reset, and the BUSY flag in I2Cn_STATUS is thus set. To be able to carry through master operations on the I²C-bus, the bus must be idle.

The bus goes idle when a STOP condition is detected on the bus, but on buses with little activity, the time before the I²C module detects that the bus is idle can be significant. There are two ways of assuring that the I²C module gets out of the busy state.

- Use the ABORT command in I2Cn_CMD. When the ABORT command is issued, the I²C module is instructed that the bus is idle. The I²C module can then initiate master operations.
- Use the Bus Idle Timeout. When SCL has been high for a long period of time, it is very likely that the bus is idle. Set BITO in I2Cn_CTRL to an appropriate timeout period and set GIBITO in I2Cn_CTRL. If activity has not been detected on the bus within the timeout period, the bus is then automatically assumed idle, and master operations can be initiated.

Note

If operating in slave mode, the above approach is not necessary.

16.3.7.4 Master Transmitter

To transmit data to a slave, the master must operate as a master transmitter. Table 16.5 (p. 345) shows the states the I²C module goes through while acting as a master transmitter. Every state where an interaction is required has the possible interactions listed, along with the result of the interactions. The table also shows which interrupt flags are set in the different states. The interrupt flags enclosed in parenthesis may be set. If the BUSHOLD interrupt in I2Cn_IF is set, the module is waiting for an interaction, and the bus is frozen. The value of I2Cn_STATE will be equal to the values given in the table when the BUSHOLD interrupt flag is set, and can be used to determine which interaction is required to make the transmission continue.

The interrupt flag START in I2Cn_IF is set when the I²C module transmits the START.

A master operation is started by issuing a START command by setting START in I2Cn_CMD. ADDR +W, i.e. the address of the slave to address + the R/W bit is then required by the I²C module. If this is not available in the transmit buffer, then the bus is held and the BUSHOLD interrupt flag is set. The

value of I2Cn_STATE will then be 0x57. As seen in the table, the I²C module also stops in this state if the address is not available after a repeated start condition.

To continue, write a byte to I2Cn_TXDATA with the address of the slave in the 7 most significant bits and the least significant bit cleared (ADDR+W). This address will then be transmitted, and the slave will reply with an ACK or a NACK. If no slave replies to the address, the response will also be NACK. If the address was acknowledged, the master now has four choices. It can send a data byte by placing it in I2Cn_TXDATA (the master should check the TXBL interrupt flag before writing to I2Cn_TXDATA), this byte is then transmitted. The master can also stop the transmission by sending a STOP, it can send a repeated start by sending START, or it can send a STOP and then a START as soon as possible.

If a NACK was received, the master has to issue a CONT command in addition to providing data in order to continue transmission. This is not standard I²C, but is provided for flexibility. The rest of the options are similar to when an ACK was received.

If a new byte was transmitted, an ACK or NACK is received after the transmission of the byte, and the master has the same options as for when the address was sent.

The master may lose arbitration at any time during transmission. In this case, the ARBLOST interrupt flag in I2Cn_IF is set. If the arbitration was lost during the transfer of an address, and SLAVE in I2Cn_CTRL is set, the master then checks which address was transmitted. If it was the address of the master, then the master goes to slave mode.

After a master has transmitted a START and won any arbitration, it owns the bus until it transmits a STOP. After a STOP, the bus is released, and arbitration decides which bus master gains the bus next. The MSTOP interrupt flag in I2Cn_IF is set when a STOP condition is transmitted by the master.

Table 16.5. I²C Master Transmitter

I2Cn_STAT	Description	I2Cn_IF	Required interaction	Response
0x57	Start transmitted	START interrupt flag (BUSHOLD interrupt flag)	ADDR +W -> TXDATA	ADDR+W will be sent
			STOP	STOP will be sent and bus released.
			STOP + START	STOP will be sent and bus released. Then a START will be sent when bus becomes idle.
0x57	Repeated start transmitted	START interrupt flag (BUSHOLD interrupt flag)	ADDR +W -> TXDATA	ADDR+W will be sent
			STOP	STOP will be sent and bus released.
			STOP + START	STOP will be sent and bus released. Then a START will be sent when bus becomes idle.
-	ADDR+W transmitted	TXBL interrupt flag (TXC interrupt flag)	None	
0x97	ADDR+W transmitted, ACK received	ACK interrupt flag (BUSHOLD interrupt flag)	TXDATA	DATA will be sent
			STOP	STOP will be sent. Bus will be released
			START	Repeated start condition will be sent
			STOP + START	STOP will be sent and the bus released. Then a START will be sent when the bus becomes idle
0x9F	ADDR+W transmitted,NACK received	NACK (BUSHOLD interrupt flag)	CONT + TXDATA	DATA will be sent
			STOP	STOP will be sent. Bus will be released

I2Cn_STAT	Description	I2Cn_IF	Required interaction	Response
			START	Repeated start condition will be sent
			STOP + START	STOP will be sent and the bus released. Then a START will be sent when the bus becomes idle
-	Data transmitted	TXBL interrupt flag (TXC interrupt flag)	None	
0xD7	Data transmitted,ACK received	ACK interrupt flag (BUSHOLD interrupt flag)	TXDATA	DATA will be sent
			STOP	STOP will be sent. Bus will be released
			START	Repeated start condition will be sent
			STOP + START	STOP will be sent and the bus released. Then a START will be sent when the bus becomes idle
0xDF	Data transmitted,NACK received	NACK(BUSHOLD interrupt flag)	CONT + TXDATA	DATA will be sent
			STOP	STOP will be sent. Bus will be released
			START	Repeated start condition will be sent
			STOP + START	STOP will be sent and the bus released. Then a START will be sent when the bus becomes idle
-	Stop transmitted	MSTOP interrupt flag	None	
			START	START will be sent when bus becomes idle
-	Arbitration lost	ARBLOST interrupt flag	None	
			START	START will be sent when bus becomes idle

16.3.7.5 Master Receiver

To receive data from a slave, the master must operate as a master receiver, see Table 16.6 (p. 347) . This is done by transmitting ADDR+R as the address byte instead of ADDR+W, which is transmitted to become a master transmitter. The address byte loaded into the data register thus has to contain the 7-bit slave address in the 7 most significant bits of the byte, and have the least significant bit set.

When the address has been transmitted, the master receives an ACK or a NACK. If an ACK is received, the ACK interrupt flag in I2Cn_IF is set, and if space is available in the receive shift register, reception of a byte from the slave begins. If the receive buffer and shift register is full however, the bus is held until data is read from the receive buffer or another interaction is made. Note that the STOP and START interactions have a higher priority than the data-available interaction, so if a STOP or START command is pending, the highest priority interaction will be performed, and data will not be received from the slave.

If a NACK was received, the CONT command in I2Cn_CMD has to be issued in order to continue receiving data, even if there is space available in the receive buffer and/or shift register.

After a data byte has been received the master must ACK or NACK the received byte. If an ACK is pending or AUTOACK in I2Cn_CTRL is set, an ACK is sent automatically and reception continues if space is available in the receive buffer.

If a NACK is sent, the CONT command must be used in order to continue transmission. If an ACK or NACK is issued along with a START or STOP or both, then the ACK/NACK is transmitted and the reception is ended. If START in I2Cn_CMD is set alone, a repeated start condition is transmitted after the ACK/NACK. If STOP in I2Cn_CMD is set, a stop condition is sent regardless of whether START is set. If START is set in this case, it is set as pending.

As when operating as a master transmitter, arbitration can be lost as a master receiver. When this happens the ARBLOST interrupt flag in I2Cn_IF is set, and the master has a possibility of being selected as a slave given the correct conditions.

Table 16.6. I^2C Master Receiver

I2Cn_STAT	Description	I2Cn_IF	Required interaction	Response
0x57	START transmitted	START interrupt flag (BUSHOLD interrupt flag)	ADDR +R -> TXDATA	ADDR+R will be sent
			STOP	STOP will be sent and bus released.
			STOP + START	STOP will be sent and bus released. Then a START will be sent when bus becomes idle.
0x57	Repeated START transmitted	START interrupt flag(BUSHOLD interrupt flag)	ADDR +R -> TXDATA	ADDR+R will be sent
			STOP	STOP will be sent and bus released.
			STOP + START	STOP will be sent and bus released. Then a START will be sent when bus becomes idle.
-	ADDR+R transmitted	TXBL interrupt flag (TXC interrupt flag)	None	
0x93	ADDR+R transmitted, ACK received	ACK interrupt flag(BUSHOLD)	RXDATA	Start receiving
			STOP	STOP will be sent and the bus released
			START	Repeated START will be sent
			STOP + START	STOP will be sent and the bus released. Then a START will be sent when the bus becomes idle
0x9B	ADDR+R transmitted,NACK received	NACK(BUSHOLD)	CONT + RXDATA	Continue, start receiving
			STOP	STOP will be sent and the bus released
			START	Repeated START will be sent
			STOP + START	STOP will be sent and the bus released. Then a START will be sent when the bus becomes idle
0xB3	Data received	RXDATA interrupt flag(BUSHOLD interrupt flag)	ACK + RXDATA	ACK will be transmitted, reception continues
			NACK + CONT + RXDATA	NACK will be transmitted, reception continues
			ACK/ NACK + STOP	ACK/NACK will be sent and the bus will be released.
			ACK/ NACK + START	ACK/NACK will be sent, and then a repeated start condition.
			ACK/ NACK + STOP + START	ACK/NACK will be sent and the bus will be released. Then a START will be sent when the bus becomes idle
-	Stop received	MSTOP interrupt flag	None	

I2Cn_STAT	Description	I2Cn_IF	Required interaction	Response
			START	START will be sent when bus becomes idle
-	Arbitration lost	ARBLOST interrupt flag	None	
			START	START will be sent when bus becomes idle

16.3.8 Bus States

The I2Cn_STATE register can be used to determine which state the I^{2}C module and the I^{2}C bus are in at a given time. The register consists of the STATE bit-field, which shows which state the I^{2}C module is at in any ongoing transmission, and a set of single-bits, which reveal the transmission mode, whether the bus is busy or idle, and whether the bus is held by this I^{2}C module waiting for a software response.

The possible values of the STATE field are summarized in Table 16.7 (p. 348) . When this field is cleared, the I^{2}C module is not a part of any ongoing transmission. The remaining status bits in the I2Cn_STATE register are listed in Table 16.8 (p. 348) .

Table 16.7. I^{2}C STATE Values

Mode	Value	Description
IDLE	0	No transmission is being performed by this module.
WAIT	1	Waiting for idle. Will send a start condition as soon as the bus is idle.
START	2	Start being transmitted
ADDR	3	Address being transmitted or has been received
ADDRACK	4	Address ACK/NACK being transmitted or received
DATA	5	Data being transmitted or received
DATAACK	6	Data ACK/NACK being transmitted or received

Table 16.8. I^{2}C Transmission Status

Bit	Description
BUSY	Set whenever there is activity on the bus. Whether or not this module is responsible for the activity cannot be determined by this byte.
MASTER	Set when operating as a master. Cleared at all other times.
TRANSMITTER	Set when operating as a transmitter; either a master transmitter or a slave transmitter. Cleared at all other times
BUSHOLD	Set when the bus is held by this I^{2}C module because an action is required by software.
NACK	Only valid when bus is held and STATE is ADDRACK or DATAACK. In that case it is set if a NACK was received. In all other cases, the bit is cleared.

Note

I2Cn_STATE reflects the internal state of the I^{2}C module, and therefore only held constant as long as the bus is held, i.e. as long as BUSHOLD in I2Cn_STATUS is set.

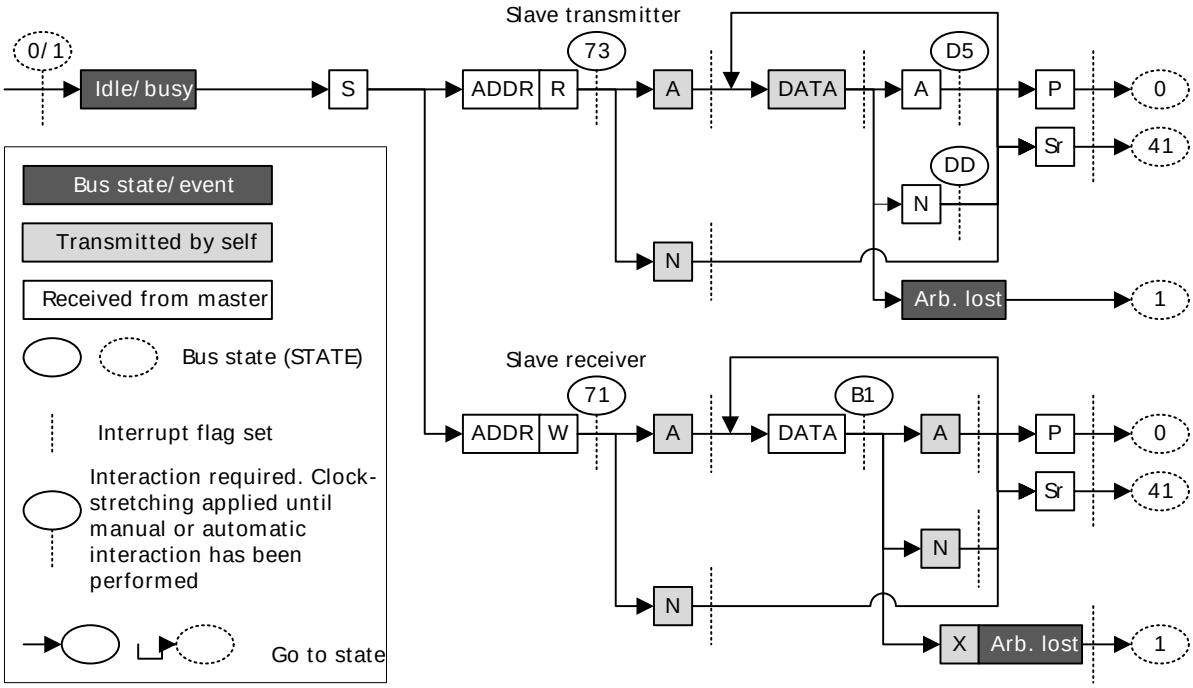
16.3.9 Slave Operation

The I^{2}C module operates in master mode by default. To enable slave operation, i.e. to allow the device to be addressed as an I^{2}C slave, the SLAVE bit in I2Cn_CTRL must be set. In this case the slave operates in a mixed mode, both capable of starting transmissions as a master, and being addressed as a slave. When operating in the slave mode, HFPERCLK frequency must be higher than 4.2 MHz for Standard-mode, 11 MHz for Fast-mode, and 24.4 MHz for Fast-mode Plus.

16.3.9.1 Slave State Machine

The slave state machine is shown in Figure 16.11 (p. 349) . The dotted lines show where I²C-specific interrupt flags are set. The full-drawn circles show places where interaction may be required by software to let the transmission proceed.

Figure 16.11. I²C Slave State Machine



16.3.9.2 Address Recognition

The I²C module provides automatic address recognition for 7-bit addresses. 10-bit address recognition is not fully automatic, but can be assisted by the 7-bit address comparator as shown in Section 16.3.11 (p. 353) . Address recognition is supported in all energy modes (except EM4).

The slave address, i.e. the address which the I²C module should be addressed with, is defined in the I2Cn_SADDR register. In addition to the address, a mask must be specified, telling the address comparator which bits of an incoming address to compare with the address defined in I2Cn_SADDR. The mask is defined in I2Cn_SADDRMASK, and for every zero in the mask, the corresponding bit in the slave address is treated as a don't-care.

An incoming address that fails address recognition is automatically replied to with a NACK. Since only the bits defined by the mask are checked, a mask with a value 0x00 will result in all addresses being accepted. A mask with a value 0x7F will only match the exact address defined in I2Cn_SADDR, while a mask 0x70 will match all addresses where the three most significant bits in I2Cn_SADDR and the incoming address are equal.

If GCAMEN in I2Cn_CTRL is set, the general call address is always accepted regardless of the result of the address recognition. The start-byte, i.e. the general call address with the R/W bit set is ignored unless it is included in the defined slave address.

When an address is accepted by the address comparator, the decision of whether to ACK or NACK the address is passed to software.

16.3.9.3 Slave Transmitter

When SLAVE in I2Cn_CTRL is set, the RSTART interrupt flag in I2Cn_IF will be set when repeated START conditions are detected. After a START or repeated START condition, the bus master will transmit an address along with an R/W bit. If there is no room in the receive shift register for the address, the bus will be held by the slave until room is available in the shift register. Transmission then continues and the address is loaded into the shift register. If this address does not pass address recognition, it is automatically NACK'ed by the slave, and the slave goes to an idle state. The address byte is in this case discarded, making the shift register ready for a new address. It is not loaded into the receive buffer.

If the address was accepted and the R/W bit was set (R), indicating that the master wishes to read from the slave, the slave now goes into the slave transmitter mode. Software interaction is now required to decide whether the slave wants to acknowledge the request or not. The accepted address byte is loaded into the receive buffer like a regular data byte. If no valid interaction is pending, the bus is held until the slave responds with a command. The slave can reject the request with a single NACK command.

The slave will in that case go to an idle state, and wait for the next start condition. To continue the transmission, the slave must make sure data is loaded into the transmit buffer and send an ACK. The loaded data will then be transmitted to the master, and an ACK or NACK will be received from the master.

Data transmission can also continue after a NACK if a CONT command is issued along with the NACK. This is not standard I^2C however.

If the master responds with an ACK, it may expect another byte of data, and data should be made available in the transmit buffer. If data is not available, the bus is held until data is available.

If the response is a NACK however, this is an indication of that the master has received enough bytes and wishes to end the transmission. The slave now automatically goes idle, unless CONT in I2Cn_CMD is set and data is available for transmission. The latter is not standard I^2C.

The master ends the transmission by sending a STOP or a repeated START. The SSTOP interrupt flag in I2Cn_IF is set when the master transmits a STOP condition. If the transmission is ended with a repeated START, then the SSTOP interrupt flag is not set.

Note

The SSTOP interrupt flag in I2Cn_IF will be set regardless of whether the slave is participating in the transmission or not, as long as SLAVE in I2Cn_CTRL is set and a STOP condition is detected

If arbitration is lost at any time during transmission, the ARBLOST interrupt flag in I2Cn_IF is set, the bus is released and the slave goes idle.

See Table 16.9 (p. 351) for more information.

Table 16.9. I²C Slave Transmitter

I2Cn_STAT	Description	I2Cn_IF	Required interaction	Response
0x41	Repeated START received	RSTART interrupt flag (BUSHOLD interrupt flag)	RXDATA	Receive and compare address
0x75	ADDR + R received	ADDR interrupt flag	ACK + TXDATA	ACK will be sent, then DATA
		RXDATA interrupt flag	NACK	NACK will be sent, slave goes idle
		(BUSHOLD interrupt flag)	NACK + CONT + TXDATA	NACK will be sent, then DATA.
-	Data transmitted	TXBL interrupt flag (TXC interrupt flag)	None	
0xD5	Data transmitted, ACK received	ACK interrupt flag (BUSHOLD interrupt flag)	TXDATA	DATA will be transmitted
0xDD	Data transmitted, NACK received	NACK interrupt flag	None	The slave goes idle
		(BUSHOLD interrupt flag)	CONT + TXDATA	DATA will be transmitted
-	Stop received	SSTOP interrupt flag	None	The slave goes idle
			START	START will be sent when bus becomes idle
-	Arbitration lost	ARBLOST interrupt flag	None	The slave goes idle
			START	START will be sent when the bus becomes idle

16.3.9.4 Slave Receiver

A slave receiver operation is started in the same way as a slave transmitter operation, with the exception that the address transmitted by the master has the R/W bit cleared (W), indicating that the master wishes to write to the slave. The slave then goes into slave receiver mode.

To receive data from the master, the slave should respond to the address with an ACK and make sure space is available in the receive buffer. Transmission will then continue, and the slave will receive a byte from the master.

If a NACK is sent without a CONT, the transmission is ended for the slave, and it goes idle. If the slave issues both the NACK and CONT commands and has space available in the receive buffer, it will be open for continuing reception from the master.

When a byte has been received from the master, the slave must ACK or NACK the byte. The responses here are the same as for the reception of the address byte.

The master ends the transmission by sending a STOP or a repeated START. The SSTOP interrupt flag is set when the master transmits a STOP condition. If the transmission is ended with a repeated START, then the SSTOP interrupt flag in I2Cn_IF is not set.

Note

The SSTOP interrupt flag in I2Cn_IF will be set regardless of whether the slave is participating in the transmission or not, as long as SLAVE in I2Cn_CTRL is set and a STOP condition is detected

If arbitration is lost at any time during transmission, the ARBLOST interrupt flag in I2Cn_IF is set, the bus is released and the slave goes idle.

See Table 16.10 (p. 352) for more information.

Table 16.10. I^2C - Slave Receiver

I2Cn_STAT	Description	I2Cn_IF	Required interaction	Response
-	Repeated START received	RSTART interrupt flag (BUSHOLD interrupt flag)	RXDATA	Receive and compare address
0x71	ADDR + W received	ADDR interrupt flag RXDATA interrupt flag (BUSHOLD interrupt flag)	ACK + RXDATA	ACK will be sent and data will be received
			NACK	NACK will be sent, slave goes idle
			NACK + CONT + RXDATA	NACK will be sent and DATA will be received.
0xB1	Data received	RXDATA interrupt flag (BUSHOLD interrupt flag)	ACK + RXDATA	ACK will be sent and data will be received
			NACK	NACK will be sent and slave will go idle
			NACK + CONT + RXDATA	NACK will be sent and data will be received
-	Stop received	SSTOP interrupt flag	None	The slave goes idle
			START	START will be sent when bus becomes idle
-	Arbitration lost	ARBLOST interrupt flag	None	The slave goes idle
			START	START will be sent when the bus becomes idle

16.3.10 Transfer Automation

The I^2C can be set up to complete transfers with a minimal amount of interaction.

16.3.10.1 DMA

DMA can be used to automatically load data into the transmit buffer and load data out from the receive buffer. When using DMA, software is thus relieved of moving data to and from memory after each transferred byte.

16.3.10.2 Automatic ACK

When AUTOACK in I2Cn_CTRL is set, an ACK is sent automatically whenever an ACK interaction is possible and no higher priority interactions are pending.

16.3.10.3 Automatic STOP

A STOP can be generated automatically on two conditions. These apply only to the master transmitter.

If AUTOSN in I2Cn_CTRL is set, the I^2C module ends a transmission by transmitting a STOP condition when operating as a master transmitter and a NACK is received.

If AUTOSE in I2Cn_CTRL is set, the I^2C module always ends a transmission when there is no more data in the transmit buffer. If data has been transmitted on the bus, the transmission is ended after the (N)ACK has been received by the slave. If a START is sent when no data is available in the transmit buffer and AUTOSE is set, then the STOP condition is sent immediately following the START. Software must thus make sure data is available in the transmit buffer before the START condition has been fully transmitted if data is to be transferred.

16.3.11 Using 10-bit Addresses

When using 10-bit addresses in slave mode, set the I2Cn_SADDR register to 1111 0XX where XX are the two most significant bits of the 10-bit address, and set I2Cn_SADDRMASK to 0xFF. Address matches will now be given on all 10-bit addresses where the two most significant bits are correct.

When receiving an address match, the slave must acknowledge the address and receive the first data byte. This byte contains the second part of the 10-bit address. If it matches the address of the slave, the slave should ACK the byte to continue the transmission, and if it does not match, the slave should NACK it.

When the master is operating as a master transmitter, the data bytes will follow after the second address byte. When the master is operating as a master receiver however, a repeated START condition is sent after the second address byte. The address sent after this repeated START is equal to the first of the address bytes transmitted previously, but now with the R/W byte set, and only the slave that found a match on the entire 10-bit address in the previous message should ACK this address. The repeated start should take the master into a master receiver mode, and after the single address byte sent this time around, the slave begins transmission to the master.

16.3.12 Error Handling

16.3.12.1 ABORT Command

Some bus errors may require software intervention to be resolved. The I^2C module provides an ABORT command, which can be set in I2Cn_CMD, to help resolve bus errors.

When the bus for some reason is locked up and the I^2C module is in the middle of a transmission it cannot get out of, or for some other reason the I^2C wants to abort a transmission, the ABORT command can be used.

Setting the ABORT command will make the I^2C module discard any data currently being transmitted or received, release the SDA and SCL lines and go to an idle mode. ABORT effectively makes the I^2C module forget about any ongoing transfers.

16.3.12.2 Bus Reset

A bus reset can be performed by setting the START and STOP commands in I2Cn_CMD while the transmit buffer is empty. A START condition will then be transmitted, immediately followed by a STOP condition. A bus reset can also be performed by transmitting a START command with the transmit buffer empty and AUTOSE set.

16.3.12.3 I^2C-Bus Errors

An I^2C-bus error occurs when a START or STOP condition is misplaced, which happens when the value on SDA changes while SCL is high during bit-transmission on the I^2C-bus. If the I^2C module is part of the current transmission when a bus error occurs, any data currently being transmitted or received is discarded, SDA and SCL are released, the BUSERR interrupt flag in I2Cn_IF is set to indicate the error, and the module automatically takes a course of action as defined in Table 16.11 (p. 353) .

Table 16.11. I^2C Bus Error Response

	Misplaced START	Misplaced STOP
In a master/slave operation	Treated as START. Receive address.	Go idle. Perform any pending actions.

16.3.12.4 Bus Lockup

A lockup occurs when a master or slave on the I^2C-bus has locked the SDA or SCL at a low value, preventing other devices from putting high values on the bus, and thus making communication on the bus impossible.

Many slave-only devices operating on an I2C-bus are not capable of driving SCL low, but in the rare case that SCL is stuck LOW, the advice is to apply a hardware reset signal to the slaves on the bus. If this does not work, cycle the power to the devices in order to make them release SCL.

When SDA is stuck low and SCL is free, a master should send 9 clock pulses on SCL while tristating the SDA. This procedure is performed in the GPIO module after clearing the I2C_ROUTE register and disabling the I2C module. The device that held the bus low should release it sometime within those 9 clocks. If not, use the same approach as for when SCL is stuck, resetting and possibly cycling power to the slaves.

Lockup of SDA can be detected by keeping count of the number of continuous arbitration losses during address transmission. If arbitration is also lost during the transmission of a general call address, i.e. during the transmission of the STOP condition, which should never happen during normal operation, this is a good indication of SDA lockup.

Detection of SCL lockups can be done using the timeout functionality defined in Section 16.3.12.6 (p. 354)

16.3.12.5 Bus Idle Timeout

When SCL has been high for a significant amount of time, this is a good indication of that the bus is idle. On an SMBus system, the bus is only allowed to be in this state for a maximum of 50 µs before the bus is considered idle.

The bus idle timeout BITO in I2Cn_CTRL can be used to detect situations where the bus goes idle in the middle of a transmission. The timeout can be configured in BITO, and when the bus has been idle for the given amount of time, the BITO interrupt flag in I2Cn_IF is set. The bus can also be set idle automatically on a bus idle timeout. This is enabled by setting GIBITO in I2Cn_CTRL.

When the bus idle timer times out, it wraps around and continues counting as long as its condition is true. If the bus is not set idle using GIBITO or the ABORT command in I2Cn_CMD, this will result in periodic timeouts.

Note

This timeout will be generated even if SDA is held low.

The bus idle timeout is active as long as the bus is busy, i.e. BUSY in I2Cn_STATUS is set. The timeout can be used to get the I2C module out of the busy-state it enters when reset, see Section 16.3.7.3 (p. 344) .

16.3.12.6 Clock Low Timeout

The clock timeout, which can be configured in CLTO in I2Cn_CTRL, starts counting whenever SCL goes low, and times out if SCL does not go high within the configured timeout. A clock low timeout results in CLTOIF in I2Cn_IF being set, allowing software to take action.

When the timer times out, it wraps around and continues counting as long as SCL is low. An SCL lockup will thus result in periodic clock low timeouts as long as SCL is low.

16.3.13 DMA Support

The I2C module has full DMA support. The DMA controller can write to the transmit buffer using the I2Cn_TXDATA register, and it can read from the receive buffer using the RXDATA register. A request for the DMA controller to read from the I2C receive buffer can come from the following source:

- Data available in the receive buffer

A write request can come from one of the following sources:

- Transmit buffer and shift register empty. No data to send
- Transmit buffer empty

16.3.14 Interrupts

The interrupts generated by the I^2C module are combined into one interrupt vector, I2C_INT. If I^2C interrupts are enabled, an interrupt will be made if one or more of the interrupt flags in I2Cn_IF and their corresponding bits in I2Cn_IEN are set.

16.3.15 Wake-up

The I^2C receive section can be active all the way down to energy mode EM3, and can wake up the CPU on address interrupt. All address match modes are supported.

16.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	I2Cn_CTRL	RW	Control Register
0x004	I2Cn_CMD	W1	Command Register
0x008	I2Cn_STATE	R	State Register
0x00C	I2Cn_STATUS	R	Status Register
0x010	I2Cn_CLKDIV	RW	Clock Division Register
0x014	I2Cn_SADDR	RW	Slave Address Register
0x018	I2Cn_SADDRMASK	RW	Slave Address Mask Register
0x01C	I2Cn_RXDATA	R	Receive Buffer Data Register
0x020	I2Cn_RXDATAP	R	Receive Buffer Data Peek Register
0x024	I2Cn_TXDATA	W	Transmit Buffer Data Register
0x028	I2Cn_IF	R	Interrupt Flag Register
0x02C	I2Cn_IFS	W1	Interrupt Flag Set Register
0x030	I2Cn_IFC	W1	Interrupt Flag Clear Register
0x034	I2Cn_IEN	RW	Interrupt Enable Register
0x038	I2Cn_ROUTE	RW	I/O Routing Register

16.5 Register Description

16.5.1 I2Cn_CTRL - Control Register

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset														0x0			0		0x0				0x0			0	0	0	0	0	0	0
Access														RW			RW		RW				RW			RW	RW	RW	RW	RW	RW	RW
Name														CLTO			GIBITO		BITO				CLHR			GCAMEN	ARBDIS	AUTOSN	AUTOSE	AUTOACK	SLAVE	EN

Bit	Name	Reset	Access	Description
31:19	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
18:16	CLTO	0x0	RW	**Clock Low Timeout**

Use to generate a timeout when CLK has been low for the given amount of time. Wraps around and continues counting when the timeout is reached.

Value	Mode	Description
0	OFF	Timeout disabled
1	40PCC	Timeout after 40 prescaled clock cycles. In standard mode at 100 kHz, this results in a 50us timeout.
2	80PCC	Timeout after 80 prescaled clock cycles. In standard mode at 100 kHz, this results in a 100us timeout.
3	160PCC	Timeout after 160 prescaled clock cycles. In standard mode at 100 kHz, this results in a 200us timeout.
4	320PPC	Timeout after 320 prescaled clock cycles. In standard mode at 100 kHz, this results in a 400us timeout.
5	1024PPC	Timeout after 1024 prescaled clock cycles. In standard mode at 100 kHz, this results in a 1280us timeout.

15	GIBITO	0	RW	**Go Idle on Bus Idle Timeout**

Bit	Name	Reset	Access	Description

When set, the bus automatically goes idle on a bus idle timeout, allowing new transfers to be initiated.

Value	Description
0	A bus idle timeout has no effect on the bus state.
1	A bus idle timeout tells the I^2C module that the bus is idle, allowing new transfers to be initiated.

14 Reserved — To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)

13:12 BITO 0x0 RW **Bus Idle Timeout**

Use to generate a timeout when SCL has been high for a given amount time between a START and STOP condition. When in a bus transaction, i.e. the BUSY flag is set, a timer is started whenever SCL goes high. When the timer reaches the value defined by BITO, it sets the BITO interrupt flag. The BITO interrupt flag will then be set periodically as long as SCL remains high. The bus idle timeout is active as long as BUSY is set. It is thus stopped automatically on a timeout if GIBITO is set. It is also stopped a STOP condition is detected and when the ABORT command is issued. The timeout is activated whenever the bus goes BUSY, i.e. a START condition is detected.

Value	Mode	Description
0	OFF	Timeout disabled
1	40PCC	Timeout after 40 prescaled clock cycles. In standard mode at 100 kHz, this results in a 50us timeout.
2	80PCC	Timeout after 80 prescaled clock cycles. In standard mode at 100 kHz, this results in a 100us timeout.
3	160PCC	Timeout after 160 prescaled clock cycles. In standard mode at 100 kHz, this results in a 200us timeout.

11:10 Reserved — To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)

9:8 CLHR 0x0 RW **Clock Low High Ratio**

Determines the ratio between the low and high parts of the clock signal generated on SCL as master.

Value	Mode	Description
0	STANDARD	The ratio between low period and high period counters (N_{low}:N_{high}) is 4:4
1	ASYMMETRIC	The ratio between low period and high period counters (N_{low}:N_{high}) is 6:3
2	FAST	The ratio between low period and high period counters (N_{low}:N_{high}) is 11:6

7 Reserved — To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)

6 GCAMEN 0 RW **General Call Address Match Enable**

Set to enable address match on general call in addition to the programmed slave address.

Value	Description
0	General call address will be NACK'ed if it is not included by the slave address and address mask.
1	When a general call address is received, a software response is required.

5 ARBDIS 0 RW **Arbitration Disable**

A master or slave will not release the bus upon losing arbitration.

Value	Description
0	When a device loses arbitration, the ARB interrupt flag is set and the bus is released.
1	When a device loses arbitration, the ARB interrupt flag is set, but communication proceeds.

4 AUTOSN 0 RW **Automatic STOP on NACK**

Write to 1 to make a master transmitter send a STOP when a NACK is received from a slave.

Value	Description
0	Stop is not automatically sent if a NACK is received from a slave.
1	The master automatically sends a STOP if a NACK is received from a slave.

3 AUTOSE 0 RW **Automatic STOP when Empty**

Write to 1 to make a master transmitter send a STOP when no more data is available for transmission.

Value	Description
0	A stop must be sent manually when no more data is to be transmitted.
1	The master automatically sends a STOP when no more data is available for transmission.

2 AUTOACK 0 RW **Automatic Acknowledge**

Set to enable automatic acknowledges.

Bit	Name	Reset	Access	Description

Value	Description
0	Software must give one ACK command for each ACK transmitted on the I2C bus.
1	Addresses that are not automatically NACK'ed, and all data is automatically acknowledged.

1 SLAVE 0 RW **Addressable as Slave**

Set this bit to allow the device to be selected as an I2C slave.

Value	Description
0	All addresses will be responded to with a NACK
1	Addresses matching the programmed slave address or the general call address (if enabled) require a response from software. Other addresses are automatically responded to with a NACK.

0 EN 0 RW **I2C Enable**

Use this bit to enable or disable the I2C module.

Value	Description
0	The I2C module is disabled. And its internal state is cleared
1	The I2C module is enabled.

16.5.2 I2Cn_CMD - Command Register

Offset	Bit Position																													
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0	0	0	0	0	0	0	0
Access																									W1	W1	W1	W1	W1	W1	W1	W1
Name																									CLEARPC	CLEARTX	ABORT	CONT	NACK	ACK	STOP	START

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7	CLEARPC	0	W1	**Clear Pending Commands**
	Set to clear pending commands.			
6	CLEARTX	0	W1	**Clear TX**
	Set to clear transmit buffer and shift register. Will not abort ongoing transfer.			
5	ABORT	0	W1	**Abort transmission**
	Abort the current transmission making the bus go idle. When used in combination with STOP, a STOP condition is sent as soon as possible before aborting the transmission. The stop condition is subject to clock synchronization.			
4	CONT	0	W1	**Continue transmission**
	Set to continue transmission after a NACK has been received.			
3	NACK	0	W1	**Send NACK**
	Set to transmit a NACK the next time an acknowledge is required.			
2	ACK	0	W1	**Send ACK**
	Set to transmit an ACK the next time an acknowledge is required.			
1	STOP	0	W1	**Send stop condition**
	Set to send stop condition as soon as possible.			
0	START	0	W1	**Send start condition**
	Set to send start condition as soon as possible. If a transmission is ongoing and not owned, the start condition will be sent as soon as the bus is idle. If the current transmission is owned by this module, a repeated start condition will be sent. Use in combination with a STOP command to automatically send a STOP, then a START when the bus becomes idle.			

16.5.3 I2Cn_STATE - State Register

Offset	Bit Position																														
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0x0			0	0	0	0	1
Access																									R			R	R	R	R	R
Name																									STATE			BUSHOLD	NACKED	TRANSMITTER	MASTER	BUSY

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:5	STATE	0x0	R	**Transmission State**

The state of any current transmission. Cleared if the I^{2}C module is idle.

Value	Mode	Description
0	IDLE	No transmission is being performed.
1	WAIT	Waiting for idle. Will send a start condition as soon as the bus is idle.
2	START	Start transmitted or received
3	ADDR	Address transmitted or received
4	ADDRACK	Address ack/nack transmitted or received
5	DATA	Data transmitted or received
6	DATAACK	Data ack/nack transmitted or received

Bit	Name	Reset	Access	Description
4	BUSHOLD	0	R	**Bus Held**

Set if the bus is currently being held by this I^{2}C module.

Bit	Name	Reset	Access	Description
3	NACKED	0	R	**Nack Received**

Set if a NACK was received and STATE is ADDRACK or DATAACK.

Bit	Name	Reset	Access	Description
2	TRANSMITTER	0	R	**Transmitter**

Set when operating as a master transmitter or a slave transmitter. When cleared, the system may be operating as a master receiver, a slave receiver or the current mode is not known.

Bit	Name	Reset	Access	Description
1	MASTER	0	R	**Master**

Set when operating as an I^{2}C master. When cleared, the system may be operating as an I^{2}C slave.

Bit	Name	Reset	Access	Description
0	BUSY	1	R	**Bus Busy**

Set when the bus is busy. Whether the I^{2}C module is in control of the bus or not has no effect on the value of this bit. When the MCU comes out of reset, the state of the bus is not known, and thus BUSY is set. Use the ABORT command or a bus idle timeout to force the I^{2}C module out of the BUSY state.

16.5.4 I2Cn_STATUS - Status Register

Offset	Bit Position																														
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																								0	1	0	0	0	0	0	0	0
Access																								R	R	R	R	R	R	R	R	R
Name																								RXDATAV	TXBL	TXC	PABORT	PCONT	PNACK	PACK	PSTOP	PSTART

Bit	Name	Reset	Access	Description
31:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
8	RXDATAV	0	R	**RX Data Valid**
	Set when data is available in the receive buffer. Cleared when the receive buffer is empty.			
7	TXBL	1	R	**TX Buffer Level**
	Indicates the level of the transmit buffer. Set when the transmit buffer is empty, and cleared when it is full.			
6	TXC	0	R	**TX Complete**
	Set when a transmission has completed and no more data is available in the transmit buffer. Cleared when a new transmission starts.			
5	PABORT	0	R	**Pending abort**
	An abort is pending and will be transmitted as soon as possible.			
4	PCONT	0	R	**Pending continue**
	A continue is pending and will be transmitted as soon as possible.			
3	PNACK	0	R	**Pending NACK**
	A not-acknowledge is pending and will be transmitted as soon as possible.			
2	PACK	0	R	**Pending ACK**
	An acknowledge is pending and will be transmitted as soon as possible.			
1	PSTOP	0	R	**Pending STOP**
	A stop condition is pending and will be transmitted as soon as possible.			
0	PSTART	0	R	**Pending START**
	A start condition is pending and will be transmitted as soon as possible.			

16.5.5 I2Cn_CLKDIV - Clock Division Register

Offset	Bit Position																													
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																								0x000								
Access																								RW								
Name																								DIV								

Bit	Name	Reset	Access	Description
31:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:0	DIV	0x000	RW	**Clock Divider**
	Specifies the clock divider for the I^2C. Note that DIV must be 1 or higher when slave is enabled.			

16.5.6 I2Cn_SADDR - Slave Address Register

Offset	Bit Position																													
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0x00							
Access																									RW							
Name																									ADDR							

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:1	ADDR	0x00	RW	**Slave address**
	Specifies the slave address of the device.			
0	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

16.5.7 I2Cn_SADDRMASK - Slave Address Mask Register

Offset	Bit Position																													
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0x00							
Access																									RW							
Name																									MASK							

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:1	MASK	0x00	RW	**Slave Address Mask**
	Specifies the significant bits of the slave address. Setting the mask to 0x00 will match all addresses, while setting it to 0x7F will only match the exact address specified by ADDR.			
0	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

16.5.8 I2Cn_RXDATA - Receive Buffer Data Register

Offset	Bit Position																													
0x01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0x00							
Access																									R							
Name																									RXDATA							

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:0	RXDATA	0x00	R	**RX Data**
	Use this register to read from the receive buffer. Buffer is emptied on read access.			

16.5.9 I2Cn_RXDATAP - Receive Buffer Data Peek Register

Offset	Bit Position																															
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0x00							
Access																									R							
Name																									RXDATAP							

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:0	RXDATAP	0x00	R	**RX Data Peek**
	Use this register to read from the receive buffer. Buffer is not emptied on read access.			

16.5.10 I2Cn_TXDATA - Transmit Buffer Data Register

Offset	Bit Position																															
0x024	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0x00							
Access																									W							
Name																									TXDATA							

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:0	TXDATA	0x00	W	**TX Data**
	Use this register to write a byte to the transmit buffer.			

16.5.11 I2Cn_IF - Interrupt Flag Register

Offset	Bit Position																															
0x028	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Access																R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Name																SSTOP	CLTO	BITO	RXUF	TXOF	BUSHOLD	BUSERR	ARBLOST	MSTOP	NACK	ACK	RXDATAV	TXBL	TXC	ADDR	RSTART	START

Bit	Name	Reset	Access	Description
31:17	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
16	SSTOP	0	R	**Slave STOP condition Interrupt Flag**
	Set when a STOP condition has been received. Will be set regardless of the EZR32 being involved in the transaction or not.			
15	CLTO	0	R	**Clock Low Timeout Interrupt Flag**

Bit	Name	Reset	Access	Description
	Set on each clock low timeout. The timeout value can be set in CLTO bit field in the I2Cn_CTRL register.			
14	BITO	0	R	**Bus Idle Timeout Interrupt Flag**
	Set on each bus idle timeout. The timeout value can be set in the BITO bit field in the I2Cn_CTRL register.			
13	RXUF	0	R	**Receive Buffer Underflow Interrupt Flag**
	Set when data is read from the receive buffer through the I2Cn_RXDATA register while the receive buffer is empty.			
12	TXOF	0	R	**Transmit Buffer Overflow Interrupt Flag**
	Set when data is written to the transmit buffer while the transmit buffer is full.			
11	BUSHOLD	0	R	**Bus Held Interrupt Flag**
	Set when the bus becomes held by the I^2C module.			
10	BUSERR	0	R	**Bus Error Interrupt Flag**
	Set when a bus error is detected. The bus error is resolved automatically, but the current transfer is aborted.			
9	ARBLOST	0	R	**Arbitration Lost Interrupt Flag**
	Set when arbitration is lost.			
8	MSTOP	0	R	**Master STOP Condition Interrupt Flag**
	Set when a STOP condition has been successfully transmitted. If arbitration is lost during the transmission of the STOP condition, then the MSTOP interrupt flag is not set.			
7	NACK	0	R	**Not Acknowledge Received Interrupt Flag**
	Set when a NACK has been received.			
6	ACK	0	R	**Acknowledge Received Interrupt Flag**
	Set when an ACK has been received.			
5	RXDATAV	0	R	**Receive Data Valid Interrupt Flag**
	Set when data is available in the receive buffer. Cleared automatically when the receive buffer is read.			
4	TXBL	1	R	**Transmit Buffer Level Interrupt Flag**
	Set when the transmit buffer becomes empty. Cleared automatically when new data is written to the transmit buffer.			
3	TXC	0	R	**Transfer Completed Interrupt Flag**
	Set when the transmit shift register becomes empty and there is no more data in the transmit buffer.			
2	ADDR	0	R	**Address Interrupt Flag**
	Set when incoming address is accepted, i.e. own address or general call address is received.			
1	RSTART	0	R	**Repeated START condition Interrupt Flag**
	Set when a repeated start condition is detected.			
0	START	0	R	**START condition Interrupt Flag**
	Set when a start condition is successfully transmitted.			

16.5.12 I2Cn_IFS - Interrupt Flag Set Register

Offset	Bit Position																														
0x02C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																0	0	0	0	0	0	0	0	0	0	0			0	0	0	0
Access																W1	W1	W1	W1	W1	W1	W1	W1	W1	W1	W1			W1	W1	W1	W1
Name																SSTOP	CLTO	BITO	RXUF	TXOF	BUSHOLD	BUSERR	ARBLOST	MSTOP	NACK	ACK			TXC	ADDR	RSTART	START

Bit	Name	Reset	Access	Description
31:17	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
16	SSTOP	0	W1	**Set SSTOP Interrupt Flag**
	Write to 1 to set the SSTOP interrupt flag.			
15	CLTO	0	W1	**Set Clock Low Interrupt Flag**
	Write to 1 to set the CLTO interrupt flag.			
14	BITO	0	W1	**Set Bus Idle Timeout Interrupt Flag**
	Write to 1 to set the BITO interrupt flag.			
13	RXUF	0	W1	**Set Receive Buffer Underflow Interrupt Flag**
	Write to 1 to set the RXUF interrupt flag.			
12	TXOF	0	W1	**Set Transmit Buffer Overflow Interrupt Flag**
	Write to 1 to set the TXOF interrupt flag.			
11	BUSHOLD	0	W1	**Set Bus Held Interrupt Flag**
	Write to 1 to set the BUSHOLD interrupt flag.			
10	BUSERR	0	W1	**Set Bus Error Interrupt Flag**
	Write to 1 to set the BUSERR interrupt flag.			
9	ARBLOST	0	W1	**Set Arbitration Lost Interrupt Flag**
	Write to 1 to set the ARBLOST interrupt flag.			
8	MSTOP	0	W1	**Set MSTOP Interrupt Flag**
	Write to 1 to set the MSTOP interrupt flag.			
7	NACK	0	W1	**Set Not Acknowledge Received Interrupt Flag**
	Write to 1 to set the NACK interrupt flag.			
6	ACK	0	W1	**Set Acknowledge Received Interrupt Flag**
	Write to 1 to set the ACK interrupt flag.			
5:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	TXC	0	W1	**Set Transfer Completed Interrupt Flag**
	Write to 1 to set the TXC interrupt flag.			
2	ADDR	0	W1	**Set Address Interrupt Flag**
	Write to 1 to set the ADDR interrupt flag.			
1	RSTART	0	W1	**Set Repeated START Interrupt Flag**
	Write to 1 to set the RSTART interrupt flag.			
0	START	0	W1	**Set START Interrupt Flag**
	Write to 1 to set the START interrupt flag.			

16.5.13 I2Cn_IFC - Interrupt Flag Clear Register

Offset	Bit Position																															
0x030	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																0	0	0	0	0	0	0	0	0	0	0			0	0	0	0
Access																W1	W1	W1	W1	W1	W1	W1	W1	W1	W1	W1			W1	W1	W1	W1
Name																SSTOP	CLTO	BITO	RXUF	TXOF	BUSHOLD	BUSERR	ARBLOST	MSTOP	NACK	ACK			TXC	ADDR	RSTART	START

Bit	Name	Reset	Access	Description
31:17	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
16	SSTOP	0	W1	**Clear SSTOP Interrupt Flag**

Bit	Name	Reset	Access	Description
	Write to 1 to clear the SSTOP interrupt flag.			
15	CLTO	0	W1	**Clear Clock Low Interrupt Flag**
	Write to 1 to clear the CLTO interrupt flag.			
14	BITO	0	W1	**Clear Bus Idle Timeout Interrupt Flag**
	Write to 1 to clear the BITO interrupt flag.			
13	RXUF	0	W1	**Clear Receive Buffer Underflow Interrupt Flag**
	Write to 1 to clear the RXUF interrupt flag.			
12	TXOF	0	W1	**Clear Transmit Buffer Overflow Interrupt Flag**
	Write to 1 to clear the TXOF interrupt flag.			
11	BUSHOLD	0	W1	**Clear Bus Held Interrupt Flag**
	Write to 1 to clear the BUSHOLD interrupt flag.			
10	BUSERR	0	W1	**Clear Bus Error Interrupt Flag**
	Write to 1 to clear the BUSERR interrupt flag.			
9	ARBLOST	0	W1	**Clear Arbitration Lost Interrupt Flag**
	Write to 1 to clear the ARBLOST interrupt flag.			
8	MSTOP	0	W1	**Clear MSTOP Interrupt Flag**
	Write to 1 to clear the MSTOP interrupt flag.			
7	NACK	0	W1	**Clear Not Acknowledge Received Interrupt Flag**
	Write to 1 to clear the NACK interrupt flag.			
6	ACK	0	W1	**Clear Acknowledge Received Interrupt Flag**
	Write to 1 to clear the ACK interrupt flag.			
5:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	TXC	0	W1	**Clear Transfer Completed Interrupt Flag**
	Write to 1 to clear the TXC interrupt flag.			
2	ADDR	0	W1	**Clear Address Interrupt Flag**
	Write to 1 to clear the ADDR interrupt flag.			
1	RSTART	0	W1	**Clear Repeated START Interrupt Flag**
	Write to 1 to clear the RSTART interrupt flag.			
0	START	0	W1	**Clear START Interrupt Flag**
	Write to 1 to clear the START interrupt flag.			

16.5.14 I2Cn_IEN - Interrupt Enable Register

Offset	Bit Position																															
0x034	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access																RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Name																SSTOP	CLTO	BITO	RXUF	TXOF	BUSHOLD	BUSERR	ARBLOST	MSTOP	NACK	ACK	RXDATAV	TXBL	TXC	ADDR	RSTART	START

Bit	Name	Reset	Access	Description
31:17	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
16	SSTOP	0	RW	**SSTOP Interrupt Enable**
	Enable interrupt on SSTOP.			

Bit	Name	Reset	Access	Description
15	CLTO	0	RW	**Clock Low Interrupt Enable**
	Enable interrupt on clock low timeout.			
14	BITO	0	RW	**Bus Idle Timeout Interrupt Enable**
	Enable interrupt on bus idle timeout.			
13	RXUF	0	RW	**Receive Buffer Underflow Interrupt Enable**
	Enable interrupt on receive buffer underflow.			
12	TXOF	0	RW	**Transmit Buffer Overflow Interrupt Enable**
	Enable interrupt on transmit buffer overflow.			
11	BUSHOLD	0	RW	**Bus Held Interrupt Enable**
	Enable interrupt on bus-held.			
10	BUSERR	0	RW	**Bus Error Interrupt Enable**
	Enable interrupt on bus error.			
9	ARBLOST	0	RW	**Arbitration Lost Interrupt Enable**
	Enable interrupt on loss of arbitration.			
8	MSTOP	0	RW	**MSTOP Interrupt Enable**
	Enable interrupt on MSTOP.			
7	NACK	0	RW	**Not Acknowledge Received Interrupt Enable**
	Enable interrupt when not-acknowledge is received.			
6	ACK	0	RW	**Acknowledge Received Interrupt Enable**
	Enable interrupt on acknowledge received.			
5	RXDATAV	0	RW	**Receive Data Valid Interrupt Enable**
	Enable interrupt on receive buffer full.			
4	TXBL	0	RW	**Transmit Buffer level Interrupt Enable**
	Enable interrupt on transmit buffer level.			
3	TXC	0	RW	**Transfer Completed Interrupt Enable**
	Enable interrupt on transfer completed.			
2	ADDR	0	RW	**Address Interrupt Enable**
	Enable interrupt on recognized address.			
1	RSTART	0	RW	**Repeated START condition Interrupt Enable**
	Enable interrupt on transmitted or received repeated START condition.			
0	START	0	RW	**START Condition Interrupt Enable**
	Enable interrupt on transmitted or received START condition.			

16.5.15 I2Cn_ROUTE - I/O Routing Register

Offset	Bit Position																													
0x038	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0x0								0	0
Access																							RW								RW	RW
Name																							LOCATION								SCLPEN	SDAPEN

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
10:8	LOCATION	0x0	RW	**I/O Location**

Decides the location of the I^2C I/O pins.

Value	Mode	Description
0	LOC0	Location 0
1	LOC1	Location 1
2	LOC2	Location 2
3	LOC3	Location 3
4	LOC4	Location 4
5	LOC5	Location 5
6	LOC6	Location 6

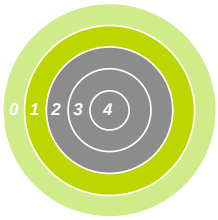
7:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	SCLPEN	0	RW	**SCL Pin Enable**

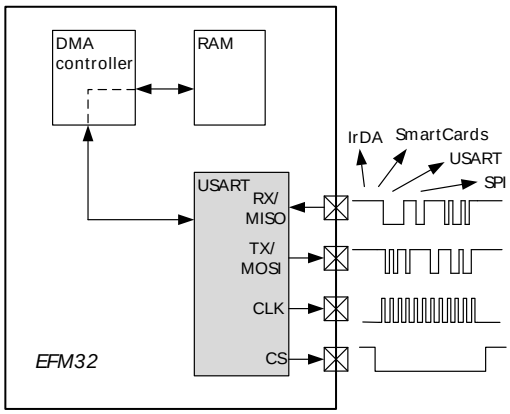
When set, the SCL pin of the I^2C is enabled.

0	SDAPEN	0	RW	**SDA Pin Enable**

When set, the SDA pin of the I^2C is enabled.

17 USART - Universal Synchronous Asynchronous Receiver/Transmitter





Quick Facts

What?

The USART handles high-speed UART, SPI-bus, SmartCards, and IrDA communication.

Why?

Serial communication is frequently used in embedded systems and the USART allows efficient communication with a wide range of external devices.

How?

The USART has a wide selection of operating modes, frame formats and baud rates. The multi-processor mode allows the USART to remain idle when not addressed. Triple buffering and DMA support makes high data-rates possible with minimal CPU intervention and it is possible to transmit and receive large frames while the MCU remains in EM1.

17.1 Introduction

The Universal Synchronous Asynchronous serial Receiver and Transmitter (USART) is a very flexible serial I/O module. It supports full duplex asynchronous UART communication as well as RS-485, SPI, MicroWire and 3-wire. It can also interface with ISO7816 SmartCards, and IrDA devices.

17.2 Features

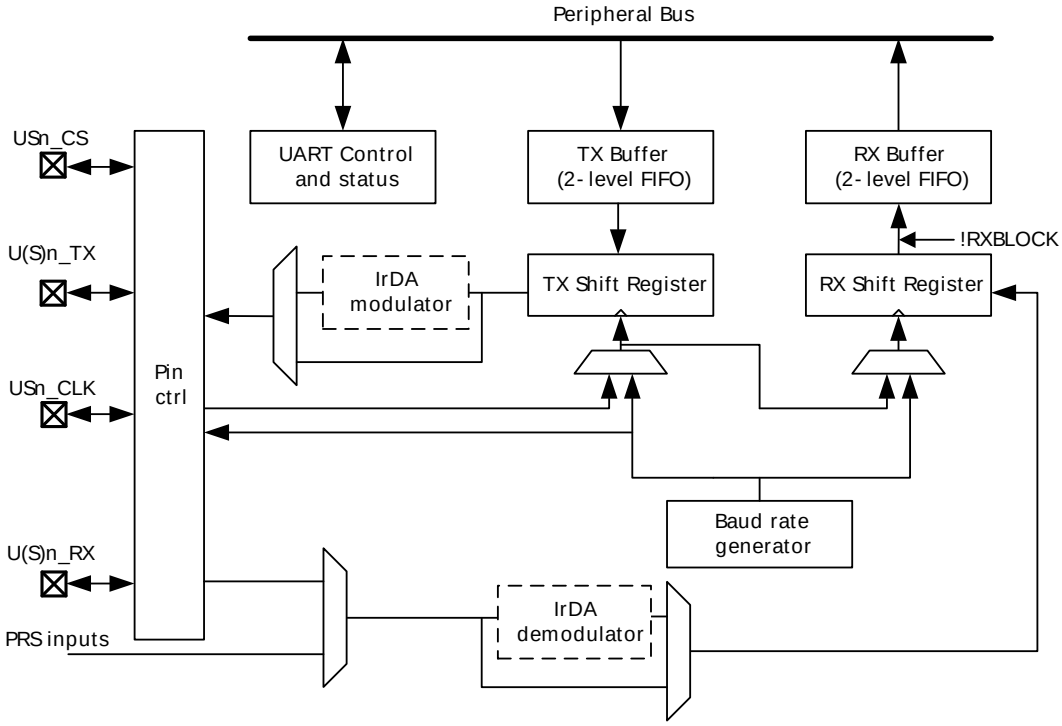
- Asynchronous and synchronous (SPI) communication
- Full duplex and half duplex
- Separate TX/RX enable
- Separate receive / transmit 2-level buffers, with additional separate shift registers
- Programmable baud rate, generated as an fractional division from the peripheral clock ($HFPERCLK_{USARTn}$)
- Max bit-rate
  - SPI master mode, peripheral clock rate/2
  - SPI slave mode, peripheral clock rate/8
  - UART mode, peripheral clock rate/16, 8, 6, or 4
- Asynchronous mode supports
  - Majority vote baud-reception
  - False start-bit detection
  - Break generation/detection
  - Multi-processor mode
- Synchronous mode supports
  - All 4 SPI clock polarity/phase configurations
  - Master and slave mode
- Data can be transmitted LSB first or MSB first
- Configurable number of data bits, 4-16 (plus the parity bit, if enabled)

  - HW parity bit generation and check
- Configurable number of stop bits in asynchronous mode: 0.5, 1, 1.5, 2
- HW collision detection
- Multi-processor mode
- IrDA modulator on USART0
- SmartCard (ISO7816) mode
- I2S mode
- Separate interrupt vectors for receive and transmit interrupts
- Loopback mode
  - Half duplex communication
  - Communication debugging
- PRS RX input

17.3 Functional Description

An overview of the USART module is shown in Figure 17.1 (p. 369) .

Figure 17.1. USART Overview



17.3.1 Modes of Operation

The USART operates in either asynchronous or synchronous mode.

In synchronous mode, a separate clock signal is transmitted with the data. This clock signal is generated by the bus master, and both the master and slave sample and transmit data according to this clock. Both master and slave modes are supported by the USART. The synchronous communication mode is compatible with the Serial Peripheral Interface Bus (SPI) standard.

In asynchronous mode, no separate clock signal is transmitted with the data on the bus. The USART receiver thus has to determine where to sample the data on the bus from the actual data. To make this possible, additional synchronization bits are added to the data when operating in asynchronous mode, resulting in a slight overhead.

Asynchronous or synchronous mode can be selected by configuring SYNC in USARTn_CTRL. The options are listed with supported protocols in Table 17.1 (p. 370) . Full duplex and half duplex communication is supported in both asynchronous and synchronous mode.

Table 17.1. USART Asynchronous vs. Synchronous Mode

SYNC	Communication Mode	Supported Protocols
0	Asynchronous	RS-232, RS-485 (w/external driver), IrDA, ISO 7816
1	Synchronous	SPI, MicroWire, 3-wire

Table 17.2 (p. 370) explains the functionality of the different USART pins when the USART operates in different modes. Pin functionality enclosed in square brackets is optional, and depends on additional configuration parameters. LOOPBK and MASTER are discussed in Section 17.3.2.5 (p. 378) and Section 17.3.3.3 (p. 386) respectively.

Table 17.2. USART Pin Usage

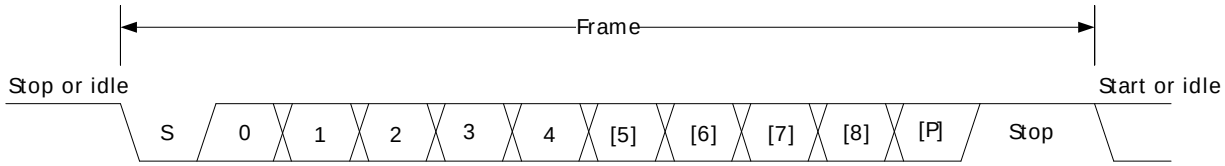
SYNC	LOOPBK	MASTER	Pin functionality			
			U(S)n_TX (MOSI)	U(S)n_RX (MISO)	USn_CLK	USn_CS
0	0	x	Data out	Data in	-	[Driver enable]
1	1	x	Data out/in	-	-	[Driver enable]
1	0	0	Data in	Data out	Clock in	Slave select
1	0	1	Data out	Data in	Clock out	[Auto slave select]
1	1	0	Data out/in	-	Clock in	Slave select
1	1	1	Data out/in	-	Clock out	[Auto slave select]

17.3.2 Asynchronous Operation

17.3.2.1 Frame Format

The frame format used in asynchronous mode consists of a set of data bits in addition to bits for synchronization and optionally a parity bit for error checking. A frame starts with one start-bit (S), where the line is driven low for one bit-period. This signals the start of a frame, and is used for synchronization. Following the start bit are 4 to 16 data bits and an optional parity bit. Finally, a number of stop-bits, where the line is driven high, end the frame. An example frame is shown in Figure 17.2 (p. 370) .

Figure 17.2. USART Asynchronous Frame Format



The number of data bits in a frame is set by DATABITS in USARTn_FRAME, see Table 17.3 (p. 371) , and the number of stop-bits is set by STOPBITS in USARTn_FRAME, see Table 17.4 (p. 371) . Whether or not a parity bit should be included, and whether it should be even or odd is defined by PARITY, also in USARTn_FRAME. For communication to be possible, all parties of an asynchronous transfer must agree on the frame format being used.

Table 17.3. USART Data Bits

DATA BITS [3:0]	Number of Data bits
0001	4
0010	5
0011	6
0100	7
0101	8 (Default)
0110	9
0111	10
1000	11
1001	12
1010	13
1011	14
1100	15
1101	16

Table 17.4. USART Stop Bits

STOP BITS [1:0]	Number of Stop bits
00	0.5
01	1 (Default)
10	1.5
11	2

The order in which the data bits are transmitted and received is defined by MSBF in USARTn_CTRL. When MSBF is cleared, data in a frame is sent and received with the least significant bit first. When it is set, the most significant bit comes first.

The frame format used by the transmitter can be inverted by setting TXINV in USARTn_CTRL, and the format expected by the receiver can be inverted by setting RXINV in USARTn_CTRL. These bits affect the entire frame, not only the data bits. An inverted frame has a low idle state, a high start-bit, inverted data and parity bits, and low stop-bits.

17.3.2.1.1 Parity bit Calculation and Handling

When parity bits are enabled, hardware automatically calculates and inserts any parity bits into outgoing frames, and verifies the received parity bits in incoming frames. This is true for both asynchronous and synchronous modes, even though it is mostly used in asynchronous communication. The possible parity modes are defined in Table 17.5 (p. 372) . When even parity is chosen, a parity bit is inserted to make the number of high bits (data + parity) even. If odd parity is chosen, the parity bit makes the total number of high bits odd.

Table 17.5. USART Parity Bits

STOP BITS [1:0]	Description
00	No parity bit (Default)
01	Reserved
10	Even parity
11	Odd parity

17.3.2.2 Clock Generation

The USART clock defines the transmission and reception data rate. When operating in asynchronous mode, the baud rate (bit-rate) is given by Equation 17.1 (p. 372)

USART Baud Rate

$$br = f_{HFPERCLK}/(oversample \times (1 + USARTn\_CLKDIV/256)) \quad (17.1)$$

where $f_{HFPERCLK}$ is the peripheral clock ($HFPERCLK_{USARTn}$) frequency and oversample is the oversampling rate as defined by OVS in USARTn_CTRL, see Table 17.6 (p. 372) .

Table 17.6. USART Oversampling

OVS [1:0]	oversample
00	16
01	8
10	6
11	4

The USART has a fractional clock divider to allow the USART clock to be controlled more accurately than what is possible with a standard integral divider.

The clock divider used in the USART is a 15-bit value, with a 13-bit integral part and a 2-bit fractional part. The fractional part is configured in the two LSBs of DIV in USART_CLKDIV. The lowest achievable baud rate at 32 MHz is about 244 bauds/sec.

Fractional clock division is implemented by distributing the selected fraction over four baud periods. The fractional part of the divider tells how many of these periods should be extended by one peripheral clock cycle.

Given a desired baud rate $br_{desired}$, the clock divider USARTn_CLKDIV can be calculated by using Equation 17.2 (p. 372) :

USART Desired Baud Rate

$$USARTn\_CLKDIV = 256 \times (f_{HFPERCLK}/(oversample \times br_{desired}) - 1) \quad (17.2)$$

Table 17.7 (p. 373) shows a set of desired baud rates and how accurately the USART is able to generate these baud rates when running at a 4 MHz peripheral clock, using 16x or 8x oversampling.

Table 17.7. USART Baud Rates @ 4MHz Peripheral Clock

Desired baud rate [baud/s]	USARTn_OVS =00			USARTn_OVS =01		
	USARTn_CLKDIV/256	Actual baud rate [baud/s]	Error %	USARTn_CLKDIV/256	Actual baud rate [baud/s]	Error %
600	415,75	599,88	-0,02	832,25	600,06	0,01
1200	207,25	1200,48	0,04	415,75	1199,76	-0,02
2400	103,25	2398,082	-0,08	207,25	2400,96	0,04
4800	51	4807,692	0,16	103,25	4796,163	-0,08
9600	25	9615,385	0,16	51	9615,385	0,16
14400	16,25	14492,75	0,64	33,75	14388,49	-0,08
19200	12	19230,77	0,16	25	19230,77	0,16
28800	7,75	28571,43	-0,79	16,25	28985,51	0,64
38400	5,5	38461,54	0,16	12	38461,54	0,16
57600	3,25	58823,53	2,12	7,75	57142,86	-0,79
76800	2,25	76923,08	0,16	5,5	76923,08	0,16
115200	1,25	111111,1	-3,55	3,25	117647,1	2,12
230400	0	250000	8,51	1,25	222222,2	-3,55

17.3.2.3 Data Transmission

Asynchronous data transmission is initiated by writing data to the transmit buffer using one of the methods described in Section 17.3.2.3.1 (p. 373) . When the transmission shift register is empty and ready for new data, a frame from the transmit buffer is loaded into the shift register, and if the transmitter is enabled, transmission begins. When the frame has been transmitted, a new frame is loaded into the shift register if available, and transmission continues. If the transmit buffer is empty, the transmitter goes to an idle state, waiting for a new frame to become available.

Transmission is enabled through the command register USARTn_CMD by setting TXEN, and disabled by setting TXDIS in the same command register. When the transmitter is disabled using TXDIS, any ongoing transmission is aborted, and any frame currently being transmitted is discarded. When disabled, the TX output goes to an idle state, which by default is a high value. Whether or not the transmitter is enabled at a given time can be read from TXENS in USARTn_STATUS.

When the USART transmitter is enabled and there is no data in the transmit shift register or transmit buffer, the TXC flag in USARTn_STATUS and the TXC interrupt flag in USARTn_IF are set, signaling that the transmitter is idle. The TXC status flag is cleared when a new frame becomes available for transmission, but the TXC interrupt flag must be cleared by software.

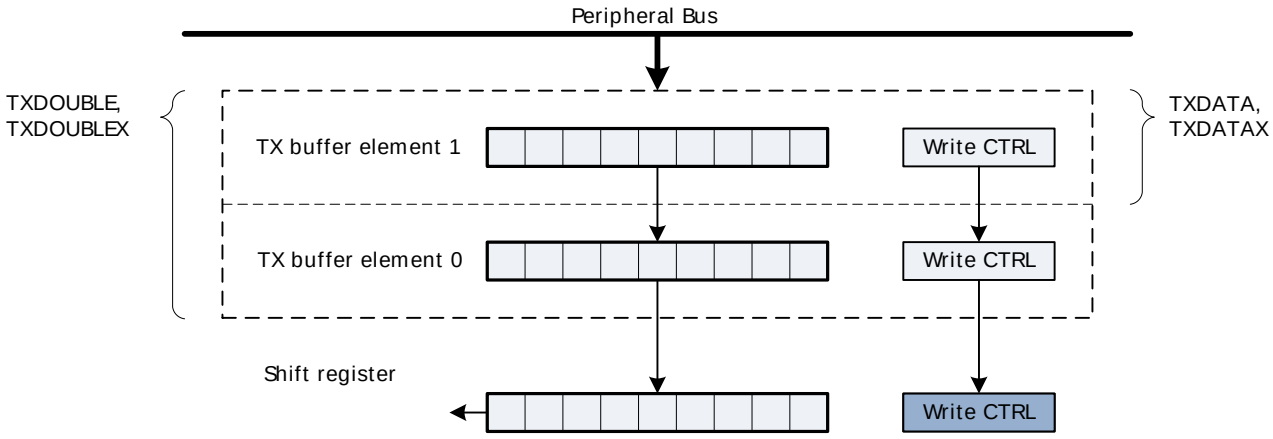
17.3.2.3.1 Transmit Buffer Operation

The transmit-buffer is a 2-level FIFO buffer. A frame can be loaded into the buffer by writing to USARTn_TXDATA, USARTn_TXDATAX, USARTn_TXDOUBLE or USARTn_TXDOUBLEX. Using USARTn_TXDATA allows 8 bits to be written to the buffer, while using USARTn_TXDOUBLE will write 2 frames of 8 bits to the buffer. If 9-bit frames are used, the 9th bit of the frames will in these cases be set to the value of BIT8DV in USARTn_CTRL.

To set the 9th bit directly and/or use transmission control, USARTn_TXDATAX and USARTn_TXDOUBLEX must be used. USARTn_TXDATAX allows 9 data bits to be written, as well as a set of control bits regarding the transmission of the written frame. Every frame in the buffer is stored with 9 data bits and additional transmission control bits. USARTn_TXDOUBLEX allows two

frames, complete with control bits to be written at once. When data is written to the transmit buffer using USARTn_TXDATAX and USARTn_TXDOUBLEX, the 9th bit(s) written to these registers override the value in BIT8DV in USARTn_CTRL, and alone define the 9th bits that are transmitted if 9-bit frames are used. Figure 17.3 (p. 374) shows the basics of the transmit buffer when DATABITS in USARTn_FRAME is configured to less than 10 bits.

Figure 17.3. USART Transmit Buffer Operation



When writing more frames to the transmit buffer than there is free space for, the TXOF interrupt flag in USARTn_IF will be set, indicating the overflow. The data already in the transmit buffer is preserved in this case, and no data is written.

In addition to the interrupt flag TXC in USARTn_IF and status flag TXC in USARTn_STATUS which are set when the transmitter is idle, TXBL in USARTn_STATUS and the TXBL interrupt flag in USARTn_IF are used to indicate the level of the transmit buffer. TXBIL in USARTn_CTRL controls the level at which these bits are set. If TXBIL is cleared, they are set whenever the transmit buffer becomes empty, and if TXBIL is set, they are set whenever the transmit buffer goes from full to half-full or empty. Both the TXBL status flag and the TXBL interrupt flag are cleared automatically when their condition becomes false

The transmit buffer, including the transmit shift register can be cleared by setting CLEARTX in USARTn_CMD. This will prevent the USART from transmitting the data in the buffer and shift register, and will make them available for new data. Any frame currently being transmitted will not be aborted. Transmission of this frame will be completed.

17.3.2.3.2 Frame Transmission Control

The transmission control bits, which can be written using USARTn_TXDATAX and USARTn_TXDOUBLEX, affect the transmission of the written frame. The following options are available:

- Generate break: By setting TXBREAK, the output will be held low during the stop-bit period to generate a framing error. A receiver that supports break detection detects this state, allowing it to be used e.g. for framing of larger data packets. The line is driven high before the next frame is transmitted so the next start condition can be identified correctly by the recipient. Continuous breaks lasting longer than a USART frame are thus not supported by the USART. GPIO can be used for this.
- Disable transmitter after transmission: If TXDISAT is set, the transmitter is disabled after the frame has been fully transmitted.
- Enable receiver after transmission: If RXENAT is set, the receiver is enabled after the frame has been fully transmitted. It is enabled in time to detect a start-bit directly after the last stop-bit has been transmitted.
- Unblock receiver after transmission: If UBRXAT is set, the receiver is unblocked and RXBLOCK is cleared after the frame has been fully transmitted.

- Tristate transmitter after transmission: If TXTRIAT is set, TXTRI is set after the frame has been fully transmitted, tristating the transmitter output. Tristating of the output can also be performed automatically by setting AUTOTRI. If AUTOTRI is set TXTRI is always read as 0.

Note

When in SmartCard mode with repeat enabled, none of the actions, except generate break, will be performed until the frame is transmitted without failure. Generation of a break in SmartCard mode with repeat enabled will cause the USART to detect a NACK on every frame.

17.3.2.4 Data Reception

Data reception is enabled by setting RXEN in USARTn_CMD. When the receiver is enabled, it actively samples the input looking for a transition from high to low indicating the start baud of a new frame. When a start baud is found, reception of the new frame begins if the receive shift register is empty and ready for new data. When the frame has been received, it is pushed into the receive buffer, making the shift register ready for another frame of data, and the receiver starts looking for another start baud. If the receive buffer is full, the received frame remains in the shift register until more space in the receive buffer is available. If an incoming frame is detected while both the receive buffer and the receive shift register are full, the data in the shift register is overwritten, and the RXOF interrupt flag in USARTn_IF is set to indicate the buffer overflow.

The receiver can be disabled by setting the command bit RXDIS in USARTn_CMD. Any frame currently being received when the receiver is disabled is discarded. Whether or not the receiver is enabled at a given time can be read out from RXENS in USARTn_STATUS.

17.3.2.4.1 Receive Buffer Operation

When data becomes available in the receive buffer, the RXDATAV flag in USARTn_STATUS, and the RXDATAV interrupt flag in USARTn_IF are set, and when the buffer becomes full, RXFULL in USARTn_STATUS and the RXFULL interrupt flag in USARTn_IF are set. The status flags RXDATAV and RXFULL are automatically cleared by hardware when their condition is no longer true. This also goes for the RXDATAV interrupt flag, but the RXFULL interrupt flag must be cleared by software. When the RXFULL flag is set, notifying that the buffer is full, space is still available in the receive shift register for one more frame.

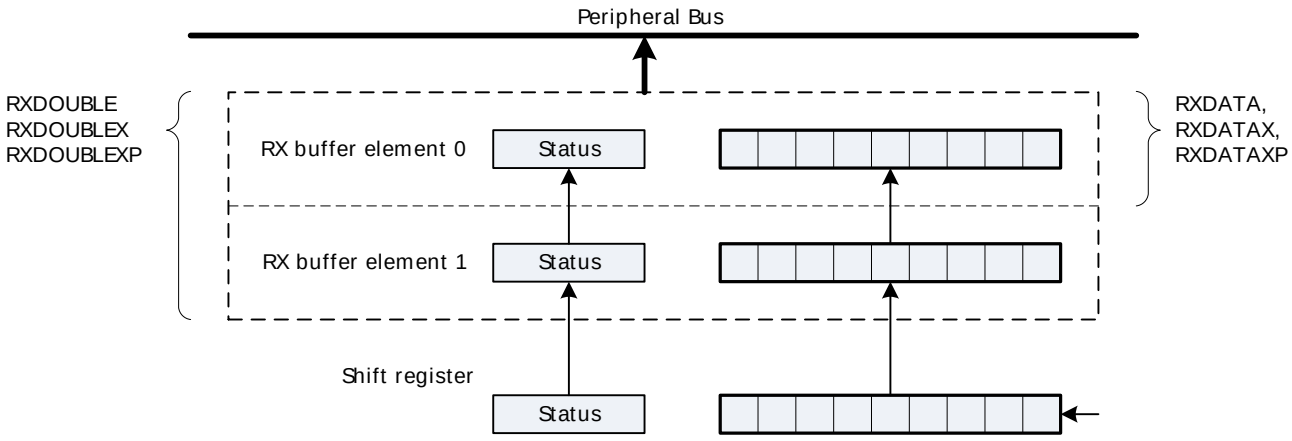
Data can be read from the receive buffer in a number of ways. USARTn_RXDATA gives access to the 8 least significant bits of the received frame, and USARTn_RXDOUBLE makes it possible to read the 8 least significant bits of two frames at once, pulling two frames from the buffer. To get access to the 9th, most significant bit, USARTn_RXDATAX must be used. This register also contains status information regarding the frame. USARTn_RXDOUBLEX can be used to get two frames complete with the 9th bits and status bits.

When a frame is read from the receive buffer using USARTn_RXDATA or USARTn_RXDATAX, the frame is pulled out of the buffer, making room for a new frame. USARTn_RXDOUBLE and USARTn_RXDOUBLEX pull two frames out of the buffer. If an attempt is done to read more frames from the buffer than what is available, the RXUF interrupt flag in USARTn_IF is set to signal the underflow, and the data read from the buffer is undefined.

Frames can be read from the receive buffer without removing the data by using USARTn_RXDATAXP and USARTn_RXDOUBLEXP. USARTn_RXDATAXP gives access the first frame in the buffer with status bits, while USARTn_RXDOUBLEXP gives access to both frames with status bits. The data read from these registers when the receive buffer is empty is undefined. If the receive buffer contains one valid frame, the first frame in USARTn_RXDOUBLEXP will be valid. No underflow interrupt is generated by a read using these registers, i.e. RXUF in USARTn_IF is never set as a result of reading from USARTn_RXDATAXP or USARTn_RXDOUBLEXP.

The basic operation of the receive buffer when DATABITS in USARTn_FRAME is configured to less than 10 bits is shown in Figure 17.4 (p. 376) .

Figure 17.4. USART Receive Buffer Operation



The receive buffer, including the receive shift register can be cleared by setting CLEARRX in USARTn_CMD. Any frame currently being received will not be discarded.

17.3.2.4.2 Blocking Incoming Data

When using hardware frame recognition, as detailed in Section 17.3.2.8 (p. 382) and Section 17.3.2.9 (p. 383) , it is necessary to be able to let the receiver sample incoming frames without passing the frames to software by loading them into the receive buffer. This is accomplished by blocking incoming data.

Incoming data is blocked as long as RXBLOCK in USARTn_STATUS is set. When blocked, frames received by the receiver will not be loaded into the receive buffer, and software is not notified by the RXDATAV flag in USARTn_STATUS or the RXDATAV interrupt flag in USARTn_IF at their arrival. For data to be loaded into the receive buffer, RXBLOCK must be cleared in the instant a frame is fully received by the receiver. RXBLOCK is set by setting RXBLOCKEN in USARTn_CMD and disabled by setting RXBLOCKDIS also in USARTn_CMD. There is one exception where data is loaded into the receive buffer even when RXBLOCK is set. This is when an address frame is received when operating in multi-processor mode. See Section 17.3.2.8 (p. 382) for more information.

Frames received containing framing or parity errors will not result in the FERR and PERR interrupt flags in USARTn_IF being set while RXBLOCK in USARTn_STATUS is set. Hardware recognition is not applied to these erroneous frames, and they are silently discarded.

Note

If a frame is received while RXBLOCK in USARTn_STATUS is cleared, but stays in the receive shift register because the receive buffer is full, the received frame will be loaded into the receive buffer when space becomes available even if RXBLOCK is set at that time.

The overflow interrupt flag RXOF in USARTn_IF will be set if a frame in the receive shift register, waiting to be loaded into the receive buffer is overwritten by an incoming frame even though RXBLOCK in USARTn_STATUS is set.

17.3.2.4.3 Clock Recovery and Filtering

The receiver samples the incoming signal at a rate 16, 8, 6 or 4 times higher than the given baud rate, depending on the oversampling mode given by OVS in USARTn_CTRL. Lower oversampling rates make higher baud rates possible, but give less room for errors.

When a high-to-low transition is registered on the input while the receiver is idle, this is recognized as a start-bit, and the baud rate generator is synchronized with the incoming frame.

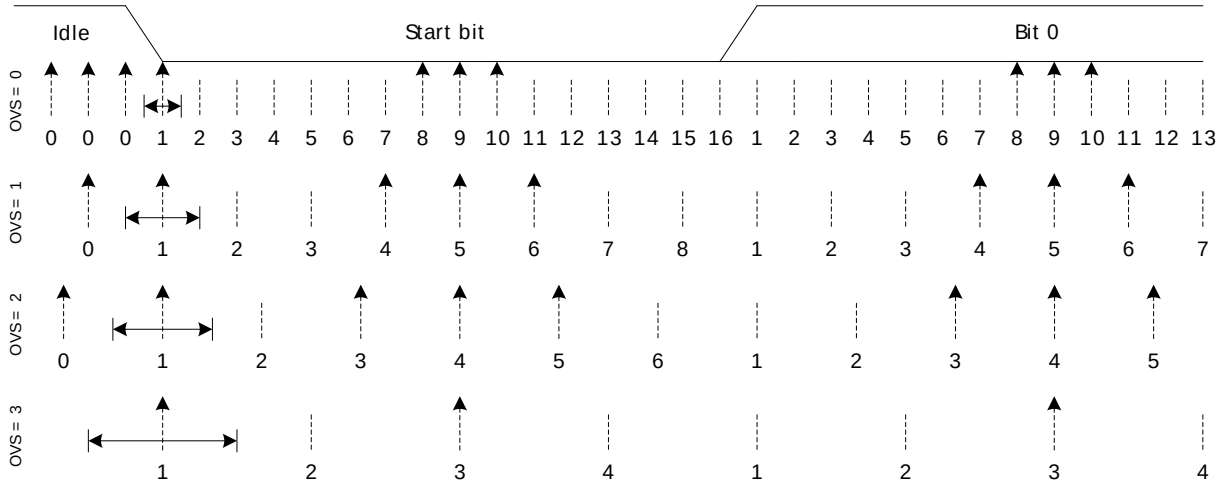
For oversampling modes 16, 8 and 6, every bit in the incoming frame is sampled three times to gain a level of noise immunity. These samples are aimed at the middle of the bit-periods, as visualized in Figure 17.5 (p. 377) . With OVS=0 in USARTn_CTRL, the start and data bits are thus sampled at locations 8, 9 and 10 in the figure, locations 4, 5 and 6 for OVS=1 and locations 3, 4, and 5 for OVS=2. The value of a sampled bit is determined by majority vote. If two or more of the three bit-samples are high, the resulting bit value is high. If the majority is low, the resulting bit value is low.

Majority vote is used for all oversampling modes except 4x oversampling. In this mode, a single sample is taken at position 3 as shown in Figure 17.5 (p. 377) .

Majority vote can be disabled by setting MVDIS in USARTn_CTRL.

If the value of the start bit is found to be high, the reception of the frame is aborted, filtering out false start bits possibly generated by noise on the input.

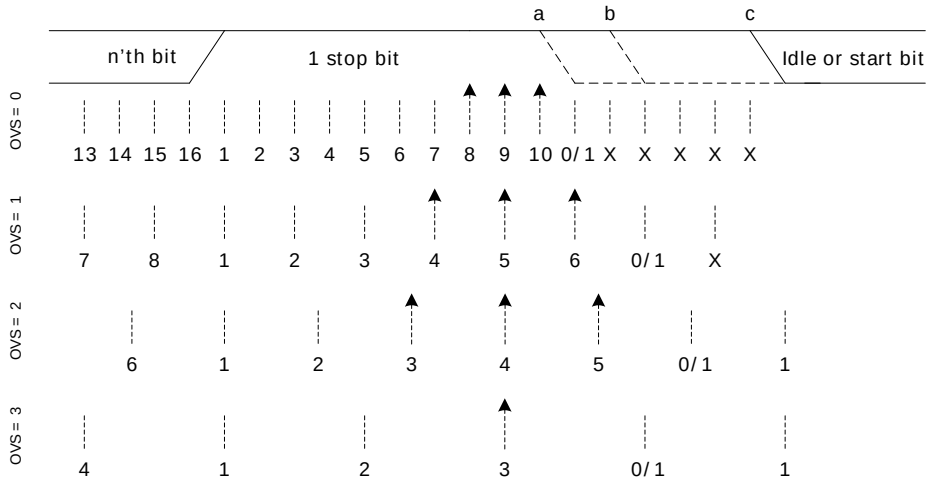
Figure 17.5. USART Sampling of Start and Data Bits



If the baud rate of the transmitter and receiver differ, the location each bit is sampled will be shifted towards the previous or next bit in the frame. This is acceptable for small errors in the baud rate, but for larger errors, it will result in transmission errors.

When the number of stop bits is 1 or more, stop bits are sampled like the start and data bits as seen in Figure 17.6 (p. 378) . When a stop bit has been detected by sampling at positions 8, 9 and 10 for normal mode, or 4, 5 and 6 for smart mode, the USART is ready for a new start bit. As seen in Figure 17.6 (p. 378) , a stop-bit of length 1 normally ends at c, but the next frame will be received correctly as long as the start-bit comes after position a for OVS=0 and OVS=3, and b for OVS=1 and OVS=2.

Figure 17.6. USART Sampling of Stop Bits when Number of Stop Bits are 1 or More



When working with stop bit lengths of half a baud period, the above sampling scheme no longer suffices. In this case, the stop-bit is not sampled, and no framing error is generated in the receiver if the stop-bit is not generated. The line must still be driven high before the next start bit however for the USART to successfully identify the start bit.

17.3.2.4.4 Parity Error

When parity bits are enabled, a parity check is automatically performed on incoming frames. When a parity error is detected in an incoming frame, the data parity error bit PERR in the frame is set, as well as the interrupt flag PERR in USARTn_IF. Frames with parity errors are loaded into the receive buffer like regular frames.

PERR can be accessed by reading the frame from the receive buffer using the USARTn_RXDATAX, USARTn_RXDATAXP, USARTn_RXDOUBLEX or USARTn_RXDOUBLEXP registers.

If ERRSTX in USARTn_CTRL is set, the transmitter is disabled on received parity and framing errors. If ERRSRX in USARTn_CTRL is set, the receiver is disabled on parity and framing errors.

17.3.2.4.5 Framing Error and Break Detection

A framing error is the result of an asynchronous frame where the stop bit was sampled to a value of 0. This can be the result of noise and baud rate errors, but can also be the result of a break generated by the transmitter on purpose.

When a framing error is detected in an incoming frame, the framing error bit FERR in the frame is set. The interrupt flag FERR in USARTn_IF is also set. Frames with framing errors are loaded into the receive buffer like regular frames.

FERR can be accessed by reading the frame from the receive buffer using the USARTn_RXDATAX, USARTn_RXDATAXP, USARTn_RXDOUBLEX or USARTn_RXDOUBLEXP registers.

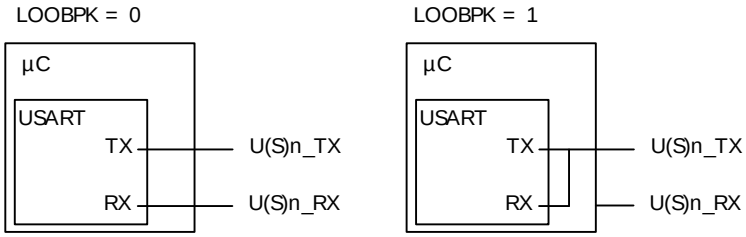
If ERRSTX in USARTn_CTRL is set, the transmitter is disabled on parity and framing errors. If ERRSRX in USARTn_CTRL is set, the receiver is disabled on parity and framing errors.

17.3.2.5 Local Loopback

The USART receiver samples U(S)n_RX by default, and the transmitter drives U(S)n_TX by default. This is not the only option however. When LOOPBK in USARTn_CTRL is set, the receiver is connected to the U(S)n_TX pin as shown in Figure 17.7 (p. 379) . This is useful for debugging, as the USART

can receive the data it transmits, but it is also used to allow the USART to read and write to the same pin, which is required for some half duplex communication modes. In this mode, the U(S)n_TX pin must be enabled as an output in the GPIO.

Figure 17.7. USART Local Loopback



17.3.2.6 Asynchronous Half Duplex Communication

When doing full duplex communication, two data links are provided, making it possible for data to be sent and received at the same time. In half duplex mode, data is only sent in one direction at a time. There are several possible half duplex setups, as described in the following sections.

17.3.2.6.1 Single Data-link

In this setup, the USART both receives and transmits data on the same pin. This is enabled by setting LOOPBK in USARTn_CTRL, which connects the receiver to the transmitter output. Because they are both connected to the same line, it is important that the USART transmitter does not drive the line when receiving data, as this would corrupt the data on the line.

When communicating over a single data-link, the transmitter must thus be tristated whenever not transmitting data. This is done by setting the command bit TXTRIEN in USARTn_CMD, which tristates the transmitter. Before transmitting data, the command bit TXTRIDIS, also in USARTn_CMD, must be set to enable transmitter output again. Whether or not the output is tristated at a given time can be read from TXTRI in USARTn_STATUS. If TXTRI is set when transmitting data, the data is shifted out of the shift register, but is not put out on U(S)n_TX.

When operating a half duplex data bus, it is common to have a bus master, which first transmits a request to one of the bus slaves, then receives a reply. In this case, the frame transmission control bits, which can be set by writing to USARTn_TXDATAX, can be used to make the USART automatically disable transmission, tristate the transmitter and enable reception when the request has been transmitted, making it ready to receive a response from the slave.

Tristating the transmitter can also be performed automatically by the USART by using AUTOTRI in USARTn_CTRL. When AUTOTRI is set, the USART automatically tristates U(S)n_TX whenever the transmitter is idle, and enables transmitter output when the transmitter goes active. If AUTOTRI is set TXTRI is always read as 0.

Note

Another way to tristate the transmitter is to enable wired-and or wired-or mode in GPIO. For wired-and mode, outputting a 1 will be the same as tristating the output, and for wired-or mode, outputting a 0 will be the same as tristating the output. This can only be done on buses with a pull-up or pull-down resistor respectively.

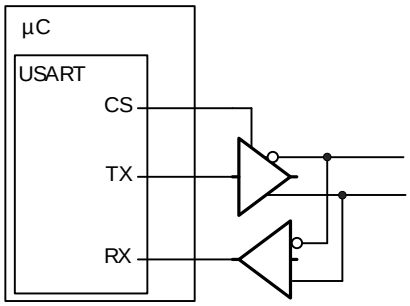
17.3.2.6.2 Single Data-link with External Driver

Some communication schemes, such as RS-485 rely on an external driver. Here, the driver has an extra input which enables it, and instead of tristating the transmitter when receiving data, the external driver must be disabled.

This can be done manually by assigning a GPIO to turn the driver on or off, or it can be handled automatically by the USART. If AUTOCS in USARTn_CTRL is set, the USn_CS output is automatically activated one baud period before the transmitter starts transmitting data, and deactivated when the last bit has been transmitted and there is no more data in the transmit buffer to transmit, or the transmitter becomes disabled. This feature can be used to turn the external driver on when transmitting data, and turn it off when the data has been transmitted.

Figure 17.8 (p. 380) shows an example configuration where USn_CS is used to automatically enable and disable an external driver.

Figure 17.8. USART Half Duplex Communication with External Driver



The USn_CS output is active low by default, but its polarity can be changed with CSINV in USARTn_CTRL. AUTOCS works regardless of which mode the USART is in, so this functionality can also be used for automatic chip/slave select when in synchronous mode (e.g. SPI).

17.3.2.6.3 Two Data-links

Some limited devices only support half duplex communication even though two data links are available. In this case software is responsible for making sure data is not transmitted when incoming data is expected.

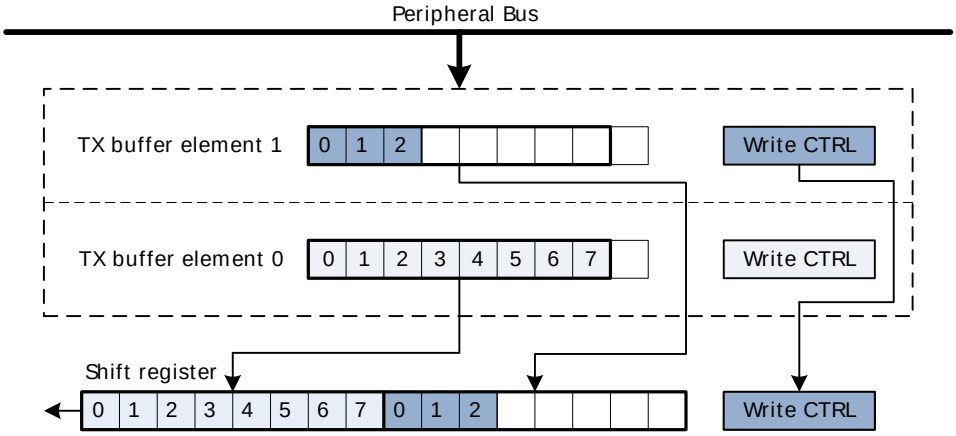
17.3.2.7 Large Frames

As each frame in the transmit and receive buffers holds a maximum of 9 bits, both the elements in the buffers are combined when working with USART-frames of 10 or more data bits.

To transmit such a frame, at least two elements must be available in the transmit buffer. If only one element is available, the USART will wait for the second element before transmitting the combined frame. Both the elements making up the frame are consumed when transmitting such a frame.

When using large frames, the 9th bits in the buffers are unused. For an 11 bit frame, the 8 least significant bits are thus taken from the first element in the buffer, and the 3 remaining bits are taken from the second element as shown in Figure 17.9 (p. 381) . The first element in the transmit buffer, i.e. element 0 in Figure 17.9 (p. 381) is the first element written to the FIFO, or the least significant byte when writing two bytes at a time using USARTn_TXDOUBLE.

Figure 17.9. USART Transmission of Large Frames



As shown in Figure 17.9 (p. 381) , frame transmission control bits are taken from the second element in FIFO.

The two buffer elements can be written at the same time using the USARTn_TXDOUBLE or USARTn_TXDOUBLEX register. The TXDATAX0 bitfield then refers to buffer element 0, and TXDATAX1 refers to buffer element 1.

Figure 17.10. USART Transmission of Large Frames, MSBF

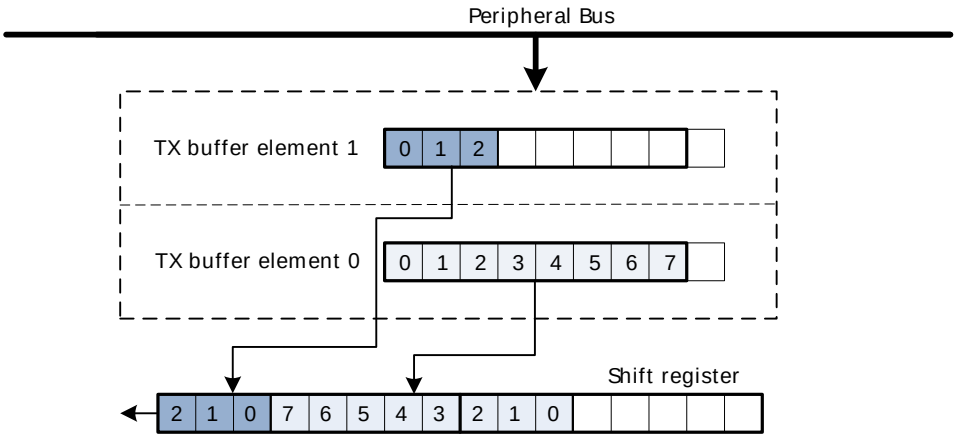
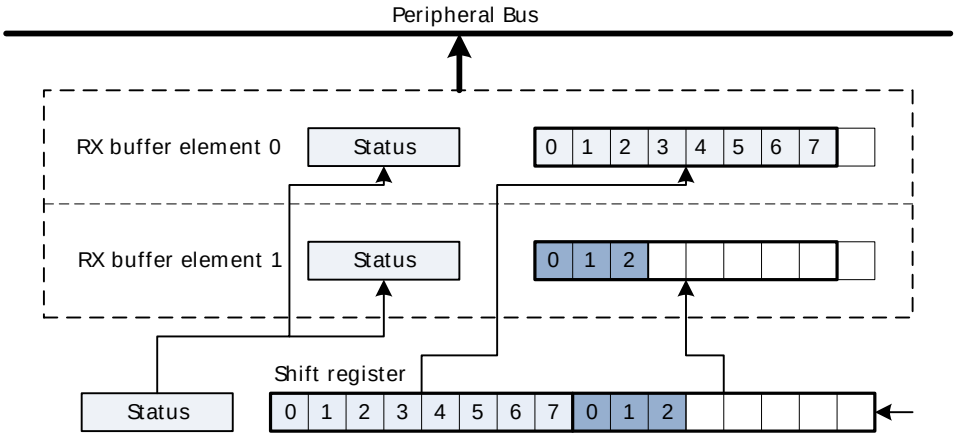


Figure 17.10 (p. 381) illustrates the order of the transmitted bits when an 11 bit frame is transmitted with MSBF set. If MSBF is set and the frame is smaller than 10 bits, only the contents of transmit buffer 0 will be transmitted.

When receiving a large frame, BYTESWAP in USARTn_CTRL determines the order the way the large frame is split into the two buffer elements. If BYTESWAP is cleared, the least significant 8 bits of the received frame are loaded into the first element of the receive buffer, and the remaining bits are loaded into the second element, as shown in Figure 17.11 (p. 382) . The first byte read from the buffer thus contains the 8 least significant bits. Set BYTESWAP to reverse the order.

The status bits are loaded into both elements of the receive buffer. The frame is not moved from the receive shift register before there are two free spaces in the receive buffer.

Figure 17.11. USART Reception of Large Frames



The two buffer elements can be read at the same time using the USARTn_RXDOUBLE or USARTn_RXDOUBLEX register. RXDATA0 then refers to buffer element 0 and RXDATA1 refers to buffer element 1.

Large frames can be used in both asynchronous and synchronous modes.

17.3.2.8 Multi-Processor Mode

To simplify communication between multiple processors, the USART supports a special multi-processor mode. In this mode the 9th data bit in each frame is used to indicate whether the content of the remaining 8 bits is data or an address.

When multi-processor mode is enabled, an incoming 9-bit frame with the 9th bit equal to the value of MPAB in USARTn_CTRL is identified as an address frame. When an address frame is detected, the MPAF interrupt flag in USARTn_IF is set, and the address frame is loaded into the receive register. This happens regardless of the value of RXBLOCK in USARTn_STATUS.

Multi-processor mode is enabled by setting MPM in USARTn_CTRL, and the value of the 9th bit in address frames can be set in MPAB. Note that the receiver must be enabled for address frames to be detected. The receiver can be blocked however, preventing data from being loaded into the receive buffer while looking for address frames.

Example 17.1 (p. 382) explains basic usage of the multi-processor mode:

Example 17.1. USART Multi-processor Mode Example

1. All slaves enable multi-processor mode and, enable and block the receiver. They will now not receive data unless it is an address frame. MPAB in USARTn_CTRL is set to identify frames with the 9th bit high as address frames.
2. The master sends a frame containing the address of a slave and with the 9th bit set.
3. All slaves receive the address frame and get an interrupt. They can read the address from the receive buffer. The selected slave unblocks the receiver to start receiving data from the master.
4. The master sends data with the 9th bit cleared.
5. Only the slave with RX enabled receives the data. When transmission is complete, the slave blocks the receiver and waits for a new address frame.

When a slave has received an address frame and wants to receive the following data, it must make sure the receiver is unblocked before the next frame has been completely received in order to prevent data loss.

BIT8DV in USARTn_CTRL can be used to specify the value of the 9th bit without writing to the transmit buffer with USARTn_TXDATAX or USARTn_TXDOUBLEX, giving higher efficiency in multi-processor mode, as the 9th bit is only set when writing address frames, and 8-bit writes to the USART can be used when writing the data frames.

17.3.2.9 Collision Detection

The USART supports a basic form of collision detection. When the receiver is connected to the output of the transmitter, either by using the LOOPBK bit in USARTn_CTRL or through an external connection, this feature can be used to detect whether data transmitted on the bus by the USART did get corrupted by a simultaneous transmission by another device on the bus.

For collision detection to be enabled, CCEN in USARTn_CTRL must be set, and the receiver enabled. The data sampled by the receiver is then continuously compared with the data output by the transmitter. If they differ, the CCF interrupt flag in USARTn_IF is set. The collision check includes all bits of the transmitted frames. The CCF interrupt flag is set once for each bit sampled by the receiver that differs from the bit output by the transmitter. When the transmitter output is disabled, i.e. the transmitter is tristated, collisions are not registered.

17.3.2.10 SmartCard Mode

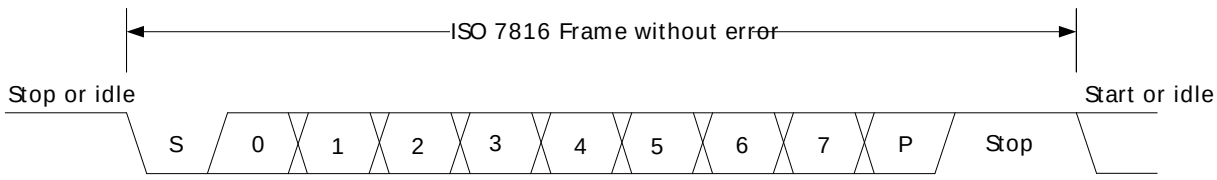
In SmartCard mode, the USART supports the ISO 7816 I/O line T0 mode. With exception of the stop-bits (guard time), the 7816 data frame is equal to the regular asynchronous frame. In this mode, the receiver pulls the line low for one baud, half a baud into the guard time to indicate a parity error. This NAK can for instance be used by the transmitter to re-transmit the frame. SmartCard mode is a half duplex asynchronous mode, so the transmitter must be tristated whenever not transmitting data.

To enable SmartCard mode, set SCMODE in USARTn_CTRL, set the number of databits in a frame to 8, and configure the number of stopbits to 1.5 by writing to STOPBITS in USARTn_FRAME.

The SmartCard mode relies on half duplex communication on a single line, so for it to work, both the receiver and transmitter must work on the same line. This can be achieved by setting LOOPBK in USARTn_CTRL or through an external connection. The TX output should be configured as open-drain in the GPIO module.

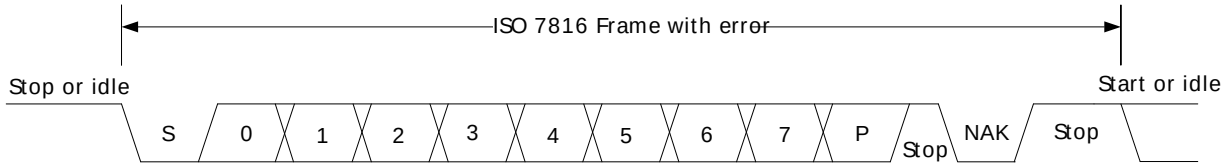
When no parity error is identified by the receiver, the data frame is as shown in Figure 17.12 (p. 383) . The frame consists of 8 data bits, a parity bit, and 2 stop bits. The transmitter does not drive the output line during the guard time.

Figure 17.12. USART ISO 7816 Data Frame Without Error



If a parity error is detected by the receiver, it pulls the line I/O line low after half a stop bit, see Figure 17.13 (p. 384) . It holds the line low for one bit-period before it releases the line. In this case, the guard time is extended by one bit period before a new transmission can start, resulting in a total of 3 stop bits.

Figure 17.13. USART ISO 7816 Data Frame With Error



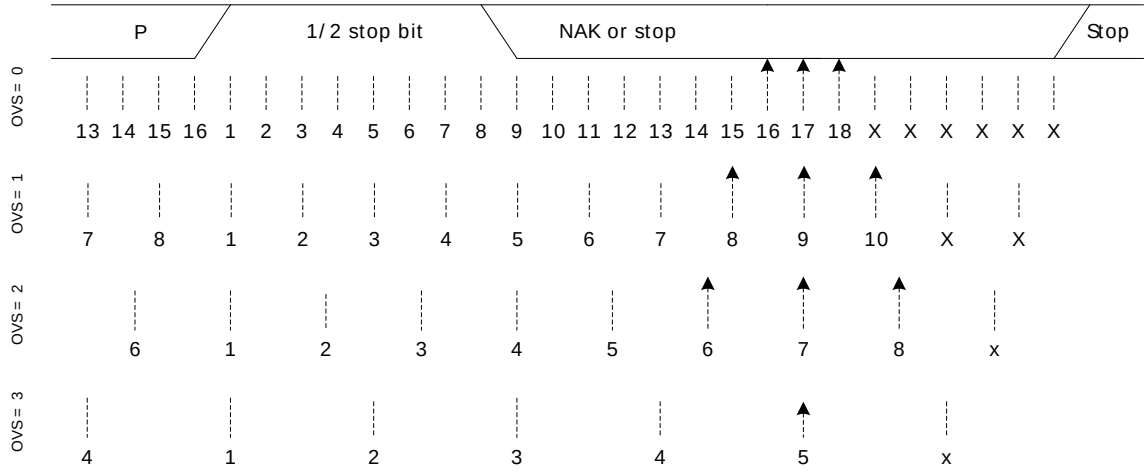
On a parity error, the NAK is generated by hardware. The NAK generated by the receiver is sampled as the stop-bit of the frame. Because of this, parity errors when in SmartCard mode are reported with both a parity error and a framing error.

When transmitting a T0 frame, the USART receiver on the transmitting side samples position 16, 17 and 18 in the stop-bit to detect the error signal when in 16x oversampling mode as shown in Figure 17.14 (p. 384) . Sampling at this location places the stop-bit sample in the middle of the bit-period used for the error signal (NAK).

If a NAK is transmitted by the receiver, it will thus appear as a framing error at the transmitter, and the FERR interrupt flag in USARTn_IF will be set. If SCRETRANS USARTn_CTRL is set, the transmitter will automatically retransmit a NACK'ed frame. The transmitter will retransmit the frame until it is ACK'ed by the receiver. This only works when the number of databits in a frame is configured to 8.

Set SKIPPERRF in USARTn_CTRL to make the receiver discard frames with parity errors. The PERR interrupt flag in USARTn_IF is set when a frame is discarded because of a parity error.

Figure 17.14. USART SmartCard Stop Bit Sampling



For communication with a SmartCard, a clock signal needs to be generated for the card. This clock output can be generated using one of the timers. See the ISO 7816 specification for more info on this clock signal.

SmartCard T1 mode is also supported. The T1 frame format used is the same as the asynchronous frame format with parity bit enabled and one stop bit. The USART must then be configured to operate in asynchronous half duplex mode.

17.3.3 Synchronous Operation

Most of the features in asynchronous mode are available in synchronous mode. Multi-processor mode can be enabled for 9-bit frames, loopback is available and collision detection can be performed.

17.3.3.1 Frame Format

The frames used in synchronous mode need no start and stop bits since a single clock is available to all parts participating in the communication. Parity bits cannot be used in synchronous mode.

The USART supports frame lengths of 4 to 16 bits per frame. Larger frames can be simulated by transmitting multiple smaller frames, i.e. a 22 bit frame can be sent using two 11-bit frames, and a 21 bit frame can be generated by transmitting three 7-bit frames. The number of bits in a frame is set using DATABITS in USARTn_FRAME.

The frames in synchronous mode are by default transmitted with the least significant bit first like in asynchronous mode. The bit-order can be reversed by setting MSBF in USARTn_CTRL.

The frame format used by the transmitter can be inverted by setting TXINV in USARTn_CTRL, and the format expected by the receiver can be inverted by setting RXINV, also in USARTn_CTRL.

17.3.3.2 Clock Generation

The bit-rate in synchronous mode is given by Equation 17.3 (p. 385) . As in the case of asynchronous operation, the clock division factor have a 13-bit integral part and a 2-bit fractional part.

USART Synchronous Mode Bit Rate

$$br = f_{HFPERCLK}/(2 \times (1 + USARTn\_CLKDIV/256)) \quad (17.3)$$

Given a desired baud rate brdesired, the clock divider USARTn_CLKDIV can be calculated using Equation 17.4 (p. 385)

USART Synchronous Mode Clock Division Factor

$$USARTn\_CLKDIV = 256 \times (f_{HFPERCLK}/(2 \times brdesired) - 1) \quad (17.4)$$

When the USART operates in master mode, the highest possible bit rate is half the peripheral clock rate. When operating in slave mode however, the highest bit rate is an eighth of the peripheral clock:

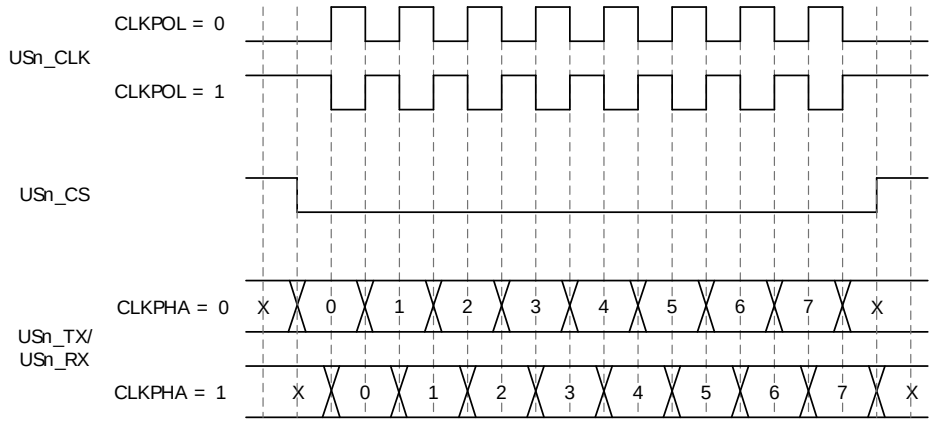
- Master mode: $br_{max} = f_{HFPERCLK}/2$
- Slave mode: $br_{max} = f_{HFPERCLK}/8$

On every clock edge data on the data lines, MOSI and MISO, is either set up or sampled. When CLKPHA in USARTn_CTRL is cleared, data is sampled on the leading clock edge and set-up is done on the trailing edge. If CLKPHA is set however, data is set-up on the leading clock edge, and sampled on the trailing edge. In addition to this, the polarity of the clock signal can be changed by setting CLKPOL in USARTn_CTRL, which also defines the idle state of the clock. This results in four different modes which are summarized in Table 17.8 (p. 385) . Figure 17.15 (p. 386) shows the resulting timing of data set-up and sampling relative to the bus clock.

Table 17.8. USART SPI Modes

SPI mode	CLKPOL	CLKPHA	Leading edge	Trailing edge
0	0	0	Rising, sample	Falling, set-up
1	0	1	Rising, set-up	Falling, sample
2	1	0	Falling, sample	Rising, set-up
3	1	1	Falling, set-up	Rising, sample

Figure 17.15. USART SPI Timing



If CPHA=1, the TX underflow flag, TXUF, will be set on the first setup clock edge of a frame in slave mode if TX data is not available. If CPHA=0, TXUF is set if data is not available in the transmit buffer three HFPERCLK cycles prior to the first sample clock edge. The RXDATAV flag is updated on the last sample clock edge of a transfer, while the RX overflow interrupt flag, RXOF, is set on the first sample clock edge if the receive buffer overflows. When a transfer has been performed, interrupt flags TXBL and TXC are updated on the first setup clock edge of the succeeding frame, or when CS is deasserted.

17.3.3.3 Master Mode

When in master mode, the USART is in full control of the data flow on the synchronous bus. When operating in full duplex mode, the slave cannot transmit data to the master without the master transmitting to the slave. The master outputs the bus clock on USn_CLK.

Communication starts whenever there is data in the transmit buffer and the transmitter is enabled. The USART clock then starts, and the master shifts bits out from the transmit shift register using the internal clock.

When there are no more frames in the transmit buffer and the transmit shift register is empty, the clock stops, and communication ends. When the receiver is enabled, it samples data using the internal clock when the transmitter transmits data. Operation of the RX and TX buffers is as in asynchronous mode.

17.3.3.3.1 Operation of USn_CS Pin

When operating in master mode, the USn_CS pin can have one of two functions, or it can be disabled.

If USn_CS is configured as an output, it can be used to automatically generate a chip select for a slave by setting AUTOCS in USARTn_CTRL. If AUTOCS is set, USn_CS is activated when a transmission begins, and deactivated directly after the last bit has been transmitted and there is no more data in the transmit buffer. By default, USn_CS is active low, but its polarity can be inverted by setting CSINV in USARTn_CTRL.

When USn_CS is configured as an input, it can be used by another master that wants control of the bus to make the USART release it. When USn_CS is driven low, or high if CSINV is set, the interrupt flag SSM in USARTn_IF is set, and if CSMA in USARTn_CTRL is set, the USART goes to slave mode.

17.3.3.3.2 AUTOTX

A synchronous master is required to transmit data to a slave in order to receive data from the slave. In some cases, only a few words are transmitted and a lot of data is then received from the slave. In that case, one solution is to keep feeding the TX with data to transmit, but that consumes system bandwidth. Instead AUTOTX can be used.

When AUTOTX in USARTn_CTRL is set, the USART transmits data as long as there is available space in the RX shift register for the chosen frame size. This happens even though there is no data in the TX buffer. The TX underflow interrupt flag TXUF in USARTn_IF is set on the first word that is transmitted which does not contain valid data.

During AUTOTX the USART will always send the previous sent bit, thus reducing the number of transitions on the TX output. So if the last bit sent was a 0, 0's will be sent during AUTOTX and if the last bit sent was a 1, 1's will be sent during AUTOTX.

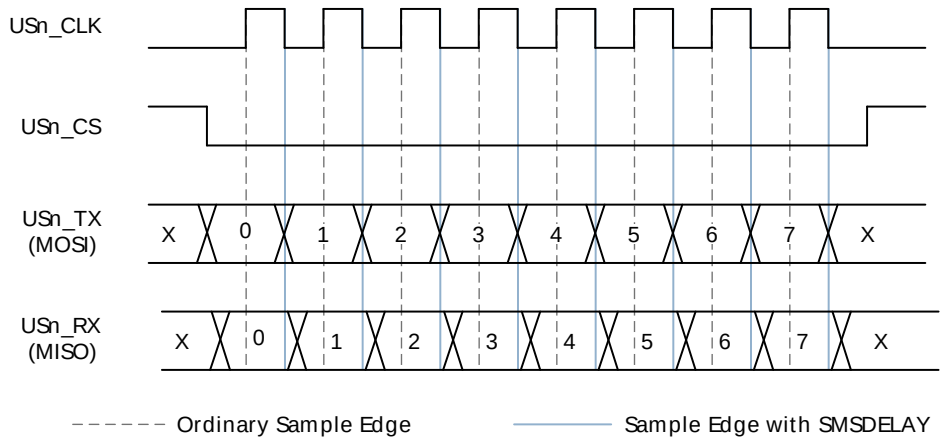
17.3.3.3.3 Synchronous Master Sample Delay

To improve speed in certain conditions by reducing the setup-time requirements for the SPI slave, the master can be configured to sample the data one half SCLK-cycle later, i.e. on the next setup edge, which, in SPI mode 0, is the rising edge. This is enabled by setting SMSDELAY in USARTn_CTRL and can be used together with all SPI slaves that does not set up new data before the next setup edge, as the propagation delay of SCLK will ensure sufficient hold time.

Note

If used together with another Silicon Laboratories chip utilizing SSSEARLY, a very thorough understanding of the timing is required.

Figure 17.16. USART SPI timing with SMSDELAY



17.3.3.4 Slave Mode

When the USART is in slave mode, data transmission is not controlled by the USART, but by an external master. The USART is therefore not able to initiate a transmission, and has no control over the number of bytes written to the master.

The output and input to the USART are also swapped when in slave mode, making the receiver take its input from USn_TX (MOSI) and the transmitter drive USn_RX (MISO).

To transmit data when in slave mode, the slave must load data into the transmit buffer and enable the transmitter. The data will remain in the USART until the master starts a transmission by pulling the USn_CS input of the slave low and transmitting data. For every frame the master transmits to the slave, a frame is transferred from the slave to the master. After a transmission, MISO remains in the same state as the last bit transmitted. This also applies if the master transmits to the slave and the slave TX buffer is empty.

If the transmitter is enabled in synchronous slave mode and the master starts transmission of a frame, the underflow interrupt flag TXUF in USARTn_IF will be set if no data is available for transmission to the master.

If the slave needs to control its own chip select signal, this can be achieved by clearing CSPEN in the ROUTE register. The internal chip select signal can then be controlled through CSINV in the CTRL register. The chip select signal will be CSINV inverted, i.e. if CSINV is cleared, the chip select is active and vice versa.

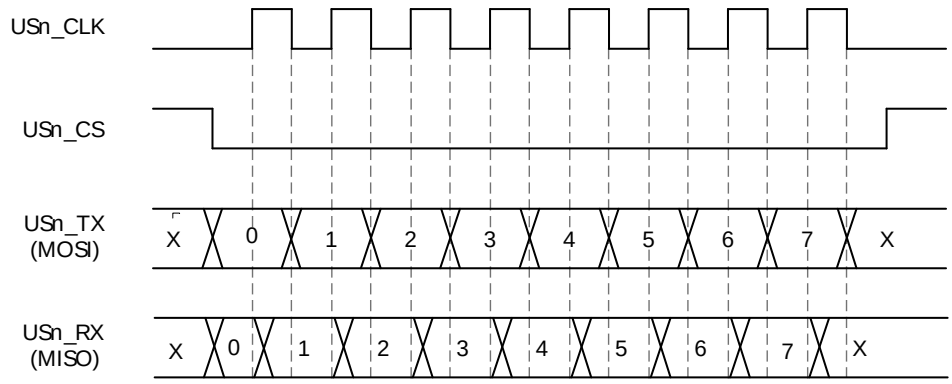
17.3.3.4.1 Synchronous Slave Setup Early

To improve speed in certain conditions by improving the setup time when running in slave mode, the slave can be configured to set up data one half SCLK-cycle earlier, i.e. on the previous sample edge, which, for SPI mode 0, is the falling edge. This is enabled by setting SSSEARLY in USARTn_CTRL and can be used with all SPI masters that samples the data on the sample edge, as the SCLK propagation delay will ensure sufficient hold time.

Note

If used together with another Silicon Laboratories chip utilizing SMSDELAY, a very thorough understanding of the timing is required.

Figure 17.17. USART SPI Slave Timing with SSSEARLY



17.3.3.5 Synchronous Half Duplex Communication

Half duplex communication in synchronous mode is very similar to half duplex communication in asynchronous mode as detailed in Section 17.3.2.6 (p. 379). The main difference is that in this mode, the master must generate the bus clock even when it is not transmitting data, i.e. it must provide the slave with a clock to receive data. To generate the bus clock, the master should transmit data with the transmitter tristated, i.e. TXTRI in USARTn_STATUS set, when receiving data. If 2 bytes are expected from the slave, then transmit 2 bytes with the transmitter tristated, and the slave uses the generated bus clock to transmit data to the master. TXTRI can be set by setting the TXTRIEN command bit in USARTn_CMD.

Note

When operating as SPI slave in half duplex mode, TX has to be tristated (not disabled) during data reception if the slave is to transmit data in the current transfer.

17.3.3.6 I2S

I2S is a synchronous format for transmission of audio data. The frame format is 32-bit, but since data is always transmitted with MSB first, an I2S device operating with 16-bit audio may choose to only process the 16 msb of the frame, and only transmit data in the 16 msb of the frame.

In addition to the bit clock used for regular synchronous transfers, I2S mode uses a separate word clock. When operating in mono mode, with only one channel of data, the word clock pulses once at the start of each new word. In stereo mode, the word clock toggles at the start of new words, and also gives away

whether the transmitted word is for the left or right audio channel; A word transmitted while the word clock is low is for the left channel, and a word transmitted while the word clock is high is for the right.

When operating in I2S mode, the CS pin is used as a the word clock. In master mode, this is automatically driven by the USART, and in slave mode, the word clock is expected from an external master.

17.3.3.6.1 Word Format

The general I2S word format is 32 bits wide, but the USART also supports 16-bit and 8-bit words. In addition to this, it can be specified how many bits of the word should actually be used by the USART. These parameters are given by FORMAT in USARTn_I2SCTRL.

As an example, configuring FORMAT to using a 32-bit word with 16-bit data will make each word on the I2S bus 32-bits wide, but when receiving data through the USART, only the 16 most significant bits of each word can be read out of the USART. Similarly, only the 16 most significant bits have to be written to the USART when transmitting. The rest of the bits will be transmitted as zeroes.

17.3.3.6.2 Major Modes

The USART supports a set of different I2S formats as shown in Table 17.9 (p. 389) , but it is not limited to these modes. MONO, JUSTIFY and DELAY in USARTn_I2SCTRL can be mixed and matched to create an appropriate format. MONO enables mono mode, i.e. one data stream instead of two which is the default. JUSTIFY aligns data within a word on the I2S bus, either left or right which can bee seen in figures Figure 17.20 (p. 390) and Figure 17.21 (p. 390). Finally, DELAY specifies whether a new I2S word should be started directly on the edge of the word-select signal, or one bit-period after the edge.

Table 17.9. USART I2S Modes

Mode	MONO	JUSTIFY	DELAY	CLKPOL
Regular I2S	0	0	1	0
Left-Justified	0	0	0	1
Right-Justified	0	1	0	1
Mono	1	0	0	0

The regular I2S waveform is shown in Figure 17.18 (p. 389) and Figure 17.19 (p. 390). The first figure shows a waveform transmitted with full accuracy. The wordlength can be configured to 32-bit, 16-bit or 8-bit using FORMAT in USARTn_I2SCTRL. In the second figure, I2S data is transmitted with reduced accuracy, i.e. the data transmitted has less bits than what is possible in the bus format.

Note that the msb of a word transmitted in regular I2S mode is delayed by one cycle with respect to word select

Figure 17.18. USART Standard I2S waveform

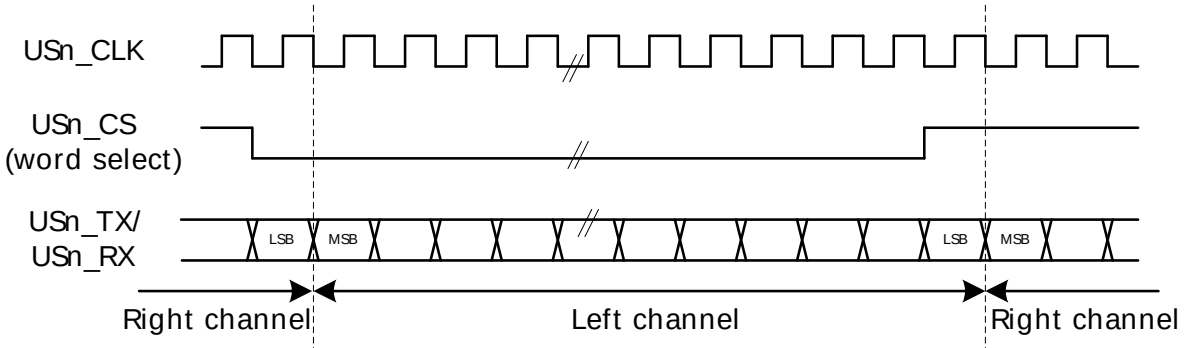
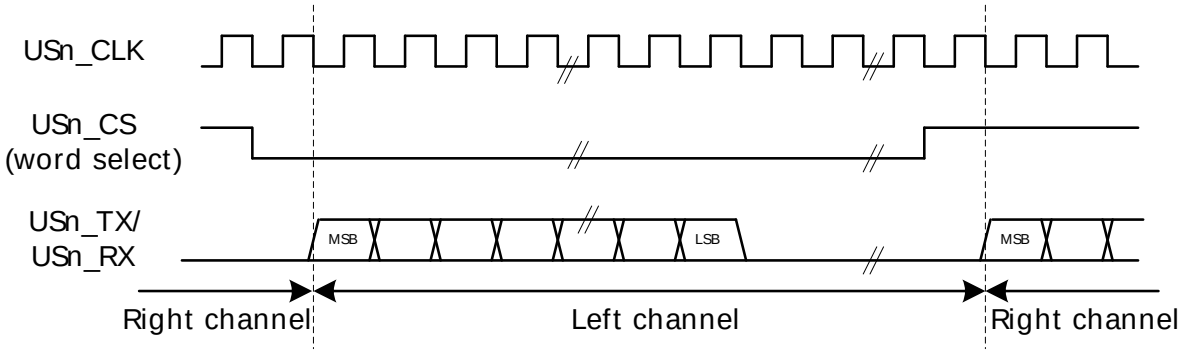
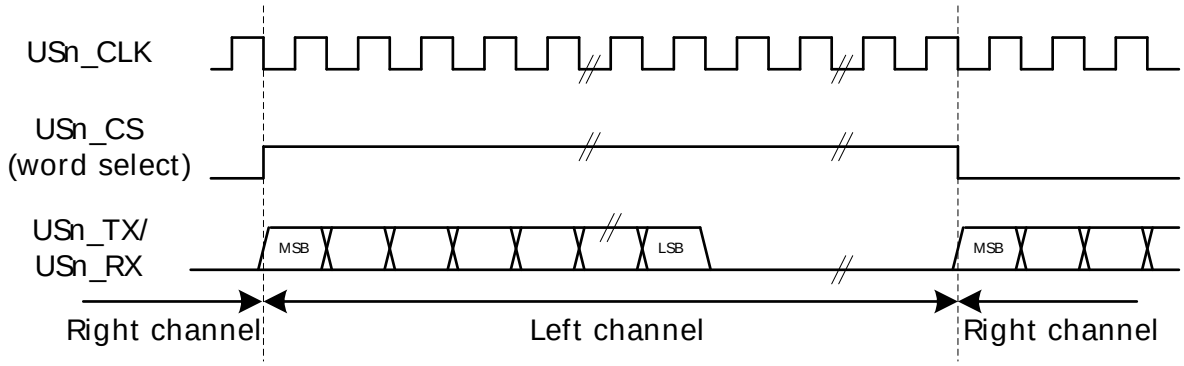


Figure 17.19. USART Standard I2S waveform (reduced accuracy)



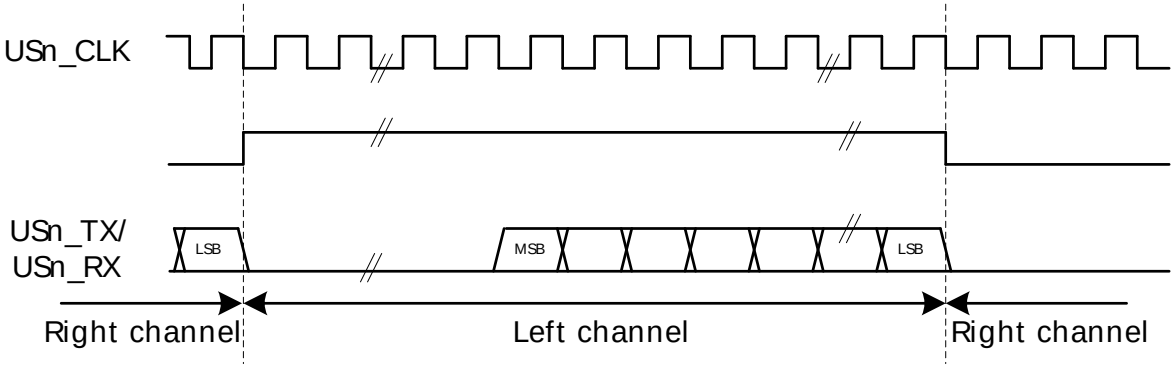
A left-justified stream is shown in Figure 17.20 (p. 390) . Note that the MSB comes directly after the edge on the word-select signal in contradiction to the regular I2S waveform where it comes one bit-period after.

Figure 17.20. USART Left-justified I2S waveform



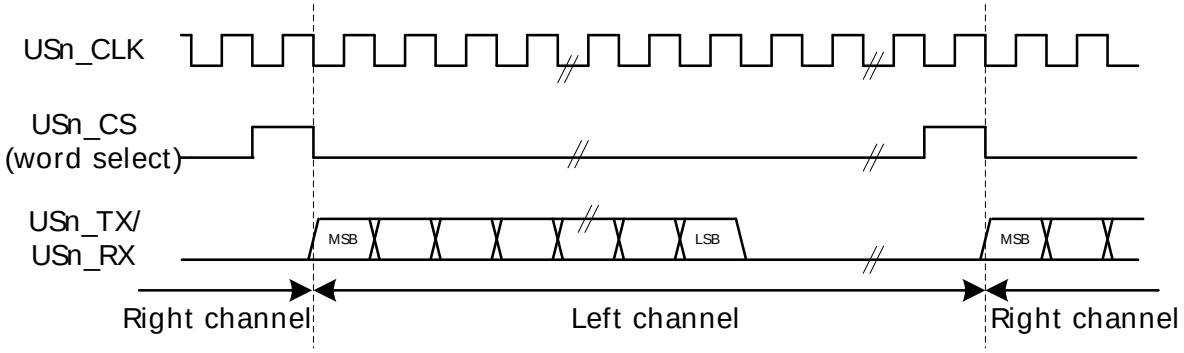
A right-justified stream is shown in Figure 17.21 (p. 390) . The left and right justified streams are equal when the data-size is equal to the word-width.

Figure 17.21. USART Right-justified I2S waveform



In mono-mode, the word-select signal pulses at the beginning of each word instead of toggling for each word. Mono I2S waveform is shown in Figure 17.22 (p. 391) .

Figure 17.22. USART Mono I2S waveform



17.3.3.6.3 Using I2S Mode

When using the USART in I2S mode, DATABITS in USARTn_FRAME must be set to 8 or 16 data-bits. 8 databits can be used in all modes, and 16 can be used in the modes where the number of bytes in the I2S word is even. In addition to this, MSBF in USARTn_CTRL should be set, and CLKPOL and CLKPHA in USARTn_CTRL should be cleared.

The USART does not have separate TX and RX buffers for left and right data, so when using I2S in stereo mode, the application must keep track of whether the buffers contain left or right data. This can be done by observing TXBLRIGHT, RXDATAVRIGHT and RXFULLRIGHT in USARTn_STATUS. TXBLRIGHT tells whether TX is expecting data for the left or right channel. It will be set with TXBL if right data is expected. The receiver will set RXDATAVRIGHT if there is at least one right element in the buffer, and RXFULLRIGHT if the buffer is full of right elements.

When using I2S with DMA, separate DMA requests can be used for left and right data by setting DMASPLIT in USARTn_I2SCTRL.

In both master and slave mode the USART always starts transmitting on the LEFT channel after being enabled. In master mode, the transmission will stop if TX becomes empty. In that case, TXC is set. Continuing the transmission in this case will make the data-stream continue where it left off. To make the USART start on the LEFT channel after going empty, disable and re-enable TX.

17.3.4 PRS-triggered Transmissions

If a transmission must be started on an event with very little delay, the PRS system can be used to trigger the transmission. The PRS channel to use as a trigger can be selected using TSEL in USARTn_TRIGCTRL. When a positive edge is detected on this signal, the receiver is enabled if RXTEN in USARTn_TRIGCTRL is set, and the transmitter is enabled if TXTEN in USARTn_TRIGCTRL is set. Only one signal input is supported by the USART.

The AUTOTX feature can also be enabled via PRS. If an external SPI device sets a pin high when there is data to be read from the device, this signal can be routed to the USART through the PRS system and be used to make the USART clock data out of the external device. If AUTOTXTEN in USARTn_TRIGCTRL is set, the USART will transmit data whenever the PRS signal selected by TSEL is high given that there is enough room in the RX buffer for the chosen frame size. Note that if there is no data in the TX buffer when using AUTOTX, the TX underflow interrupt will be set.

AUTOTXTEN can also be combined with TXTEN to make the USART transmit a command to the external device prior to clocking out data. To do this, disable TX using the TXDIS command, load the

TX buffer with the command and enable AUTOTXTEN and TXTEN. When the selected PRS input goes high, the USART will now transmit the loaded command, and then continue clocking out while both the PRS input is high and there is room in the RX buffer

17.3.5 PRS RX Input

The USART can be configured to receive data directly from a PRS channel by setting RXPRS in USARTn_INPUT. The PRS channel used is selected using RXPRSSEL in USARTn_INPUT. This way, for example, a differential RX signal can be input to the ACMP and the output routed via PRS to the USART.

17.3.6 DMA Support

The USART has full DMA support. The DMA controller can write to the transmit buffer using the registers USARTn_TXDATA, USARTn_TXDATAX, USARTn_TXDOUBLE and USARTn_TXDOUBLEX, and it can read from the receive buffer using the registers USARTn_RXDATA, USARTn_RXDATAX, USARTn_RXDOUBLE and USARTn_RXDOUBLEX. This enables single byte transfers, 9 bit data + control/status bits, double byte and double byte + control/status transfers both to and from the USART.

A request for the DMA controller to read from the USART receive buffer can come from the following source:

- Data available in the receive buffer.
- Data available in the receive buffer and data is for the RIGHT I2S channel. Only used in I2S mode.

A write request can come from one of the following sources:

- Transmit buffer and shift register empty. No data to send.
- Transmit buffer has room for more data.
- Transmit buffer has room for RIGHT I2S data. Only used in I2S mode.

Even though there are two sources for write requests to the DMA, only one should be used at a time, since the requests from both sources are cleared even though only one of the requests are used.

In some cases, it may be sensible to temporarily stop DMA access to the USART when an error such as a framing error has occurred. This is enabled by setting ERRSDMA in USARTn_CTRL.

17.3.7 Transmission Delay

By configuring TXDELAY in USARTn_CTRL, the transmitter can be forced to wait a number of bit-periods from it is ready to transmit data, to it actually transmits the data. This delay is only applied to the first frame transmitted after the transmitter has been idle. When transmitting frames back-to-back the delay is not introduced between the transmitted frames.

This is useful on half duplex buses, because the receiver always returns received frames to software during the first stop-bit. The bus may still be driven for up to 3 baud periods, depending on the current frame format. Using the transmission delay, a transmission can be started when a frame is received, and it is possible to make sure that the transmitter does not begin driving the output before the frame on the bus is completely transmitted.

TXDELAY in USARTn_CTRL only applies to asynchronous transmission.

17.3.8 Interrupts

The interrupts generated by the USART are combined into two interrupt vectors. Interrupts related to reception are assigned to one interrupt vector, and interrupts related to transmission are assigned to the other. Separating the interrupts in this way allows different priorities to be set for transmission and reception interrupts.

The transmission interrupt vector groups the transmission-related interrupts generated by the following interrupt flags:

- TXC
- TXBL
- TXOF
- CCF

The reception interrupt on the other hand groups the reception-related interrupts, triggered by the following interrupt flags:

- RXDATAV
- RXFULL
- RXOF
- RXUF
- PERR
- FERR
- MPAF
- SSM

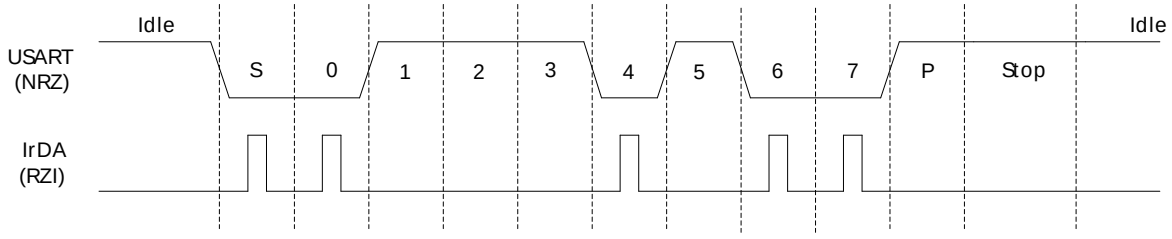
If USART interrupts are enabled, an interrupt will be made if one or more of the interrupt flags in USART_IF and their corresponding bits in USART_IEN are set.

17.3.9 IrDA Modulator/Demodulator

The IrDA modulator on USART0 implements the physical layer of the IrDA specification, which is necessary for communication over IrDA. The modulator takes the signal output from the USART module, and modulates it before it leaves USART0 . In the same way, the input signal is demodulated before it enters the actual USART module. The modulator is only available on USART0 , and implements the original Rev. 1.0 physical layer and one high speed extension which supports speeds from 2.4 kbps to 1.152 Mbps.

The data from and to the USART is represented in a NRZ (Non Return to Zero) format, where the signal value is at the same level through the entire bit period. For IrDA, the required format is RZI (Return to Zero Inverted), a format where a “1” is signalled by holding the line low, and a “0” is signalled by a short high pulse. An example is given in Figure 17.23 (p. 393) .

Figure 17.23. USART Example RZI Signal for a given Asynchronous USART Frame



The IrDA module is enabled by setting IREN. The USART transmitter output and receiver input is then routed through the IrDA modulator.

The width of the pulses generated by the IrDA modulator is set by configuring IRPW in USARTn_IRCTRL. Four pulse widths are available, each defined relative to the configured bit period as listed in Table 17.10 (p. 394) .

Table 17.10. USART IrDA Pulse Widths

IRPW	Pulse width OVS=0	Pulse width OVS=1	Pulse width OVS=2	Pulse width OVS=3
00	1/16	1/8	1/6	1/4
01	2/16	2/8	2/6	N/A
10	3/16	3/8	N/A	N/A
11	4/16	N/A	N/A	N/A

By default, no filter is enabled in the IrDA demodulator. A filter can be enabled by setting IRFILT in USARTn_IRCTRL. When the filter is enabled, an incoming pulse has to last for 4 consecutive clock cycles to be detected by the IrDA demodulator.

Note that by default, the idle value of the USART data signal is high. This means that the IrDA modulator generates negative pulses, and the IrDA demodulator expects negative pulses. To make the IrDA module use RZI signalling, both TXINV and RXINV in USARTn_CTRL must be set.

The IrDA module can also modulate a signal from the PRS system, and transmit a modulated signal to the PRS system. To use a PRS channel as transmitter source instead of the USART, set IRPRSEN in USARTn_IRCTRL high. The channel is selected by configuring IRPRSSEL in USARTn_IRCTRL.

17.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	USARTn_CTRL	RW	Control Register
0x004	USARTn_FRAME	RW	USART Frame Format Register
0x008	USARTn_TRIGCTRL	RW	USART Trigger Control register
0x00C	USARTn_CMD	W1	Command Register
0x010	USARTn_STATUS	R	USART Status Register
0x014	USARTn_CLKDIV	RW	Clock Control Register
0x018	USARTn_RXDATAX	R	RX Buffer Data Extended Register
0x01C	USARTn_RXDATA	R	RX Buffer Data Register
0x020	USARTn_RXDOUBLEX	R	RX Buffer Double Data Extended Register
0x024	USARTn_RXDOUBLE	R	RX FIFO Double Data Register
0x028	USARTn_RXDATAXP	R	RX Buffer Data Extended Peek Register
0x02C	USARTn_RXDOUBLEXP	R	RX Buffer Double Data Extended Peek Register
0x030	USARTn_TXDATAX	W	TX Buffer Data Extended Register
0x034	USARTn_TXDATA	W	TX Buffer Data Register
0x038	USARTn_TXDOUBLEX	W	TX Buffer Double Data Extended Register
0x03C	USARTn_TXDOUBLE	W	TX Buffer Double Data Register
0x040	USARTn_IF	R	Interrupt Flag Register
0x044	USARTn_IFS	W1	Interrupt Flag Set Register
0x048	USARTn_IFC	W1	Interrupt Flag Clear Register
0x04C	USARTn_IEN	RW	Interrupt Enable Register
0x050	USARTn_IRCTRL	RW	IrDA Control Register
0x054	USARTn_ROUTE	RW	I/O Routing Register
0x058	USARTn_INPUT	RW	USART Input Register
0x05C	USARTn_I2SCTRL	RW	I2S Control Register

17.5 Register Description

17.5.1 USARTn_CTRL - Control Register

Offset	Bit Position																															
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0x0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0x0		0	0	0	0	0
Access	RW	RW	RW	RW	RW		RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW		RW		RW	RW	RW	RW	RW
Name	SMSDELAY	MVDIS	AUTOTX	BYTESWAP	TXDELAY		SSSEARLY	ERRSTX	ERRSRX	ERRSDMA	BIT8DV	SKIPPERRF	SCRETRANS	SCMODE	AUTOTRI	AUTOCS	CSINV	TXINV	RXINV	TXBIL	CSMA	MSBF	CLKPHA	CLKPOL		OVS		MPAB	MPM	CCEN	LOOPBK	SYNC

Bit	Name	Reset	Access	Description
31	SMSDELAY	0	RW	**Synchronous Master Sample Delay**
	Delay Synchronous Master sample point to the next setup edge to improve timing and allow communication at higher speeds.			
30	MVDIS	0	RW	**Majority Vote Disable**
	Disable majority vote for 16x, 8x and 6x oversampling modes.			

Bit	Name	Reset	Access	Description
29	AUTOTX	0	RW	**Always Transmit When RX Not Full**

Transmits as long as RX is not full. If TX is empty, underflows are generated.

28	BYTESWAP	0	RW	**Byteswap In Double Accesses**

Set to switch the order of the bytes in double accesses.

Value	Description
0	Normal byte order
1	Byte order swapped

27:26	TXDELAY	0x0	RW	**TX Delay Transmission**

Configurable delay before new transfers. Frames sent back-to-back are not delayed.

Value	Mode	Description
0	NONE	Frames are transmitted immediately
1	SINGLE	Transmission of new frames are delayed by a single baud period
2	DOUBLE	Transmission of new frames are delayed by two baud periods
3	TRIPLE	Transmission of new frames are delayed by three baud periods

25	SSSEARLY	0	RW	**Synchronous Slave Setup Early**

Setup data on sample edge in synchronous slave mode to improve MOSI setup time.

24	ERRSTX	0	RW	**Disable TX On Error**

When set, the transmitter is disabled on framing and parity errors (asynchronous mode only) in the receiver.

Value	Description
0	Received framing and parity errors have no effect on transmitter
1	Received framing and parity errors disable the transmitter

23	ERRSRX	0	RW	**Disable RX On Error**

When set, the receiver is disabled on framing and parity errors (asynchronous mode only).

Value	Description
0	Framing and parity errors have no effect on receiver
1	Framing and parity errors disable the receiver

22	ERRSDMA	0	RW	**Halt DMA On Error**

When set, DMA requests will be cleared on framing and parity errors (asynchronous mode only).

Value	Description
0	Framing and parity errors have no effect on DMA requests from the USART
1	DMA requests from the USART are blocked while the PERR or FERR interrupt flags are set

21	BIT8DV	0	RW	**Bit 8 Default Value**

The default value of the 9th bit. If 9-bit frames are used, and an 8-bit write operation is done, leaving the 9th bit unspecified, the 9th bit is set to the value of BIT8DV.

20	SKIPPERRF	0	RW	**Skip Parity Error Frames**

When set, the receiver discards frames with parity errors (asynchronous mode only). The PERR interrupt flag is still set.

19	SCRETRANS	0	RW	**SmartCard Retransmit**

When in SmartCard mode, a NACK'ed frame will be kept in the shift register and retransmitted if the transmitter is still enabled.

18	SCMODE	0	RW	**SmartCard Mode**

Use this bit to enable or disable SmartCard mode.

17	AUTOTRI	0	RW	**Automatic TX Tristate**

When enabled, TXTRI is set by hardware whenever the transmitter is idle, and TXTRI is cleared by hardware when transmission starts.

Value	Description
0	The output on U(S)n_TX when the transmitter is idle is defined by TXINV
1	U(S)n_TX is tristated whenever the transmitter is idle

16	AUTOCS	0	RW	**Automatic Chip Select**

Bit	Name	Reset	Access	Description
	When enabled, the output on USn_CS will be activated one baud-period before transmission starts, and deactivated when transmission ends.			
15	CSINV	0	RW	**Chip Select Invert**

Default value is active low. This affects both the selection of external slaves, as well as the selection of the microcontroller as a slave.

Value	Description
0	Chip select is active low
1	Chip select is active high

14	TXINV	0	RW	**Transmitter output Invert**

The output from the USART transmitter can optionally be inverted by setting this bit.

Value	Description
0	Output from the transmitter is passed unchanged to U(S)n_TX
1	Output from the transmitter is inverted before it is passed to U(S)n_TX

13	RXINV	0	RW	**Receiver Input Invert**

Setting this bit will invert the input to the USART receiver.

Value	Description
0	Input is passed directly to the receiver
1	Input is inverted before it is passed to the receiver

12	TXBIL	0	RW	**TX Buffer Interrupt Level**

Determines the interrupt and status level of the transmit buffer.

Value	Mode	Description
0	EMPTY	TXBL and the TXBL interrupt flag are set when the transmit buffer becomes empty. TXBL is cleared when the buffer becomes nonempty.
1	HALFFULL	TXBL and TXBLIF are set when the transmit buffer goes from full to half-full or empty. TXBL is cleared when the buffer becomes full.

11	CSMA	0	RW	**Action On Slave-Select In Master Mode**

This register determines the action to be performed when slave-select is configured as an input and driven low while in master mode.

Value	Mode	Description
0	NOACTION	No action taken
1	GOTOSLAVEMODE	Go to slave mode

10	MSBF	0	RW	**Most Significant Bit First**

Decides whether data is sent with the least significant bit first, or the most significant bit first.

Value	Description
0	Data is sent with the least significant bit first
1	Data is sent with the most significant bit first

9	CLKPHA	0	RW	**Clock Edge For Setup/Sample**

Determines where data is set-up and sampled according to the bus clock when in synchronous mode.

Value	Mode	Description
0	SAMPLELEADING	Data is sampled on the leading edge and set-up on the trailing edge of the bus clock in synchronous mode
1	SAMPLETRAILING	Data is set-up on the leading edge and sampled on the trailing edge of the bus clock in synchronous mode

8	CLKPOL	0	RW	**Clock Polarity**

Determines the clock polarity of the bus clock used in synchronous mode.

Value	Mode	Description
0	IDLELOW	The bus clock used in synchronous mode has a low base value
1	IDLEHIGH	The bus clock used in synchronous mode has a high base value

7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:5	OVS	0x0	RW	**Oversampling**

Bit	Name	Reset	Access	Description

Sets the number of clock periods in a UART bit-period. More clock cycles gives better robustness, while less clock cycles gives better performance.

Value	Mode	Description
0	X16	Regular UART mode with 16X oversampling in asynchronous mode
1	X8	Double speed with 8X oversampling in asynchronous mode
2	X6	6X oversampling in asynchronous mode
3	X4	Quadruple speed with 4X oversampling in asynchronous mode

4 MPAB 0 RW **Multi-Processor Address-Bit**

Defines the value of the multi-processor address bit. An incoming frame with its 9th bit equal to the value of this bit marks the frame as a multi-processor address frame.

3 MPM 0 RW **Multi-Processor Mode**

Multi-processor mode uses the 9th bit of the USART frames to tell whether the frame is an address frame or a data frame.

Value	Description
0	The 9th bit of incoming frames has no special function
1	An incoming frame with the 9th bit equal to MPAB will be loaded into the receive buffer regardless of RXBLOCK and will result in the MPAB interrupt flag being set

2 CCEN 0 RW **Collision Check Enable**

Enables collision checking on data when operating in half duplex modus.

Value	Description
0	Collision check is disabled
1	Collision check is enabled. The receiver must be enabled for the check to be performed

1 LOOPBK 0 RW **Loopback Enable**

Allows the receiver to be connected directly to the USART transmitter for loopback and half duplex communication.

Value	Description
0	The receiver is connected to and receives data from U(S)n_RX
1	The receiver is connected to and receives data from U(S)n_TX

0 SYNC 0 RW **USART Synchronous Mode**

Determines whether the USART is operating in asynchronous or synchronous mode.

Value	Description
0	The USART operates in asynchronous mode
1	The USART operates in synchronous mode

17.5.2 USARTn_FRAME - USART Frame Format Register

Offset	Bit Position																													
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																			0x1				0x0						0x5			
Access																			RW				RW						RW			
Name																			STOPBITS				PARITY						DATABITS			

Bit	Name	Reset	Access	Description
31:14	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

13:12 STOPBITS 0x1 RW **Stop-Bit Mode**

Determines the number of stop-bits used.

Value	Mode	Description
0	HALF	The transmitter generates a half stop bit. Stop-bits are not verified by receiver

Bit	Name	Reset	Access	Description

Value	Mode	Description
1	ONE	One stop bit is generated and verified
2	ONEANDAHALF	The transmitter generates one and a half stop bit. The receiver verifies the first stop bit
3	TWO	The transmitter generates two stop bits. The receiver checks the first stop-bit only

11:10 *Reserved* *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

9:8 PARITY 0x0 RW **Parity-Bit Mode**

Determines whether parity bits are enabled, and whether even or odd parity should be used. Only available in asynchronous mode.

Value	Mode	Description
0	NONE	Parity bits are not used
2	EVEN	Even parity are used. Parity bits are automatically generated and checked by hardware.
3	ODD	Odd parity is used. Parity bits are automatically generated and checked by hardware.

7:4 *Reserved* *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

3:0 DATABITS 0x5 RW **Data-Bit Mode**

This register sets the number of data bits in a USART frame.

Value	Mode	Description
1	FOUR	Each frame contains 4 data bits
2	FIVE	Each frame contains 5 data bits
3	SIX	Each frame contains 6 data bits
4	SEVEN	Each frame contains 7 data bits
5	EIGHT	Each frame contains 8 data bits
6	NINE	Each frame contains 9 data bits
7	TEN	Each frame contains 10 data bits
8	ELEVEN	Each frame contains 11 data bits
9	TWELVE	Each frame contains 12 data bits
10	THIRTEEN	Each frame contains 13 data bits
11	FOURTEEN	Each frame contains 14 data bits
12	FIFTEEN	Each frame contains 15 data bits
13	SIXTEEN	Each frame contains 16 data bits

17.5.3 USARTn_TRIGCTRL - USART Trigger Control register

Offset	Bit Position																														
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																										0	0	0			0x0	
Access																										RW	RW	RW			RW	
Name																										AUTOTXTEN	TXTEN	RXTEN			TSEL	

Bit	Name	Reset	Access	Description
31:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	AUTOTXTEN	0	RW	**AUTOTX Trigger Enable**
	When set, AUTOTX is enabled as long as the PRS channel selected by TSEL has a high value.			
5	TXTEN	0	RW	**Transmit Trigger Enable**
	When set, the PRS channel selected by TSEL sets TXEN, enabling the transmitter on positive trigger edges.			
4	RXTEN	0	RW	**Receive Trigger Enable**
	When set, the PRS channel selected by TSEL sets RXEN, enabling the receiver on positive trigger edges.			

Bit	Name	Reset	Access	Description
3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2:0	TSEL	0x0	RW	**Trigger PRS Channel Select**

Select USART PRS trigger channel. The PRS signal can enable RX and/or TX, depending on the setting of RXTEN and TXTEN.

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected
1	PRSCH1	PRS Channel 1 selected
2	PRSCH2	PRS Channel 2 selected
3	PRSCH3	PRS Channel 3 selected
4	PRSCH4	PRS Channel 4 selected
5	PRSCH5	PRS Channel 5 selected

17.5.4 USARTn_CMD - Command Register

Offset	Bit Position																											
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																					0	0	0	0	0	0	0	0	0	0	0	0
Access																					W1	W1	W1	W1	W1	W1	W1	W1	W1	W1	W1	W1
Name																					CLEARRX	CLEARTX	TXTRIDIS	TXTRIEN	RXBLOCKDIS	RXBLOCKEN	MASTERDIS	MASTEREN	TXDIS	TXEN	RXDIS	RXEN

Bit	Name	Reset	Access	Description
31:12	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
11	CLEARRX	0	W1	**Clear RX**
	Set to clear receive buffer and the RX shift register.			
10	CLEARTX	0	W1	**Clear TX**
	Set to clear transmit buffer and the TX shift register.			
9	TXTRIDIS	0	W1	**Transmitter Tristate Disable**
	Disables tristating of the transmitter output.			
8	TXTRIEN	0	W1	**Transmitter Tristate Enable**
	Tristates the transmitter output.			
7	RXBLOCKDIS	0	W1	**Receiver Block Disable**
	Set to clear RXBLOCK, resulting in all incoming frames being loaded into the receive buffer.			
6	RXBLOCKEN	0	W1	**Receiver Block Enable**
	Set to set RXBLOCK, resulting in all incoming frames being discarded.			
5	MASTERDIS	0	W1	**Master Disable**
	Set to disable master mode, clearing the MASTER status bit and putting the USART in slave mode.			
4	MASTEREN	0	W1	**Master Enable**
	Set to enable master mode, setting the MASTER status bit. Master mode should not be enabled while TXENS is set to 1. To enable both master and TX mode, write MASTEREN before TXEN, or enable them both in the same write operation.			
3	TXDIS	0	W1	**Transmitter Disable**
	Set to disable transmission.			
2	TXEN	0	W1	**Transmitter Enable**
	Set to enable data transmission.			
1	RXDIS	0	W1	**Receiver Disable**

Bit	Name	Reset	Access	Description
	Set to disable data reception. If a frame is under reception when the receiver is disabled, the incoming frame is discarded.			
0	RXEN	0	W1	**Receiver Enable**
	Set to activate data reception on U(S)n_RX.			

17.5.5 USARTn_STATUS - USART Status Register

Offset	Bit Position																													
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																				0	0	0	0	0	0	1	0	0	0	0	0	0
Access																				R	R	R	R	R	R	R	R	R	R	R	R	R
Name																				RXFULLRIGHT	RXDATAVRIGHT	TXBSRIGHT	TXBDRIGHT	RXFULL	RXDATAV	TXBL	TXC	TXTRI	RXBLOCK	MASTER	TXENS	RXENS

Bit	Name	Reset	Access	Description
31:13	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
12	RXFULLRIGHT	0	R	**RX Full of Right Data**
	When set, the entire RX buffer contains right data. Only used in I2S mode.			
11	RXDATAVRIGHT	0	R	**RX Data Right**
	When set, reading RXDATA or RXDATAX gives right data. Else left data is read. Only used in I2S mode.			
10	TXBSRIGHT	0	R	**TX Buffer Expects Single Right Data**
	When set, the TX buffer expects at least a single right data. Else it expects left data. Only used in I2S mode.			
9	TXBDRIGHT	0	R	**TX Buffer Expects Double Right Data**
	When set, the TX buffer expects double right data. Else it may expect a single right data or left data. Only used in I2S mode.			
8	RXFULL	0	R	**RX FIFO Full**
	Set when the RXFIFO is full. Cleared when the receive buffer is no longer full. When this bit is set, there is still room for one more frame in the receive shift register.			
7	RXDATAV	0	R	**RX Data Valid**
	Set when data is available in the receive buffer. Cleared when the receive buffer is empty.			
6	TXBL	1	R	**TX Buffer Level**
	Indicates the level of the transmit buffer. If TXBIL is cleared, TXBL is set whenever the transmit buffer is empty, and if TXBIL is set, TXBL is set whenever the transmit buffer is half-full or empty.			
5	TXC	0	R	**TX Complete**
	Set when a transmission has completed and no more data is available in the transmit buffer and shift register. Cleared when data is written to the transmit buffer.			
4	TXTRI	0	R	**Transmitter Tristated**
	Set when the transmitter is tristated, and cleared when transmitter output is enabled. If AUTOTRI in USARTn_CTRL is set this bit is always read as 0.			
3	RXBLOCK	0	R	**Block Incoming Data**
	When set, the receiver discards incoming frames. An incoming frame will not be loaded into the receive buffer if this bit is set at the instant the frame has been completely received.			
2	MASTER	0	R	**SPI Master Mode**
	Set when the USART operates as a master. Set using the MASTEREN command and clear using the MASTERDIS command.			
1	TXENS	0	R	**Transmitter Enable Status**
	Set when the transmitter is enabled.			

Bit	Name	Reset	Access	Description
0	RXENS	0	R	**Receiver Enable Status**
	Set when the receiver is enabled.			

17.5.6 USARTn_CLKDIV - Clock Control Register

Offset	Bit Position																																	
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Reset												0x0000															0x0							
Access												RW															RW							
Name												DIV															DIVEXT							

Bit	Name	Reset	Access	Description
31:21	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
20:6	DIV	0x0000	RW	**Fractional Clock Divider**
	Specifies the fractional clock divider for the USART.			
5:3	DIVEXT	0x0	RW	**Extended Fractional Clock Divider**
	Specifies the extended fractional clock divider for the USART.			
2:0	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

17.5.7 USARTn_RXDATAX - RX Buffer Data Extended Register

Offset	Bit Position																																	
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Reset																	0	0						0x000										
Access																	R	R						R										
Name																	FERR	PERR						RXDATA										

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15	FERR	0	R	**Data Framing Error**
	Set if data in buffer has a framing error. Can be the result of a break condition.			
14	PERR	0	R	**Data Parity Error**
	Set if data in buffer has a parity error (asynchronous mode only).			
13:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:0	RXDATA	0x000	R	**RX Data**
	Use this register to access data read from the USART. Buffer is cleared on read access.			

17.5.8 USARTn_RXDATA - RX Buffer Data Register

Offset	Bit Position																															
0x01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0x00							
Access																									R							
Name																									RXDATA							

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:0	RXDATA	0x00	R	**RX Data**
	Use this register to access data read from USART. Buffer is cleared on read access. Only the 8 LSB can be read using this register.			

17.5.9 USARTn_RXDOUBLEX - RX Buffer Double Data Extended Register

Offset	Bit Position																															
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0						0x000									0	0						0x000								
Access	R	R						R									R	R						R								
Name	FERR1	PERR1						RXDATA1									FERR0	PERR0						RXDATA0								

Bit	Name	Reset	Access	Description
31	FERR1	0	R	**Data Framing Error 1**
	Set if data in buffer has a framing error. Can be the result of a break condition.			
30	PERR1	0	R	**Data Parity Error 1**
	Set if data in buffer has a parity error (asynchronous mode only).			
29:25	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
24:16	RXDATA1	0x000	R	**RX Data 1**
	Second frame read from buffer.			
15	FERR0	0	R	**Data Framing Error 0**
	Set if data in buffer has a framing error. Can be the result of a break condition.			
14	PERR0	0	R	**Data Parity Error 0**
	Set if data in buffer has a parity error (asynchronous mode only).			
13:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:0	RXDATA0	0x000	R	**RX Data 0**
	First frame read from buffer.			

17.5.10 USARTn_RXDOUBLE - RX FIFO Double Data Register

Offset	Bit Position																													
0x024	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x00								0x00							
Access																	R								R							
Name																	RXDATA1								RXDATA0							

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:8	RXDATA1	0x00	R	**RX Data 1**
	Second frame read from buffer.			
7:0	RXDATA0	0x00	R	**RX Data 0**
	First frame read from buffer.			

17.5.11 USARTn_RXDATAXP - RX Buffer Data Extended Peek Register

Offset	Bit Position																													
0x028	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0	0						0x000								
Access																	R	R						R								
Name																	FERRP	PERRP						RXDATAP								

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15	FERRP	0	R	**Data Framing Error Peek**
	Set if data in buffer has a framing error. Can be the result of a break condition.			
14	PERRP	0	R	**Data Parity Error Peek**
	Set if data in buffer has a parity error (asynchronous mode only).			
13:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:0	RXDATAP	0x000	R	**RX Data Peek**
	Use this register to access data read from the USART.			

17.5.12 USARTn_RXDOUBLEXP - RX Buffer Double Data Extended Peek Register

Offset	Bit Position																													
0x02C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0						0x000									0	0						0x000								
Access	R	R						R									R	R						R								
Name	FERRP1	PERRP1						RXDATAP1									FERRP0	PERRP0						RXDATAP0								

Bit	Name	Reset	Access	Description
31	FERRP1	0	R	**Data Framing Error 1 Peek**
	Set if data in buffer has a framing error. Can be the result of a break condition.			
30	PERRP1	0	R	**Data Parity Error 1 Peek**
	Set if data in buffer has a parity error (asynchronous mode only).			
29:25	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
24:16	RXDATAP1	0x000	R	**RX Data 1 Peek**
	Second frame read from FIFO.			
15	FERRP0	0	R	**Data Framing Error 0 Peek**
	Set if data in buffer has a framing error. Can be the result of a break condition.			
14	PERRP0	0	R	**Data Parity Error 0 Peek**
	Set if data in buffer has a parity error (asynchronous mode only).			
13:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:0	RXDATAP0	0x000	R	**RX Data 0 Peek**
	First frame read from FIFO.			

17.5.13 USARTn_TXDATAX - TX Buffer Data Extended Register

Offset	Bit Position																													
0x030	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0	0	0	0	0			0x000								
Access																	W	W	W	W	W			W								
Name																	RXENAT	TXDISAT	TXBREAK	TXTRIAT	UBRXAT			TXDATAX								

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15	RXENAT	0	W	**Enable RX After Transmission**
	Set to enable reception after transmission.			
14	TXDISAT	0	W	**Clear TXEN After Transmission**

Bit	Name	Reset	Access	Description
	Set to disable transmitter and release data bus directly after transmission.			
13	TXBREAK	0	W	**Transmit Data As Break**
	Set to send data as a break. Recipient will see a framing error or a break condition depending on its configuration and the value of TXDATA.			
12	TXTRIAT	0	W	**Set TXTRI After Transmission**
	Set to tristate transmitter by setting TXTRI after transmission.			
11	UBRXAT	0	W	**Unblock RX After Transmission**
	Set clear RXBLOCK after transmission, unblocking the receiver.			
10:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:0	TXDATAX	0x000	W	**TX Data**
	Use this register to write data to the USART. If TXEN is set, a transfer will be initiated at the first opportunity.			

17.5.14 USARTn_TXDATA - TX Buffer Data Register

Offset	Bit Position																													
0x034	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0x00							
Access																									W							
Name																									TXDATA							

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:0	TXDATA	0x00	W	**TX Data**
	This frame will be added to TX buffer. Only 8 LSB can be written using this register. 9th bit and control bits will be cleared.			

17.5.15 USARTn_TXDOUBLEX - TX Buffer Double Data Extended Register

Offset	Bit Position																													
0x038	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0			0x000									0	0	0	0	0			0x000								
Access	W	W	W	W	W			W									W	W	W	W	W			W								
Name	RXENAT1	TXDISAT1	TXBREAK1	TXTRIAT1	UBRXAT1			TXDATA1									RXENAT0	TXDISAT0	TXBREAK0	TXTRIAT0	UBRXAT0			TXDATA0								

Bit	Name	Reset	Access	Description
31	RXENAT1	0	W	**Enable RX After Transmission**
	Set to enable reception after transmission.			
30	TXDISAT1	0	W	**Clear TXEN After Transmission**

Bit	Name	Reset	Access	Description
	Set to disable transmitter and release data bus directly after transmission.			
29	TXBREAK1	0	W	**Transmit Data As Break**
	Set to send data as a break. Recipient will see a framing error or a break condition depending on its configuration and the value of USARTn_TXDATA.			
28	TXTRIAT1	0	W	**Set TXTRI After Transmission**
	Set to tristate transmitter by setting TXTRI after transmission.			
27	UBRXAT1	0	W	**Unblock RX After Transmission**
	Set clear RXBLOCK after transmission, unblocking the receiver.			
26:25	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
24:16	TXDATA1	0x000	W	**TX Data**
	Second frame to write to FIFO.			
15	RXENAT0	0	W	**Enable RX After Transmission**
	Set to enable reception after transmission.			
14	TXDISAT0	0	W	**Clear TXEN After Transmission**
	Set to disable transmitter and release data bus directly after transmission.			
13	TXBREAK0	0	W	**Transmit Data As Break**
	Set to send data as a break. Recipient will see a framing error or a break condition depending on its configuration and the value of TXDATA.			
12	TXTRIAT0	0	W	**Set TXTRI After Transmission**
	Set to tristate transmitter by setting TXTRI after transmission.			
11	UBRXAT0	0	W	**Unblock RX After Transmission**
	Set clear RXBLOCK after transmission, unblocking the receiver.			
10:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:0	TXDATA0	0x000	W	**TX Data**
	First frame to write to buffer.			

17.5.16 USARTn_TXDOUBLE - TX Buffer Double Data Register

Offset	Bit Position																													
0x03C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x00								0x00							
Access																	W								W							
Name																	TXDATA1								TXDATA0							

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:8	TXDATA1	0x00	W	**TX Data**
	Second frame to write to buffer.			
7:0	TXDATA0	0x00	W	**TX Data**
	First frame to write to buffer.			

17.5.17 USARTn_IF - Interrupt Flag Register

Offset	Bit Position																													
0x040	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																				0	0	0	0	0	0	0	0	0	0	0	1	0
Access																				R	R	R	R	R	R	R	R	R	R	R	R	R
Name																				CCF	SSM	MPAF	FERR	PERR	TXUF	TXOF	RXUF	RXOF	RXFULL	RXDATAV	TXBL	TXC

Bit	Name	Reset	Access	Description
31:13	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
12	CCF	0	R	**Collision Check Fail Interrupt Flag**
	Set when a collision check notices an error in the transmitted data.			
11	SSM	0	R	**Slave-Select In Master Mode Interrupt Flag**
	Set when the device is selected as a slave when in master mode.			
10	MPAF	0	R	**Multi-Processor Address Frame Interrupt Flag**
	Set when a multi-processor address frame is detected.			
9	FERR	0	R	**Framing Error Interrupt Flag**
	Set when a frame with a framing error is received while RXBLOCK is cleared.			
8	PERR	0	R	**Parity Error Interrupt Flag**
	Set when a frame with a parity error (asynchronous mode only) is received while RXBLOCK is cleared.			
7	TXUF	0	R	**TX Underflow Interrupt Flag**
	Set when operating as a synchronous slave, no data is available in the transmit buffer when the master starts transmission of a new frame.			
6	TXOF	0	R	**TX Overflow Interrupt Flag**
	Set when a write is done to the transmit buffer while it is full. The data already in the transmit buffer is preserved.			
5	RXUF	0	R	**RX Underflow Interrupt Flag**
	Set when trying to read from the receive buffer when it is empty.			
4	RXOF	0	R	**RX Overflow Interrupt Flag**
	Set when data is incoming while the receive shift register is full. The data previously in the shift register is lost.			
3	RXFULL	0	R	**RX Buffer Full Interrupt Flag**
	Set when the receive buffer becomes full.			
2	RXDATAV	0	R	**RX Data Valid Interrupt Flag**
	Set when data becomes available in the receive buffer.			
1	TXBL	1	R	**TX Buffer Level Interrupt Flag**
	Set when the buffer becomes empty if TXBIL is cleared, and is set whenever the transmit buffer goes from full to half-full or empty if TXBIL is set.			
0	TXC	0	R	**TX Complete Interrupt Flag**
	This interrupt is used after a transmission when both the TX buffer and shift register are empty.			

17.5.18 USARTn_IFS - Interrupt Flag Set Register

Offset	Bit Position																													
0x044	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																				0	0	0	0	0	0	0	0	0	0			0
Access																				W1	W1	W1	W1	W1	W1	W1	W1	W1	W1			W1
Name																				CCF	SSM	MPAF	FERR	PERR	TXUF	TXOF	RXUF	RXOF	RXFULL			TXC

Bit	Name	Reset	Access	Description
31:13	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
12	CCF	0	W1	**Set Collision Check Fail Interrupt Flag**
	Write to 1 to set the CCF interrupt flag.			
11	SSM	0	W1	**Set Slave-Select in Master mode Interrupt Flag**
	Write to 1 to set the SSM interrupt flag.			
10	MPAF	0	W1	**Set Multi-Processor Address Frame Interrupt Flag**
	Write to 1 to set the MPAF interrupt flag.			
9	FERR	0	W1	**Set Framing Error Interrupt Flag**
	Write to 1 to set the FERR interrupt flag.			
8	PERR	0	W1	**Set Parity Error Interrupt Flag**
	Write to 1 to set the PERR interrupt flag.			
7	TXUF	0	W1	**Set TX Underflow Interrupt Flag**
	Write to 1 to set the TXUF interrupt flag.			
6	TXOF	0	W1	**Set TX Overflow Interrupt Flag**
	Write to 1 to set the TXOF interrupt flag.			
5	RXUF	0	W1	**Set RX Underflow Interrupt Flag**
	Write to 1 to set the RXUF interrupt flag.			
4	RXOF	0	W1	**Set RX Overflow Interrupt Flag**
	Write to 1 to set the RXOF interrupt flag.			
3	RXFULL	0	W1	**Set RX Buffer Full Interrupt Flag**
	Write to 1 to set the RXFULL interrupt flag.			
2:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	TXC	0	W1	**Set TX Complete Interrupt Flag**
	Write to 1 to set the TXC interrupt flag.			

17.5.19 USARTn_IFC - Interrupt Flag Clear Register

Offset	Bit Position																													
0x048	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																				0	0	0	0	0	0	0	0	0	0			0
Access																				W1	W1	W1	W1	W1	W1	W1	W1	W1	W1			W1
Name																				CCF	SSM	MPAF	FERR	PERR	TXUF	TXOF	RXUF	RXOF	RXFULL			TXC

Bit	Name	Reset	Access	Description
31:13	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
12	CCF	0	W1	**Clear Collision Check Fail Interrupt Flag**
	Write to 1 to clear the CCF interrupt flag.			
11	SSM	0	W1	**Clear Slave-Select In Master Mode Interrupt Flag**
	Write to 1 to clear the SSM interrupt flag.			
10	MPAF	0	W1	**Clear Multi-Processor Address Frame Interrupt Flag**
	Write to 1 to clear the MPAF interrupt flag.			
9	FERR	0	W1	**Clear Framing Error Interrupt Flag**
	Write to 1 to clear the FERR interrupt flag.			
8	PERR	0	W1	**Clear Parity Error Interrupt Flag**
	Write to 1 to clear the PERR interrupt flag.			
7	TXUF	0	W1	**Clear TX Underflow Interrupt Flag**
	Write to 1 to clear the TXUF interrupt flag.			
6	TXOF	0	W1	**Clear TX Overflow Interrupt Flag**
	Write to 1 to clear the TXOF interrupt flag.			
5	RXUF	0	W1	**Clear RX Underflow Interrupt Flag**
	Write to 1 to clear the RXUF interrupt flag.			
4	RXOF	0	W1	**Clear RX Overflow Interrupt Flag**
	Write to 1 to clear the RXOF interrupt flag.			
3	RXFULL	0	W1	**Clear RX Buffer Full Interrupt Flag**
	Write to 1 to clear the RXFULL interrupt flag.			
2:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	TXC	0	W1	**Clear TX Complete Interrupt Flag**
	Write to 1 to clear the TXC interrupt flag.			

17.5.20 USARTn_IEN - Interrupt Enable Register

Offset	Bit Position																													
0x04C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																				0	0	0	0	0	0	0	0	0	0	0	0	0
Access																				RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Name																				CCF	SSM	MPAF	FERR	PERR	TXUF	TXOF	RXUF	RXOF	RXFULL	RXDATAV	TXBL	TXC

Bit	Name	Reset	Access	Description
31:13	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
12	CCF	0	RW	**Collision Check Fail Interrupt Enable**
	Enable interrupt on collision check error detected.			
11	SSM	0	RW	**Slave-Select In Master Mode Interrupt Enable**
	Enable interrupt on slave-select in master mode.			
10	MPAF	0	RW	**Multi-Processor Address Frame Interrupt Enable**
	Enable interrupt on multi-processor address frame.			
9	FERR	0	RW	**Framing Error Interrupt Enable**
	Enable interrupt on framing error.			

Bit	Name	Reset	Access	Description
8	PERR	0	RW	**Parity Error Interrupt Enable**
	Enable interrupt on parity error (asynchronous mode only).			
7	TXUF	0	RW	**TX Underflow Interrupt Enable**
	Enable interrupt on TX underflow.			
6	TXOF	0	RW	**TX Overflow Interrupt Enable**
	Enable interrupt on TX overflow.			
5	RXUF	0	RW	**RX Underflow Interrupt Enable**
	Enable interrupt on RX underflow.			
4	RXOF	0	RW	**RX Overflow Interrupt Enable**
	Enable interrupt on RX overflow.			
3	RXFULL	0	RW	**RX Buffer Full Interrupt Enable**
	Enable interrupt on RX Buffer full.			
2	RXDATAV	0	RW	**RX Data Valid Interrupt Enable**
	Enable interrupt on RX data.			
1	TXBL	0	RW	**TX Buffer Level Interrupt Enable**
	Enable interrupt on TX buffer level.			
0	TXC	0	RW	**TX Complete Interrupt Enable**
	Enable interrupt on TX complete.			

17.5.21 USARTn_IRCTRL - IrDA Control Register

Offset	Bit Position																														
0x050	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0	0x0			0	0x0		0
Access																									RW	RW			RW	RW		RW
Name																									IRPRSEN	IRPRSSEL			IRFILT	IRPW		IREN

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7	IRPRSEN	0	RW	**IrDA PRS Channel Enable**
	Enable the PRS channel selected by IRPRSSEL as input to IrDA module instead of TX.			
6:4	IRPRSSEL	0x0	RW	**IrDA PRS Channel Select**
	A PRS can be used as input to the pulse modulator instead of TX. This value selects the channel to use.			

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected
1	PRSCH1	PRS Channel 1 selected
2	PRSCH2	PRS Channel 2 selected
3	PRSCH3	PRS Channel 3 selected
4	PRSCH4	PRS Channel 4 selected
5	PRSCH5	PRS Channel 5 selected

Bit	Name	Reset	Access	Description
3	IRFILT	0	RW	**IrDA RX Filter**
	Set to enable filter on IrDA demodulator.			

Value	Description
0	No filter enabled

Bit	Name	Reset	Access	Description

Value	Description
1	Filter enabled. IrDA pulse must be high for at least 4 consecutive clock cycles to be detected

2:1 IRPW 0x0 RW **IrDA TX Pulse Width**

Configure the pulse width generated by the IrDA modulator as a fraction of the configured USART bit period.

Value	Mode	Description
0	ONE	IrDA pulse width is 1/16 for OVS=0 and 1/8 for OVS=1
1	TWO	IrDA pulse width is 2/16 for OVS=0 and 2/8 for OVS=1
2	THREE	IrDA pulse width is 3/16 for OVS=0 and 3/8 for OVS=1
3	FOUR	IrDA pulse width is 4/16 for OVS=0 and 4/8 for OVS=1

0 IREN 0 RW **Enable IrDA Module**

Enable IrDA module and rout USART signals through it.

17.5.22 USARTn_ROUTE - I/O Routing Register

Offset	Bit Position																													
0x054	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0x0						0	0	0	0
Access																							RW						RW	RW	RW	RW
Name																							LOCATION						CLKPEN	CSPEN	TXPEN	RXPEN

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	LOCATION	0x0	RW	**I/O Location**

Decides the location of the USART I/O pins.

Value	Mode	Description
0	LOC0	Location 0
1	LOC1	Location 1
2	LOC2	Location 2
3	LOC3	Location 3
4	LOC4	Location 4
5	LOC5	Location 5
6	LOC6	Location 6

7:4 *Reserved* *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

3 CLKPEN 0 RW **CLK Pin Enable**

When set, the CLK pin of the USART is enabled.

Value	Description
0	The USn_CLK pin is disabled
1	The USn_CLK pin is enabled

2 CSPEN 0 RW **CS Pin Enable**

When set, the CS pin of the USART is enabled.

Value	Description
0	The USn_CS pin is disabled
1	The USn_CS pin is enabled

1 TXPEN 0 RW **TX Pin Enable**

When set, the TX/MOSI pin of the USART is enabled

Bit	Name	Reset	Access	Description

Value	Description
0	The U(S)n_TX (MOSI) pin is disabled
1	The U(S)n_TX (MOSI) pin is enabled

0	RXPEN	0	RW	**RX Pin Enable**

When set, the RX/MISO pin of the USART is enabled.

Value	Description
0	The U(S)n_RX (MISO) pin is disabled
1	The U(S)n_RX (MISO) pin is enabled

17.5.23 USARTn_INPUT - USART Input Register

Offset	Bit Position																														
0x058	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																												0		0x0		
Access																												RW		RW		
Name																												RXPRS		RXPRSSEL		

Bit	Name	Reset	Access	Description
31:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	RXPRS	0	RW	**PRS RX Enable**

When set, the PRS channel selected as input to RX.

3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2:0	RXPRSSEL	0x0	RW	**RX PRS Channel Select**

Select PRS channel as input to RX.

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected
1	PRSCH1	PRS Channel 1 selected
2	PRSCH2	PRS Channel 2 selected
3	PRSCH3	PRS Channel 3 selected
4	PRSCH4	PRS Channel 4 selected
5	PRSCH5	PRS Channel 5 selected

17.5.24 USARTn_I2SCTRL - I2S Control Register

Offset	Bit Position																														
0x05C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0x0				0	0	0	0	0
Access																							RW				RW	RW	RW	RW	RW
Name																						FORMAT						DELAY	DMASPLIT	JUSTIFY	MONO	EN

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
10:8	FORMAT	0x0	RW	**I2S Word Format**

Configure the data-width used internally for I2S data

Value	Mode	Description
0	W32D32	32-bit word, 32-bit data
1	W32D24M	32-bit word, 32-bit data with 8 lsb masked
2	W32D24	32-bit word, 24-bit data
3	W32D16	32-bit word, 16-bit data
4	W32D8	32-bit word, 8-bit data
5	W16D16	16-bit word, 16-bit data
6	W16D8	16-bit word, 8-bit data
7	W8D8	8-bit word, 8-bit data

7:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	DELAY	0	RW	**Delay on I2S data**

Set to add a one-cycle delay between a transition on the word-clock and the start of the I2S word. Should be set for standard I2S format

3	DMASPLIT	0	RW	**Separate DMA Request For Left/Right Data**

When set DMA requests for right-channel data are put on the TXBLRIGHT and RXDATAVRIGHT DMA requests.

2	JUSTIFY	0	RW	**Justification of I2S Data**

Determines whether the I2S data is left or right justified

Value	Mode	Description
0	LEFT	Data is left-justified
1	RIGHT	Data is right-justified

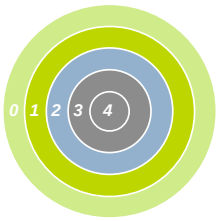
1	MONO	0	RW	**Stero or Mono**

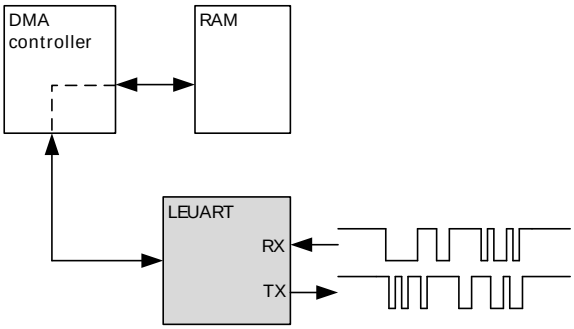
Switch between stereo and mono mode. Set for mono

0	EN	0	RW	**Enable I2S Mode**

Set the U(S)ART in I2S mode.

18 LEUART - Low Energy Universal Asynchronous Receiver/Transmitter





Quick Facts

What?

The LEUART provides full UART communication using a low frequency 32.768 kHz clock, and has special features for communication without CPU intervention.

Why?

It allows UART communication to be performed in low energy modes, using only a few µA during active communication and only 150 nA when waiting for incoming data.

How?

A low frequency clock signal allows communication with less energy. Using DMA, the LEUART can transmit and receive data with minimal CPU intervention. Special UART-frames can be configured to help control the data flow, further automating data transmission.

18.1 Introduction

The unique LEUART™, the Low Energy UART, is a UART that allows two-way UART communication on a strict power budget. Only a 32.768 kHz clock is needed to allow UART communication at baud rates up to 9600.

Even when the EFM is in low energy mode EM2 (with most core functionality turned off), the LEUART can wait for an incoming UART frame while having an extremely low energy consumption. When a UART frame is completely received, the CPU can quickly be woken up. Alternatively, multiple frames can be transferred via the Direct Memory Access (DMA) module into RAM memory before waking up the CPU.

Received data can optionally be blocked until a configurable start frame is detected. A signal frame can be configured to generate an interrupt to indicate e.g. the end of a data transmission. The start frame and signal frame can be used in combination for instance to handle higher level communication protocols.

Similarly, data can be transmitted in EM2 either on a frame-by-frame basis with data from the CPU or through use of the DMA.

The LEUART includes all necessary hardware support to make asynchronous serial communication possible with minimum of software intervention and energy consumption.

18.2 Features

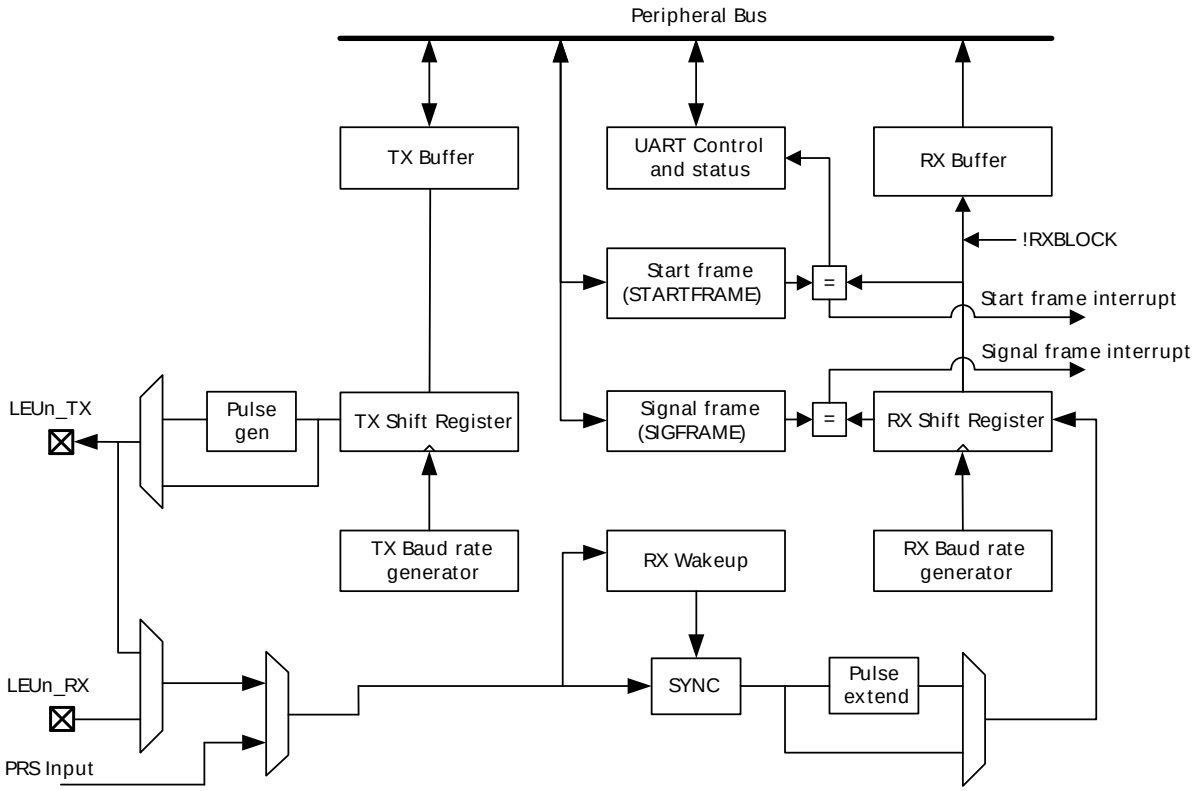
- Low energy asynchronous serial communications
- Full/half duplex communication
- Separate TX / RX enable
- Separate double buffered transmit buffer and receive buffer
- Programmable baud rate, generated as a fractional division of the LFBCLK
  - Supports baud rates from 300 baud/s to 9600 baud/s

- Can use a high frequency clock source for even higher baud rates
- Configurable number of data bits: 8 or 9 (plus parity bit, if enabled)
- Configurable parity: off, even or odd
  - HW parity bit generation and check
- Configurable number of stop bits, 1 or 2
- Capable of sleep-mode wake-up on received frame
  - Either wake-up on any received byte or
  - Wake up only on specified start and signal frames
- Supports transmission and reception in EM0, EM1 and EM2 with
  - Full DMA support
  - Specified start-byte can start reception automatically
- IrDA modulator (pulse generator, pulse extender)
- Multi-processor mode
- Loopback mode
  - Half duplex communication
  - Communication debugging
- PRS RX input

18.3 Functional Description

An overview of the LEUART module is shown in Figure 18.1 (p. 416) .

Figure 18.1. LEUART Overview

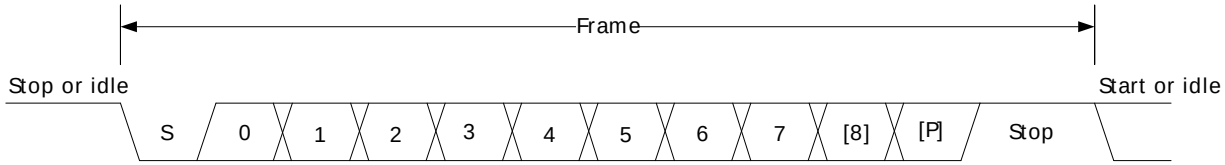


18.3.1 Frame Format

The frame format used by the LEUART consists of a set of data bits in addition to bits for synchronization and optionally a parity bit for error checking. A frame starts with one start-bit (S), where the line is driven

low for one bit-period. This signals the start of a frame, and is used for synchronization. Following the start bit are 8 or 9 data bits and an optional parity bit. The data is transmitted with the least significant bit first. Finally, a number of stop-bits, where the line is driven high, end the frame. The frame format is shown in Figure 18.2 (p. 417) .

Figure 18.2. LEUART Asynchronous Frame Format



The number of data bits in a frame is set by DATABITS in LEUARTn_CTRL, and the number of stop-bits is set by STOPBITS in LEUARTn_CTRL. Whether or not a parity bit should be included, and whether it should be even or odd is defined by PARITY in LEUARTn_CTRL. For communication to be possible, all parties of an asynchronous transfer must agree on the frame format being used.

The frame format used by the LEUART can be inverted by setting INV in LEUARTn_CTRL. This affects the entire frame, resulting in a low idle state, a high start-bit, inverted data and parity bits, and low stop-bits. INV should only be changed while the receiver is disabled.

18.3.1.1 Parity Bit Calculation and Handling

Hardware automatically inserts parity bits into outgoing frames and checks the parity bits of incoming frames. The possible parity modes are defined in Table 18.1 (p. 417) . When even parity is chosen, a parity bit is inserted to make the number of high bits (data + parity) even. If odd parity is chosen, the parity bit makes the total number of high bits odd. When parity bits are disabled, which is the default configuration, the parity bit is omitted.

Table 18.1. LEUART Parity Bit

PARITY [1:0]	Description
00	No parity (default)
01	Reserved
10	Even parity
11	Odd parity

See Section 18.3.5.4 (p. 422) for more information on parity bit handling.

18.3.2 Clock Source

The LEUART clock source is selected by the LFB bit field the CMU_LFCLKSEL register. The clock is prescaled by the LEUARTn bitfield in the CMU_LFBPRESC0 register and enabled by the LEUARTn bit in the CMU_LFBCLKEN0.

To use this module, the LE interface clock must be enabled in CMU_HFCORECLKEN0, in addition to the module clock.

18.3.3 Clock Generation

The LEUART clock defines the transmission and reception data rate. The clock generator employs a fractional clock divider to allow baud rates that are not attainable by integral division of the 32.768 kHz clock that drives the LEUART.

The clock divider used in the LEUART is a 12-bit value, with a 7-bit integral part and a 5-bit fractional part. The baud rate of the LEUART is given by :

LEUART Baud Rate Equation

$$br = fLEUARTn/(1 + LEUARTn\_CLKDIV/256) \quad (18.1)$$

where fLEUARTn is the clock frequency supplied to the LEUART. The value of LEUARTn_CLKDIV thus defines the baud rate of the LEUART. The integral part of the divider is right-aligned in the upper 24 bits of LEUARTn_CLKDIV and the fractional part is left-aligned in the lower 8 bits. The divider is thus a 256th of LEUARTn_CLKDIV as seen in the equation.

For a desired baud rate $br_{DESIRED}$, LEUARTn_CLKDIV can be calculated by using:

LEUART CLKDIV Equation

$$LEUARTn\_CLKDIV = 256 \times (fLEUARTn/br_{DESIRED} - 1) \quad (18.2)$$

Table 18.2 (p. 418) lists a set of desired baud rates and the closest baud rates reachable by the LEUART with a 32.768 kHz clock source. It also shows the average baud rate error.

Table 18.2. LEUART Baud Rates

Desired baud rate [baud/s]	LEUARTn_CLKDIV	LEUARTn_CLKDIV/256	Actual baud rate [baud/s]	Error [%]
300	27704	108,21875	300,0217	0,01
600	13728	53,625	599,8719	-0,02
1200	6736	26,3125	1199,744	-0,02
2400	3240	12,65625	2399,487	-0,02
4800	1488	5,8125	4809,982	0,21
9600	616	2,40625	9619,963	0,21

18.3.4 Data Transmission

Data transmission is initiated by writing data to the transmit buffer using one of the methods described in Section 18.3.4.1 (p. 418) . When the transmission shift register is empty and ready for new data, a frame from the transmit buffer is loaded into the shift register, and if the transmitter is enabled, transmission begins. When the frame has been transmitted, a new frame is loaded into the shift register if available, and transmission continues. If the transmit buffer is empty, the transmitter goes to an idle state, waiting for a new frame to become available. Transmission is enabled through the command register LEUARTn_CMD by setting TXEN, and disabled by setting TXDIS. When the transmitter is disabled using TXDIS, any ongoing transmission is aborted, and any frame currently being transmitted is discarded. When disabled, the TX output goes to an idle state, which by default is a high value. Whether or not the transmitter is enabled at a given time can be read from TXENS in LEUARTn_STATUS. After a transmission, when there is no more data in the shift register or transmit buffer, the TXC flag in LEUARTn_STATUS and the TXC interrupt flag in LEUARTn_IF are set, signaling that the transmitter is idle. The TXC status flag is cleared when a new byte becomes available for transmission, but the TXC interrupt flag must be cleared by software.

18.3.4.1 Transmit Buffer Operation

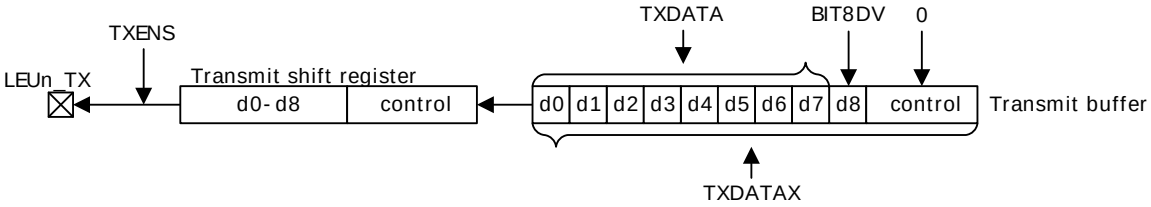
A frame can be loaded into the transmit buffer by writing to LEUARTn_TXDATA or LEUARTn_TXDATAX. Using LEUARTn_TXDATA allows 8 bits to be written to the buffer. If 9 bit frames are used, the 9th bit will in that case be set to the value of BIT8DV in LEUARTn_CTRL. To set the 9th bit directly and/or use transmission control, LEUARTn_TXDATAX must be used. When writing data to the transmit buffer using LEUARTn_TXDATAX, the 9th bit written to LEUARTn_TXDATAX overrides the value in BIT8DV, and alone defines the 9th bit that is transmitted if 9-bit frames are used.

If a write is attempted to the transmit buffer when it is not empty, the TXOF interrupt flag in LEUARTn_IF is set, indicating the overflow. The data already in the buffer is in that case preserved, and no data is written.

In addition to the interrupt flag TXC in LEUARTn_IF and the status flag TXC in LEUARTn_STATUS which are set when the transmitter becomes idle, TXBL in LEUARTn_STATUS and the TXBL interrupt flag in LEUARTn_IF are used to indicate the level of the transmit buffer. Whenever the transmit buffer becomes empty, these flags are set high. Both the TXBL status flag and the TXBL interrupt flag are cleared automatically when data is written to the transmit buffer.

The transmit buffer, including the TX shift register can be cleared by setting command bit CLEARTX in LEUARTn_CMD. This will prevent the LEUART from transmitting the data in the buffer and shift register, and will make them available for new data. Any frame currently being transmitted will not be aborted. Transmission of this frame will be completed. An overview of the operation of the transmitter is shown in Figure 18.3 (p. 419) .

Figure 18.3. LEUART Transmitter Overview



18.3.4.2 Frame Transmission Control

The transmission control bits, which can be written using LEUARTn_TXDATAX, affect the transmission of the written frame. The following options are available:

- Generate break: By setting WBREAK, the output will be held low during the first stop-bit period to generate a framing error. A receiver that supports break detection detects this state, allowing it to be used e.g. for framing of larger data packets. The line is driven high for one baud period before the next frame is transmitted so the next start condition can be identified correctly by the recipient. Continuous breaks lasting longer than an UART frame are thus not supported by the LEUART. GPIO can be used for this. Note that when AUTOTRI in LEUARTn_CTRL is used, the transmitter is not tristated before the high-bit after the break has been transmitted.
- Disable transmitter after transmission: If TXDISAT is set, the transmitter is disabled after the frame has been fully transmitted.
- Enable receiver after transmission: If RXENAT is set, the receiver is enabled after the frame has been fully transmitted. It is enabled in time to detect a start-bit directly after the last stop-bit has been transmitted.

The transmission control bits in the LEUART cannot tristate the transmitter. This is performed automatically by hardware however, if AUTOTRI in LEUARTn_CTRL is set. See Section 18.3.7 (p. 424) for more information on half duplex operation.

18.3.4.3 Jitter in Transmitted Data

Internally the LEUART module uses only the positive edges of the 32.768 kHz clock (LFBCLK) for transmission and reception. Transmitted data will thus have jitter equal to the difference between the optimal data set-up location and the closest positive edge on the 32.768 kHz clock. The jitter in on the location data is set up by the transmitter will thus be no more than half a clock period according to the optimal set-up location. The jitter in the period of a single baud output by the transmitter will never be more than one clock period.

18.3.5 Data Reception

Data reception is enabled by setting RXEN in LEUARTn_CMD. When the receiver is enabled, it actively samples the input looking for a transition from high to low indicating the start baud of a new frame. When a start baud is found, reception of the new frame begins if the receive shift register is empty and ready for new data. When the frame has been received, it is pushed into the receive buffer, making the shift register ready for another frame of data, and the receiver starts looking for another start baud. If the receive buffer is full, the received frame remains in the shift register until more space in the receive buffer is available.

If an incoming frame is detected while both the receive buffer and the receive shift register are full, the data in the receive shift register is overwritten, and the RXOF interrupt flag in LEUARTn_IF is set to indicate the buffer overflow.

The receiver can be disabled by setting the command bit RXDIS in LEUARTn_CMD. Any frame currently being received when the receiver is disabled is discarded. Whether or not the receiver is enabled at a given time can be read out from RXENS in LEUARTn_STATUS.

18.3.5.1 Receive Buffer Operation

When data becomes available in the receive buffer, the RXDATAV flag in LEUARTn_STATUS and the RXDATAV interrupt flag in LEUARTn_IF are set. Both the RXDATAV status flag and the RXDATAV interrupt flag are cleared by hardware when data is no longer available, i.e. when data has been read out of the buffer.

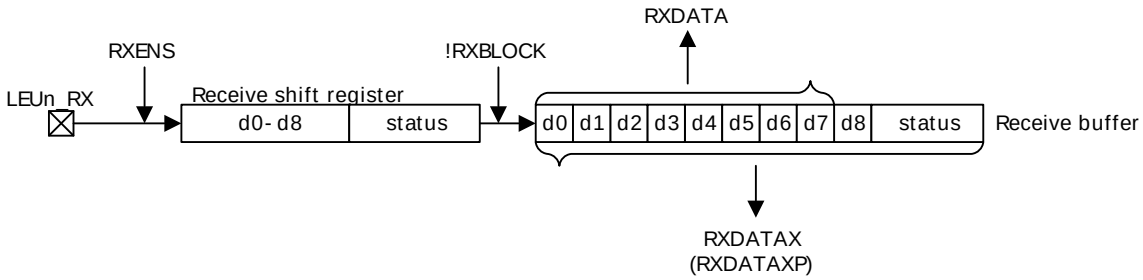
Data can be read from receive buffer using either LEUARTn_RXDATA or LEUARTn_RXDATAX. LEUARTn_RXDATA gives access to the 8 least significant bits of the received frame, while LEUARTn_RXDATAX must be used to get access to the 9th, most significant bit. The latter register also contains status information regarding the frame.

When a frame is read from the receive buffer using LEUARTn_RXDATA or LEUARTn_RXDATAX, the frame is removed from the buffer, making room for a new one. If an attempt is done to read more frames from the buffer than what is available, the RXUF interrupt flag in LEUARTn_IF is set to signal the underflow, and the data read from the buffer is undefined.

Frames can also be read from the receive buffer without removing the data by using LEUARTn_RXDATAXP, which gives access to the frame in the buffer including control bits. Data read from this register when the receive buffer is empty is undefined. No underflow interrupt is generated by a read using LEUARTn_RXDATAXP, i.e. the RXUF interrupt flag is never set as a result of reading from LEUARTn_RXDATAXP.

An overview of the operation of the receiver is shown in Figure 18.4 (p. 421) .

Figure 18.4. LEUART Receiver Overview



18.3.5.2 Blocking Incoming Data

When using hardware frame recognition, as detailed in Section 18.3.5.6 (p. 422) , Section 18.3.5.7 (p. 423) , and Section 18.3.5.8 (p. 423) , it is necessary to be able to let the receiver sample incoming frames without passing the frames to software by loading them into the receive buffer. This is accomplished by blocking incoming data.

Incoming data is blocked as long as RXBLOCK in LEUARTn_STATUS is set. When blocked, frames received by the receiver will not be loaded into the receive buffer, and software is not notified by the RXDATAV bit in LEUARTn_STATUS or the RXDATAV interrupt flag in LEUARTn_IF at their arrival. For data to be loaded into the receive buffer, RXBLOCK must be cleared in the instant a frame is fully received by the receiver. RXBLOCK is set by setting RXBLOCKEN in LEUARTn_CMD and disabled by setting RXBLOCKDIS also in LEUARTn_CMD. There are two exceptions where data is loaded into the receive buffer even when RXBLOCK is set. The first is when an address frame is received when in operating in multi-processor mode as shown in Section 18.3.5.8 (p. 423) . The other case is when receiving a start-frame when SFUBRX in LEUARTn_CTRL is set; see Section 18.3.5.6 (p. 422)

Frames received containing framing or parity errors will not result in the FERR and PERR interrupt flags in LEUARTn_IF being set while RXBLOCK is set. Hardware recognition is not applied to these erroneous frames, and they are silently discarded.

Note

If a frame is received while RXBLOCK in LEUARTn_STATUS is cleared, but stays in the receive shift register because the receive buffer is full, the received frame will be loaded into the receive buffer when space becomes available even if RXBLOCK is set at that time.

The overflow interrupt flag RXOF in LEUARTn_IF will be set if a frame in the receive shift register, waiting to be loaded into the receive buffer is overwritten by an incoming frame even though RXBLOCK is set.

18.3.5.3 Data Sampling

The receiver samples each incoming baud as close as possible to the middle of the baud-period. Except for the start-bit, only a single sample is taken of each of the incoming bauds.

The length of a baud-period is given by 1 + LEUARTn_CLKDIV/256, as a number of 32.768 kHz clock periods. Let the clock cycle where a start-bit is first detected be given the index 0. The optimal sampling point for each baud in the UART frame is then given by the following equation:

LEUART Optimal Sampling Point

$$S_{opt}(n) = n\ (1 + LEUARTn\_CLKDIV/256) + CLKDIV/512 \quad (18.3)$$

where n is the bit-index.

Since samples are only done on the positive edges of the 32.768 kHz clock, the actual samples are performed on the closest positive edge, i.e. the edge given by the following equation:

LEUART Actual Sampling Point

$$S(n) = floor(n\ x\ (1 + LEUARTn\_CLKDIV/256) + LEUARTn\_CLKDIV/512) \quad (18.4)$$

The sampling location will thus have jitter according to difference between S_{opt} and S. The start-bit is found at n=0, then follows the data bits, any parity bit, and the stop bits.

If the value of the start-bit is found to be high, then the start-bit is discarded, and the receiver waits for a new start-bit.

18.3.5.4 Parity Error

When the parity bit is enabled, a parity check is automatically performed on incoming frames. When a parity error is detected in a frame, the data parity error bit PERR in the frame is set, as well as the interrupt flag PERR. Frames with parity errors are loaded into the receive buffer like regular frames.

PERR can be accessed by reading the frame from the receive buffer using the LEUARTn_RXDATAX register.

18.3.5.5 Framing Error and Break Detection

A framing error is the result of a received frame where the stop bit was sampled to a value of 0. This can be the result of noise and baud rate errors, but can also be the result of a break generated by the transmitter on purpose.

When a framing error is detected, the framing error bit FERR in the received frame is set. The interrupt flag FERR in LEUARTn_IF is also set. Frames with framing errors are loaded into the receive buffer like regular frames.

FERR can be accessed by reading the frame from the receive buffer using the LEUARTn_RXDATAX or LEUARTn_RXDATAXP registers.

18.3.5.6 Programmable Start Frame

The LEUART can be configured to start receiving data when a special start frame is detected on the input. This can be useful when operating in low energy modes, allowing other devices to gain the attention of the LEUART by transmitting a given frame.

When SFUBRX in LEUARTn_CTRL is set, an incoming frame matching the frame defined in LEUARTn_STARTFRAME will result in RXBLOCK in LEUARTn_STATUS being cleared. This can be used to enable reception when a specified start frame is detected. If the receiver is enabled and blocked, i.e. RXENS and RXBLOCK in LEUARTn_STATUS are set, the receiver will receive all incoming frames, but unless an incoming frame is a start frame it will be discarded and not loaded into the receive buffer. When a start frame is detected, the block is cleared, and frames received from that point, including the start frame, are loaded into the receive buffer.

An incoming start frame results in the STARTF interrupt flag in LEUARTn_IF being set, regardless of the value of SFUBRX in LEUARTn_CTRL. This allows an interrupt to be made when the start frame is detected.

When 8 data-bit frame formats are used, only the 8 least significant bits of LEUARTn_STARTFRAME are compared to incoming frames. The full length of LEUARTn_STARTFRAME is used when operating with frames consisting of 9 data bits.

Note

The receiver must be enabled for start frames to be detected. In addition, a start frame with a parity error or framing error is not detected as a start frame.

18.3.5.7 Programmable Signal Frame

As well as the configurable start frame, a special signal frame can be specified. When a frame matching the frame defined in LEUARTn_SIGFRAME is detected by the receiver, the SIGF interrupt flag in LEUARTn_IF is set. As for start frame detection, the receiver must be enabled for signal frames to be detected.

One use of the programmable signal frame is to signal the end of a multi-frame message transmitted to the LEUART. An interrupt will then be triggered when the packet has been completely received, allowing software to process it. Used in conjunction with the programmable start frame and DMA, this makes it possible for the LEUART to automatically begin the reception of a packet on a specified start frame, load the entire packet into memory, and give an interrupt when reception of a packet has completed. The device can thus wait for data packets in EM2, and only be woken up when a packet has been completely received.

A signal frame with a parity error or framing error is not detected as a signal frame.

18.3.5.8 Multi-Processor Mode

To simplify communication between multiple processors and maintain compatibility with the USART, the LEUART supports a multi-processor mode. In this mode the 9th data bit in each frame is used to indicate whether the content of the remaining 8 bits is data or an address.

When multi-processor mode is enabled, an incoming 9-bit frame with the 9th bit equal to the value of MPAB in LEUARTn_CTRL is identified as an address frame. When an address frame is detected, the MPAF interrupt flag in LEUARTn_IF is set, and the address frame is loaded into the receive register. This happens regardless of the value of RXBLOCK in LEUARTn_STATUS.

Multi-processor mode is enabled by setting MPM in LEUARTn_CTRL. The mode can be used in buses with multiple slaves, allowing the slaves to be addressed using the special address frames. An addressed slave, which was previously blocking reception using RXBLOCK, would then unblock reception, receive a message from the bus master, and then block reception again, waiting for the next message. See the USART for a more detailed example.

Note

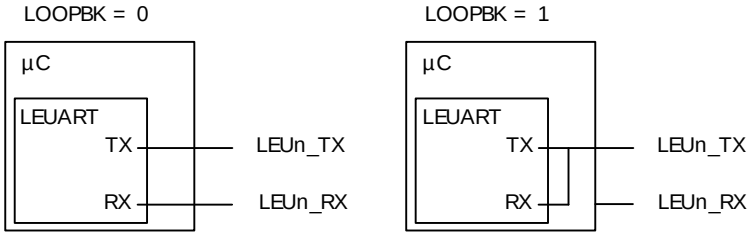
The programmable start frame functionality can be used for automatic address matching, enabling reception on a correctly configured incoming frame.

An address frame with a parity error or a framing error is not detected as an address frame.

18.3.6 Loopback

The LEUART receiver samples LEUn_RX by default, and the transmitter drives LEUn_TX by default. This is not the only configuration however. When LOOPBK in LEUARTn_CTRL is set, the receiver is connected to the LEUn_TX pin as shown in Figure 18.5 (p. 424) . This is useful for debugging, as the LEUART can receive the data it transmits, but it is also used to allow the LEUART to read and write to the same pin, which is required for some half duplex communication modes. In this mode, the LEUn_TX pin must be enabled as an output in the GPIO.

Figure 18.5. LEUART Local Loopback



18.3.7 Half Duplex Communication

When doing full duplex communication, two data links are provided, making it possible for data to be sent and received at the same time. In half duplex mode, data is only sent in one direction at a time. There are several possible half duplex setups, as described in the following sections.

18.3.7.1 Single Data-link

In this setup, the LEUART both receives and transmits data on the same pin. This is enabled by setting LOOPBK in LEUARTn_CTRL, which connects the receiver to the transmitter output. Because they are both connected to the same line, it is important that the LEUART transmitter does not drive the line when receiving data, as this would corrupt the data on the line.

When communicating over a single data-link, the transmitter must thus be tristated whenever not transmitting data. If AUTOTRI in LEUARTn_CTRL is set, the LEUART automatically tristates LEUn_TX whenever the transmitter is inactive. It is then the responsibility of the software protocol to make sure the transmitter is not transmitting data whenever incoming data is expected.

The transmitter can also be tristated from software by configuring the GPIO pin as an input and disabling the LEUART output on LEUn_TX.

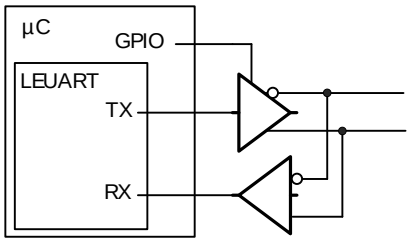
Note

Another way to tristate the transmitter is to enable wired-and or wired-or mode in GPIO. For wired-and mode, outputting a 1 will be the same as tristating the output, and for wired-or mode, outputting a 0 will be the same as tristating the output. This can only be done on buses with a pull-up or pull-down resistor respectively.

18.3.7.2 Single Data-link with External Driver

Some communication schemes, such as RS-485 rely on an external driver. Here, the driver has an extra input which enables it, and instead of Tristating the transmitter when receiving data, the external driver must be disabled. The USART has hardware support for automatically turning the driver on and off. When using the LEUART in such a setup, the driver must be controlled by a GPIO. Figure 18.6 (p. 424) shows an example configuration using an external driver.

Figure 18.6. LEUART Half Duplex Communication with External Driver



18.3.7.3 Two Data-links

Some limited devices only support half duplex communication even though two data links are available. In this case software is responsible for making sure data is not transmitted when incoming data is expected.

18.3.8 Transmission Delay

By configuring TXDELAY in LEUARTn_CTRL, the transmitter can be forced to wait a number of bit-periods from it is ready to transmit data, to it actually transmits the data. This delay is only applied to the first frame transmitted after the transmitter has been idle. When transmitting frames back-to-back the delay is not introduced between the transmitted frames.

This is useful on half duplex buses, because the receiver always returns received frames to software during the first stop-bit. The bus may still be driven for up to 3 baud periods, depending on the current frame format. Using the transmission delay, a transmission can be started when a frame is received, and it is possible to make sure that the transmitter does not begin driving the output before the frame on the bus is completely transmitted.

18.3.9 PRS RX Input

The LEUART can be configured to receive data directly from the PRS channel by setting RX_PRS in LEUARTn_INPUT. The PRS channel used can be selected using RX_PRS_SEL in LEUARTn_INPUT.

18.3.10 DMA Support

The LEUART has full DMA support in energy modes EM0 – EM2. The DMA controller can write to the transmit buffer using the registers LEUARTn_TXDATA and LEUARTn_TXDATAX, and it can read from receive buffer using the registers LEUARTn_RXDATA and LEUARTn_RXDATAX. This enables single byte transfers and 9 bit data + control/status bits transfers both to and from the LEUART. The DMA will start up the HFRCO and run from this when it is waken by the LEUART in EM2. The HFRCO is disabled once the transaction is done.

A request for the DMA controller to read from the receive buffer can come from one of the following sources:

- Receive buffer full

A write request can come from one of the following sources:

- Transmit buffer and shift register empty. No data to send.
- Transmit buffer empty

In some cases, it may be sensible to temporarily stop DMA access to the LEUART when a parity or framing error has occurred. This is enabled by setting ERRSDMA in LEUARTn_CTRL. When this bit is set, the DMA controller will not get requests from the receive buffer if a framing error or parity error is detected in the received byte. The ERRSDMA bit applies only to the RX DMA.

When operating in EM2, the DMA controller must be powered up in order to perform the transfer. This is automatically performed for read operations if RXDMAWU in LEUARTn_CTRL is set and for write operations if TXDMAWU in LEUARTn_CTRL is set. To make sure the DMA controller still transfers bits to and from the LEUART in low energy modes, these bits must thus be configured accordingly.

Note

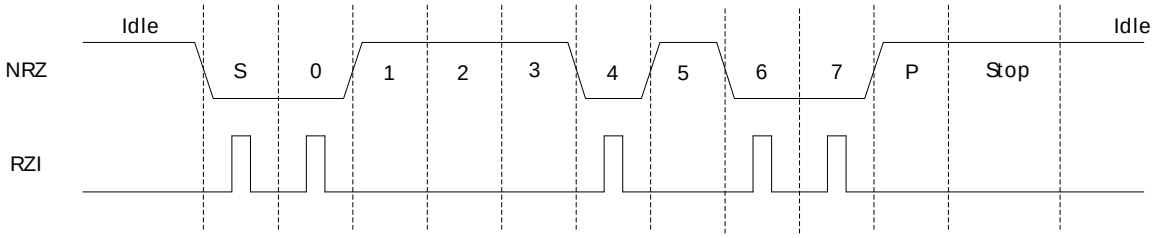
When RXDMAWU or TXDMAWU is set, the system will not be able to go to EM2/EM3 before all related LEUART DMA requests have been processed. This means that if RXDMAWU is set and the LEUART receives a frame, the system will not be able to go to

EM2/EM3 before the frame has been read from the LEUART. In order for the system to go to EM2 during the last byte transmission, LEUART_CTRL_TXDMAWU must be cleared in the DMA interrupt service routine. This is because TXBL will be high during that last byte transfer.

18.3.11 Pulse Generator/ Pulse Extender

The LEUART has an optional pulse generator for the transmitter output, and a pulse extender on the receiver input. These are enabled by setting PULSEEN in LEUARTn_PULSECTRL, and with INV in LEUARTn_CTRL set, they will change the output/input format of the LEUART from NRZ to RZI as shown in Figure 18.7 (p. 426) .

Figure 18.7. LEUART - NRZ vs. RZI



If PULSEEN in LEUARTn_PULSECTRL is set while INV in LEUARTn_CTRL is cleared, the output waveform will like RZI shown in Figure 18.7 (p. 426) , only inverted.

The width of the pulses from the pulse generator can be configured using PULSEW in LEUARTn_PULSECTRL. The generated pulse width is PULSEW + 1 cycles of the 32.768 kHz clock, which makes pulse width from 31.25µs to 500µs possible.

Since the incoming signal is only sampled on positive clock edges, the width of the incoming pulses must be at least two 32.768 kHz clock periods wide for reliable detection by the LEUART receiver. They must also be shorter than half a UART baud period.

At 2400 baud/s or lower, the pulse generator is able to generate RZI pulses compatible with the IrDA physical layer specification. The external IrDA device must generate pulses of sufficient length for successful two-way communication.

18.3.11.1 Interrupts

The interrupts generated by the LEUART are combined into one interrupt vector. If LEUART interrupts are enabled, an interrupt will be made if one or more of the interrupt flags in LEUARTn_IF and their corresponding bits in LEUART_IEN are set.

18.3.12 Register access

Since this module is a Low Energy Peripheral, and runs off a clock which is asynchronous to the HFCORECLK, special considerations must be taken when accessing registers. Please refer to Section 6.3 (p. 61) for a description on how to perform register accesses to Low Energy Peripherals.

18.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	LEUARTn_CTRL	RW	Control Register
0x004	LEUARTn_CMD	W1	Command Register
0x008	LEUARTn_STATUS	R	Status Register
0x00C	LEUARTn_CLKDIV	RW	Clock Control Register
0x010	LEUARTn_STARTFRAME	RW	Start Frame Register
0x014	LEUARTn_SIGFRAME	RW	Signal Frame Register
0x018	LEUARTn_RXDATAX	R	Receive Buffer Data Extended Register
0x01C	LEUARTn_RXDATA	R	Receive Buffer Data Register
0x020	LEUARTn_RXDATAXP	R	Receive Buffer Data Extended Peek Register
0x024	LEUARTn_TXDATAX	W	Transmit Buffer Data Extended Register
0x028	LEUARTn_TXDATA	W	Transmit Buffer Data Register
0x02C	LEUARTn_IF	R	Interrupt Flag Register
0x030	LEUARTn_IFS	W1	Interrupt Flag Set Register
0x034	LEUARTn_IFC	W1	Interrupt Flag Clear Register
0x038	LEUARTn_IEN	RW	Interrupt Enable Register
0x03C	LEUARTn_PULSECTRL	RW	Pulse Control Register
0x040	LEUARTn_FREEZE	RW	Freeze Register
0x044	LEUARTn_SYNCBUSY	R	Synchronization Busy Register
0x054	LEUARTn_ROUTE	RW	I/O Routing Register
0x0AC	LEUARTn_INPUT	RW	LEUART Input Register

18.5 Register Description

18.5.1 LEUARTn_CTRL - Control Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0		0	0	0	0	0	0	0	0	0	0	0x0		0	0
Access																	RW		RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW		RW	RW
Name																	TXDELAY		TXDMAWU	RXDMAWU	BIT8DV	MPAB	MPM	SFUBRX	LOOPBK	ERRSDMA	INV	STOPBITS	PARITY		DATABITS	AUTOTRI

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:14	TXDELAY	0x0	RW	**TX Delay Transmission**

Configurable delay before new transfers. Frames sent back-to-back are not delayed.

Value	Mode	Description
0	NONE	Frames are transmitted immediately
1	SINGLE	Transmission of new frames are delayed by a single baud period
2	DOUBLE	Transmission of new frames are delayed by two baud periods

Bit	Name	Reset	Access	Description

Value	Mode	Description
3	TRIPLE	Transmission of new frames are delayed by three baud periods

13 TXDMAWU 0 RW **TX DMA Wakeup**

Set to wake the DMA controller up when in EM2 and space is available in the transmit buffer.

Value	Description
0	While in EM2, the DMA controller will not get requests about space being available in the transmit buffer
1	DMA is available in EM2 for the request about space available in the transmit buffer

12 RXDMAWU 0 RW **RX DMA Wakeup**

Set to wake the DMA controller up when in EM2 and data is available in the receive buffer.

Value	Description
0	While in EM2, the DMA controller will not get requests about data being available in the receive buffer
1	DMA is available in EM2 for the request about data in the receive buffer

11 BIT8DV 0 RW **Bit 8 Default Value**

When 9-bit frames are transmitted, the default value of the 9th bit is given by BIT8DV. If TXDATA is used to write a frame, then the value of BIT8DV is assigned to the 9th bit of the outgoing frame. If a frame is written with TXDATAX however, the default value is overridden by the written value.

10 MPAB 0 RW **Multi-Processor Address-Bit**

Defines the value of the multi-processor address bit. An incoming frame with its 9th bit equal to the value of this bit marks the frame as a multi-processor address frame.

9 MPM 0 RW **Multi-Processor Mode**

Set to enable multi-processor mode.

Value	Description
0	The 9th bit of incoming frames have no special function
1	An incoming frame with the 9th bit equal to MPAB will be loaded into the receive buffer regardless of RXBLOCK and will result in the MPAB interrupt flag being set

8 SFUBRX 0 RW **Start-Frame UnBlock RX**

Clears RXBLOCK when the start-frame is found in the incoming data. The start-frame is loaded into the receive buffer.

Value	Description
0	Detected start-frames have no effect on RXBLOCK
1	When a start-frame is detected, RXBLOCK is cleared and the start-frame is loaded into the receive buffer

7 LOOPBK 0 RW **Loopback Enable**

Set to connect receiver to LEUn_TX instead of LEUn_RX.

Value	Description
0	The receiver is connected to and receives data from LEUn_RX
1	The receiver is connected to and receives data from LEUn_TX

6 ERRSDMA 0 RW **Clear RX DMA On Error**

When set,RX DMA requests will be cleared on framing and parity errors.

Value	Description
0	Framing and parity errors have no effect on DMA requests from the LEUART
1	RX DMA requests from the LEUART are disabled if a framing error or parity error occurs.

5 INV 0 RW **Invert Input And Output**

Set to invert the output on LEUn_TX and input on LEUn_RX.

Value	Description
0	A high value on the input/output is 1, and a low value is 0.
1	A low value on the input/output is 1, and a high value is 0.

4 STOPBITS 0 RW **Stop-Bit Mode**

Determines the number of stop-bits used. Only used when transmitting data. The receiver only verifies that one stop bit is present.

Bit	Name	Reset	Access	Description

Value	Mode	Description
0	ONE	One stop-bit is transmitted with every frame
1	TWO	Two stop-bits are transmitted with every frame

3:2 PARITY 0x0 RW **Parity-Bit Mode**

Determines whether parity bits are enabled, and whether even or odd parity should be used.

Value	Mode	Description
0	NONE	Parity bits are not used
2	EVEN	Even parity are used. Parity bits are automatically generated and checked by hardware.
3	ODD	Odd parity is used. Parity bits are automatically generated and checked by hardware.

1 DATABITS 0 RW **Data-Bit Mode**

This register sets the number of data bits.

Value	Mode	Description
0	EIGHT	Each frame contains 8 data bits
1	NINE	Each frame contains 9 data bits

0 AUTOTRI 0 RW **Automatic Transmitter Tristate**

When set, LEUn_TX is tristated whenever the transmitter is inactive.

Value	Description
0	LEUn_TX is held high when the transmitter is inactive. INV inverts the inactive state.
1	LEUn_TX is tristated when the transmitter is inactive

18.5.2 LEUARTn_CMD - Command Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																													
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0	0	0	0	0	0	0	0
Access																									W1	W1	W1	W1	W1	W1	W1	W1
Name																									CLEARRX	CLEARTX	RXBLOCKDIS	RXBLOCKEN	TXDIS	TXEN	RXDIS	RXEN

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7	CLEARRX	0	W1	**Clear RX**
	Set to clear receive buffer and the RX shift register.			
6	CLEARTX	0	W1	**Clear TX**
	Set to clear transmit buffer and the TX shift register.			
5	RXBLOCKDIS	0	W1	**Receiver Block Disable**
	Set to clear RXBLOCK, resulting in all incoming frames being loaded into the receive buffer.			
4	RXBLOCKEN	0	W1	**Receiver Block Enable**
	Set to set RXBLOCK, resulting in all incoming frames being discarded.			
3	TXDIS	0	W1	**Transmitter Disable**
	Set to disable transmission.			
2	TXEN	0	W1	**Transmitter Enable**
	Set to enable data transmission.			

Bit	Name	Reset	Access	Description
1	RXDIS	0	W1	**Receiver Disable**
	Set to disable data reception. If a frame is under reception when the receiver is disabled, the incoming frame is discarded.			
0	RXEN	0	W1	**Receiver Enable**
	Set to activate data reception on LEUn_RX.			

18.5.3 LEUARTn_STATUS - Status Register

Offset	Bit Position																														
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	1	0	0	0	0
Access																											R	R	R	R	R	R
Name																											RXDATAV	TXBL	TXC	RXBLOCK	TXENS	RXENS

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	RXDATAV	0	R	**RX Data Valid**
	Set when data is available in the receive buffer. Cleared when the receive buffer is empty.			
4	TXBL	1	R	**TX Buffer Level**
	Indicates the level of the transmit buffer. Set when the transmit buffer is empty, and cleared when it is full.			
3	TXC	0	R	**TX Complete**
	Set when a transmission has completed and no more data is available in the transmit buffer. Cleared when a new transmission starts.			
2	RXBLOCK	0	R	**Block Incoming Data**
	When set, the receiver discards incoming frames. An incoming frame will not be loaded into the receive buffer if this bit is set at the instant the frame has been completely received.			
1	TXENS	0	R	**Transmitter Enable Status**
	Set when the transmitter is enabled.			
0	RXENS	0	R	**Receiver Enable Status**
	Set when the receiver is enabled. The receiver must be enabled for start frames, signal frames, and multi-processor address bit detection.			

18.5.4 LEUARTn_CLKDIV - Clock Control Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																														
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																		0x000														
Access																		RW														
Name																		DIV														

Bit	Name	Reset	Access	Description
31:15	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
14:3	DIV	0x000	RW	**Fractional Clock Divider**
	Specifies the fractional clock divider for the LEUART.			
2:0	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

18.5.5 LEUARTn_STARTFRAME - Start Frame Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																															
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																								0x000								
Access																								RW								
Name																								STARTFRAME								

Bit	Name	Reset	Access	Description
31:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:0	STARTFRAME	0x000	RW	**Start Frame**
	When a frame matching STARTFRAME is detected by the receiver, STARTF interrupt flag is set, and if SFUBRX is set, RXBLOCK is cleared. The start-frame is be loaded into the RX buffer.			

18.5.6 LEUARTn_SIGFRAME - Signal Frame Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																															
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																								0x000								
Access																								RW								
Name																								SIGFRAME								

Bit	Name	Reset	Access	Description
31:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:0	SIGFRAME	0x000	RW	**Signal Frame**
	When a frame matching SIGFRAME is detected by the receiver, SIGF interrupt flag is set.			

18.5.7 LEUARTn_RXDATAX - Receive Buffer Data Extended Register

Offset	Bit Position																													
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0	0						0x000								
Access																	R	R						R								
Name																	FERR	PERR						RXDATA								

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15	FERR	0	R	**Receive Data Framing Error**
	Set if data in buffer has a framing error. Can be the result of a break condition.			
14	PERR	0	R	**Receive Data Parity Error**
	Set if data in buffer has a parity error.			
13:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:0	RXDATA	0x000	R	**RX Data**
	Use this register to access data read from the LEUART. Buffer is cleared on read access.			

18.5.8 LEUARTn_RXDATA - Receive Buffer Data Register

Offset	Bit Position																													
0x01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0x00						
Access																									R						
Name																									RXDATA						

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:0	RXDATA	0x00	R	**RX Data**
	Use this register to access data read from LEUART. Buffer is cleared on read access. Only the 8 LSB can be read using this register.			

18.5.9 LEUARTn_RXDATAXP - Receive Buffer Data Extended Peek Register

Offset	Bit Position																															
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0	0						0x000								
Access																	R	R						R								
Name																	FERRP	PERRP						RXDATAP								

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15	FERRP	0	R	**Receive Data Framing Error Peek**
	Set if data in buffer has a framing error. Can be the result of a break condition.			
14	PERRP	0	R	**Receive Data Parity Error Peek**
	Set if data in buffer has a parity error.			
13:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:0	RXDATAP	0x000	R	**RX Data Peek**
	Use this register to access data read from the LEUART.			

18.5.10 LEUARTn_TXDATAX - Transmit Buffer Data Extended Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																															
0x024	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0	0	0					0x000								
Access																	W	W	W					W								
Name																	RXENAT	TXDISAT	TXBREAK					TXDATA								

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15	RXENAT	0	W	**Enable RX After Transmission**
	Set to enable reception after transmission.			

Value	Description
0	-
1	The receiver is enabled, setting RXENS after the frame has been transmitted

14	TXDISAT	0	W	**Disable TX After Transmission**
	Set to disable transmitter directly after transmission has competed.			

Value	Description
0	-

Bit	Name	Reset	Access	Description

Value	Description
1	The transmitter is disabled, clearing TXENS after the frame has been transmitted

Bit	Name	Reset	Access	Description
13	TXBREAK	0	W	**Transmit Data As Break**

Set to send data as a break. Recipient will see a framing error or a break condition depending on its configuration and the value of TXDATA.

Value	Description
0	The specified number of stop-bits are transmitted
1	Instead of the ordinary stop-bits, 0 is transmitted to generate a break. A single stop-bit is generated after the break to allow the receiver to detect the start of the next frame

Bit	Name	Reset	Access	Description
12:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:0	TXDATA	0x000	W	**TX Data**

Use this register to write data to the LEUART. If the transmitter is enabled, a transfer will be initiated at the first opportunity.

18.5.11 LEUARTn_TXDATA - Transmit Buffer Data Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																													
0x028	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0x00							
Access																									W							
Name																									TXDATA							

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7:0	TXDATA	0x00	W	**TX Data**

This frame will be added to the transmit buffer. Only 8 LSB can be written using this register. 9th bit and control bits will be cleared.

18.5.12 LEUARTn_IF - Interrupt Flag Register

Offset	Bit Position																													
0x02C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																						0	0	0	0	0	0	0	0	0	1	0
Access																						R	R	R	R	R	R	R	R	R	R	R
Name																						SIGF	STARTF	MPAF	FERR	PERR	TXOF	RXUF	RXOF	RXDATAV	TXBL	TXC

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	SIGF	0	R	**Signal Frame Interrupt Flag**

Set when a signal frame is detected.

Bit	Name	Reset	Access	Description
9	STARTF	0	R	**Start Frame Interrupt Flag**

Set when a start frame is detected.

Bit	Name	Reset	Access	Description
8	MPAF	0	R	**Multi-Processor Address Frame Interrupt Flag**

Bit	Name	Reset	Access	Description
	Set when a multi-processor address frame is detected.			
7	FERR	0	R	**Framing Error Interrupt Flag**
	Set when a frame with a framing error is received while RXBLOCK is cleared.			
6	PERR	0	R	**Parity Error Interrupt Flag**
	Set when a frame with a parity error is received while RXBLOCK is cleared.			
5	TXOF	0	R	**TX Overflow Interrupt Flag**
	Set when a write is done to the transmit buffer while it is full. The data already in the transmit buffer is preserved.			
4	RXUF	0	R	**RX Underflow Interrupt Flag**
	Set when trying to read from the receive buffer when it is empty.			
3	RXOF	0	R	**RX Overflow Interrupt Flag**
	Set when data is incoming while the receive shift register is full. The data previously in shift register is overwritten by the new data.			
2	RXDATAV	0	R	**RX Data Valid Interrupt Flag**
	Set when data becomes available in the receive buffer.			
1	TXBL	1	R	**TX Buffer Level Interrupt Flag**
	Set when space becomes available in the transmit buffer for a new frame.			
0	TXC	0	R	**TX Complete Interrupt Flag**
	Set after a transmission when both the TX buffer and shift register are empty.			

18.5.13 LEUARTn_IFS - Interrupt Flag Set Register

Offset	Bit Position																											
0x030	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																						0	0	0	0	0	0	0	0			0
Access																						W1	W1	W1	W1	W1	W1	W1	W1			W1
Name																						SIGF	STARTF	MPAF	FERR	PERR	TXOF	RXUF	RXOF			TXC

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	SIGF	0	W1	**Set Signal Frame Interrupt Flag**
	Write to 1 to set the SIGF interrupt flag.			
9	STARTF	0	W1	**Set Start Frame Interrupt Flag**
	Write to 1 to set the STARTF interrupt flag.			
8	MPAF	0	W1	**Set Multi-Processor Address Frame Interrupt Flag**
	Write to 1 to set the MPAF interrupt flag.			
7	FERR	0	W1	**Set Framing Error Interrupt Flag**
	Write to 1 to set the FERR interrupt flag.			
6	PERR	0	W1	**Set Parity Error Interrupt Flag**
	Write to 1 to set the PERR interrupt flag.			
5	TXOF	0	W1	**Set TX Overflow Interrupt Flag**
	Write to 1 to set the TXOF interrupt flag.			
4	RXUF	0	W1	**Set RX Underflow Interrupt Flag**
	Write to 1 to set the RXUF interrupt flag.			
3	RXOF	0	W1	**Set RX Overflow Interrupt Flag**

Bit	Name	Reset	Access	Description
	Write to 1 to set the RXOF interrupt flag.			
2:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	TXC	0	W1	**Set TX Complete Interrupt Flag**
	Write to 1 to set the TXC interrupt flag.			

18.5.14 LEUARTn_IFC - Interrupt Flag Clear Register

Offset	Bit Position																												
0x034	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																						0	0	0	0	0	0	0	0			0
Access																						W1	W1	W1	W1	W1	W1	W1	W1			W1
Name																						SIGF	STARTF	MPAF	FERR	PERR	TXOF	RXUF	RXOF			TXC

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	SIGF	0	W1	**Clear Signal-Frame Interrupt Flag**
	Write to 1 to clear the SIGF interrupt flag.			
9	STARTF	0	W1	**Clear Start-Frame Interrupt Flag**
	Write to 1 to clear the STARTF interrupt flag.			
8	MPAF	0	W1	**Clear Multi-Processor Address Frame Interrupt Flag**
	Write to 1 to clear the MPAF interrupt flag.			
7	FERR	0	W1	**Clear Framing Error Interrupt Flag**
	Write to 1 to clear the FERR interrupt flag.			
6	PERR	0	W1	**Clear Parity Error Interrupt Flag**
	Write to 1 to clear the PERR interrupt flag.			
5	TXOF	0	W1	**Clear TX Overflow Interrupt Flag**
	Write to 1 to clear the TXOF interrupt flag.			
4	RXUF	0	W1	**Clear RX Underflow Interrupt Flag**
	Write to 1 to clear the RXUF interrupt flag.			
3	RXOF	0	W1	**Clear RX Overflow Interrupt Flag**
	Write to 1 to clear the RXOF interrupt flag.			
2:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	TXC	0	W1	**Clear TX Complete Interrupt Flag**
	Write to 1 to clear the TXC interrupt flag.			

18.5.15 LEUARTn_IEN - Interrupt Enable Register

Offset	Bit Position																													
0x038	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																						0	0	0	0	0	0	0	0	0	0	0
Access																						RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Name																						SIGF	STARTF	MPAF	FERR	PERR	TXOF	RXUF	RXOF	RXDATAV	TXBL	TXC

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	SIGF	0	RW	**Signal Frame Interrupt Enable**
	Enable interrupt on signal frame.			
9	STARTF	0	RW	**Start Frame Interrupt Enable**
	Enable interrupt on start frame.			
8	MPAF	0	RW	**Multi-Processor Address Frame Interrupt Enable**
	Enable interrupt on multi-processor address frame.			
7	FERR	0	RW	**Framing Error Interrupt Enable**
	Enable interrupt on framing error.			
6	PERR	0	RW	**Parity Error Interrupt Enable**
	Enable interrupt on parity error.			
5	TXOF	0	RW	**TX Overflow Interrupt Enable**
	Enable interrupt on TX overflow.			
4	RXUF	0	RW	**RX Underflow Interrupt Enable**
	Enable interrupt on RX underflow.			
3	RXOF	0	RW	**RX Overflow Interrupt Enable**
	Enable interrupt on RX overflow.			
2	RXDATAV	0	RW	**RX Data Valid Interrupt Enable**
	Enable interrupt on RX data.			
1	TXBL	0	RW	**TX Buffer Level Interrupt Enable**
	Enable interrupt on TX buffer level.			
0	TXC	0	RW	**TX Complete Interrupt Enable**
	Enable interrupt on TX complete.			

18.5.16 LEUARTn_PULSECTRL - Pulse Control Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																													
0x03C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0x0			
Access																											RW	RW	RW			
Name																											PULSEFILT	PULSEEN	PULSEW			

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	PULSEFILT	0	RW	**Pulse Filter**

Enable a one-cycle pulse filter for pulse extender

Value	Description
0	Filter is disabled. Pulses must be at least 2 cycles long for reliable detection.
1	Filter is enabled. Pulses must be at least 3 cycles long for reliable detection.

Bit	Name	Reset	Access	Description
4	PULSEEN	0	RW	**Pulse Generator/Extender Enable**

Filter LEUART output through pulse generator and the LEUART input through the pulse extender.

Bit	Name	Reset	Access	Description
3:0	PULSEW	0x0	RW	**Pulse Width**

Configure the pulse width of the pulse generator as a number of 32.768 kHz clock cycles.

18.5.17 LEUARTn_FREEZE - Freeze Register

Offset	Bit Position																													
0x040	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																RW
Name																																REGFREEZE

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	REGFREEZE	0	RW	**Register Update Freeze**

When set, the update of the LEUART is postponed until this bit is cleared. Use this bit to update several registers simultaneously.

Value	Mode	Description
0	UPDATE	Each write access to a LEUART register is updated into the Low Frequency domain as soon as possible.
1	FREEZE	The LEUART is not updated with the new written value.

18.5.18 LEUARTn_SYNCBUSY - Synchronization Busy Register

Offset	Bit Position																													
0x044	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																									0	0	0	0	0	0	0	0
Access																									R	R	R	R	R	R	R	R
Name																									PULSECTRL	TXDATA	TXDATAX	SIGFRAME	STARTFRAME	CLKDIV	CMD	CTRL

Bit	Name	Reset	Access	Description
31:8	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
7	PULSECTRL	0	R	**PULSECTRL Register Busy**

Set when the value written to PULSECTRL is being synchronized.

Bit	Name	Reset	Access	Description
6	TXDATA	0	R	**TXDATA Register Busy**

Bit	Name	Reset	Access	Description
	Set when the value written to TXDATA is being synchronized.			
5	TXDATAX	0	R	**TXDATAX Register Busy**
	Set when the value written to TXDATAX is being synchronized.			
4	SIGFRAME	0	R	**SIGFRAME Register Busy**
	Set when the value written to SIGFRAME is being synchronized.			
3	STARTFRAME	0	R	**STARTFRAME Register Busy**
	Set when the value written to STARTFRAME is being synchronized.			
2	CLKDIV	0	R	**CLKDIV Register Busy**
	Set when the value written to CLKDIV is being synchronized.			
1	CMD	0	R	**CMD Register Busy**
	Set when the value written to CMD is being synchronized.			
0	CTRL	0	R	**CTRL Register Busy**
	Set when the value written to CTRL is being synchronized.			

18.5.19 LEUARTn_ROUTE - I/O Routing Register

Offset	Bit Position																												
0x054	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0x0								0	0
Access																							RW								RW	RW
Name																							LOCATION								TXPEN	RXPEN

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	LOCATION	0x0	RW	**I/O Location**
	Decides the location of the LEUART I/O pins.			

Value	Mode	Description
0	LOC0	Location 0
1	LOC1	Location 1
2	LOC2	Location 2
3	LOC3	Location 3
4	LOC4	Location 4
5	LOC5	Location 5

7:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	TXPEN	0	RW	**TX Pin Enable**
	When set, the TX pin of the LEUART is enabled.			

Value	Description
0	The LEUn_TX pin is disabled
1	The LEUn_TX pin is enabled

0	RXPEN	0	RW	**RX Pin Enable**
	When set, the RX pin of the LEUART is enabled.			

Value	Description
0	The LEUn_RX pin is disabled
1	The LEUn_RX pin is enabled

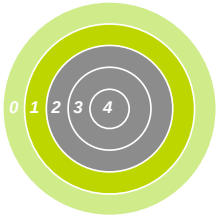
18.5.20 LEUARTn_INPUT - LEUART Input Register

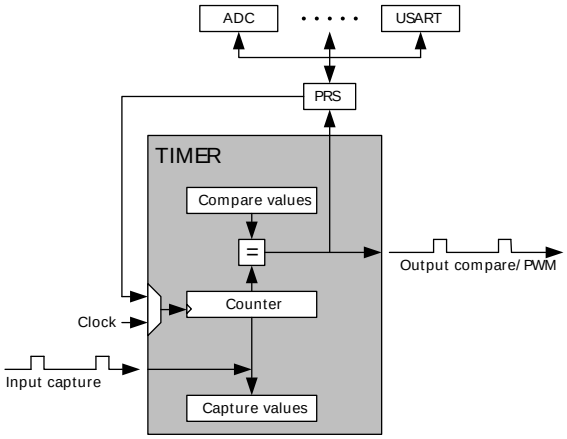
Offset	Bit Position																													
0x0AC	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																												0			0x0	
Access																												RW			RW	
Name																												RXPRS			RXPRSSEL	

Bit	Name	Reset	Access	Description
31:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	RXPRS	0	RW	**PRS RX Enable**
	When set, the PRS channel selected as input to RX.			
3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2:0	RXPRSSEL	0x0	RW	**RX PRS Channel Select**
	Select PRS channel as input to RX.			

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected
1	PRSCH1	PRS Channel 1 selected
2	PRSCH2	PRS Channel 2 selected
3	PRSCH3	PRS Channel 3 selected
4	PRSCH4	PRS Channel 4 selected
5	PRSCH5	PRS Channel 5 selected

19 TIMER - Timer/Counter





Quick Facts

What?

The TIMER (Timer/Counter) keeps track of timing and counts events, generates output waveforms and triggers timed actions in other peripherals.

Why?

Most applications have activities that need to be timed accurately with as little CPU intervention and energy consumption as possible.

How?

The flexible 16-bit TIMER can be configured to provide PWM waveforms with optional dead-time insertion for e.g. motor control, or work as a frequency generator. The Timer can also count events and control other peripherals through the PRS, which offloads the CPU and reduce energy consumption.

19.1 Introduction

The 16-bit general purpose Timer has 3 compare/capture channels for input capture and compare/Pulse-Width Modulation (PWM) output. TIMER0 also includes a Dead-Time Insertion module suitable for motor control applications.

19.2 Features

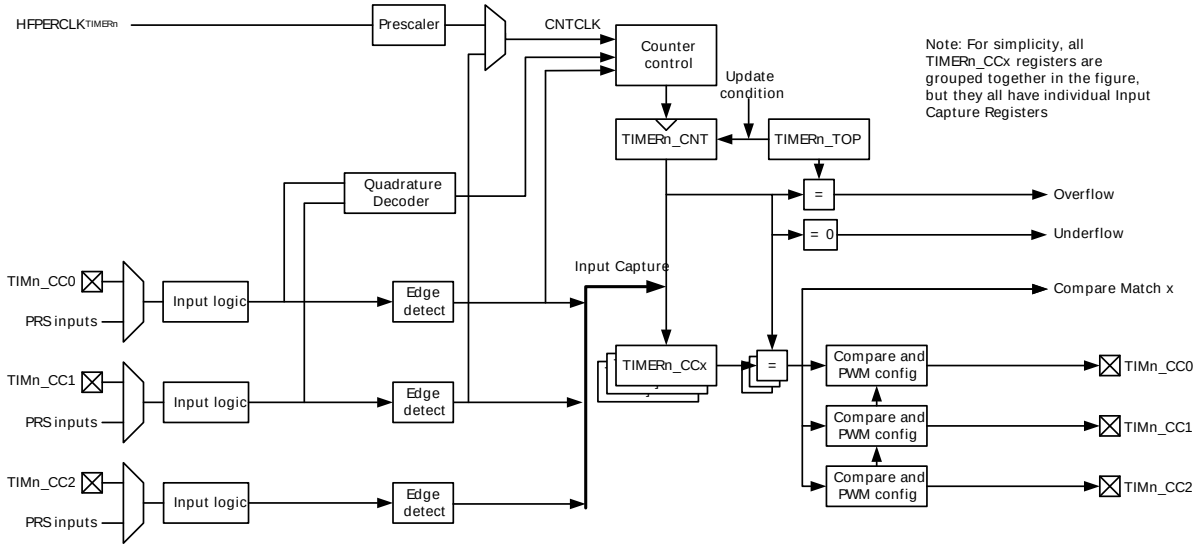
- 16-bit auto reload up/down counter
  - Dedicated 16-bit reload register which serves as counter maximum
- 3 Compare/Capture channels
  - Individual configurable as either input capture or output compare/PWM
- Multiple Counter modes
  - Count up
  - Count down
  - Count up/down
  - Quadrature Decoder
  - Direction and count from external pins
- 2x Count Mode
- Counter control from PRS or external pin
  - Start
  - Stop
  - Reload and start
- Inter-Timer connection
  - Allows 32-bit counter mode
  - Start/stop synchronization between several Timers
- Input Capture

- Period measurement
- Pulse width measurement
- Two capture registers for each capture channel
  - Capture on either positive or negative edge
  - Capture on both edges
- Optional digital noise filtering on capture inputs
- Output Compare
  - Compare output toggle/pulse on compare match
  - Immediate update of compare registers
- PWM
  - Up-count PWM
  - Up/down-count PWM
  - Predictable initial PWM output state (configured by SW)
  - Buffered compare register to ensure glitch-free update of compare values
- Clock sources
  - $HFPERCLK_{TIMERn}$
    - 10-bit Prescaler
  - External pin
  - Peripheral Reflex System
- Debug mode
  - Configurable to either run or stop when processor is stopped (break)
- Interrupts, PRS output and/or DMA request
  - Underflow
  - Overflow
  - Compare/Capture event
- Dead-Time Insertion Unit (TIMER0 only)
  - Complementary PWM outputs with programmable dead-time
    - Dead-time is specified independently for rising and falling edge
      - 10-bit prescaler
      - 6-bit time value
    - Outputs have configurable polarity
    - Outputs can be set inactive individually by software.
  - Configurable action on fault
    - Set outputs inactive
    - Clear output
    - Tristate output
  - Individual fault sources
    - One or two PRS signals
    - Debugger
      - Support for automatic restart
    - Core lockup
  - Configuration lock

19.3 Functional Description

An overview of the TIMER module is shown in Figure 19.1 (p. 443) . The Timer module consists of a 16 bit up/down counter with 3 Compare/Capture channels connected to pins TIMn_CC0, TIMn_CC1, and TIMn_CC2.

Figure 19.1. TIMER Block Overview



19.3.1 Counter Modes

The Timer consists of a counter that can be configured to the following modes:

1. Up-count: Counter counts up until it reaches the value in TIMERn_TOP, where it is reset to 0 before counting up again.
2. Down-count: The counter starts at the value in TIMERn_TOP and counts down. When it reaches 0, it is reloaded with the value in TIMERn_TOP.
3. Up/Down-count: The counter starts at 0 and counts up. When it reaches the value in TIMERn_TOP, it counts down until it reaches 0 and starts counting up again.
4. Quadrature Decoder: Two input channels where one determines the count direction, while the other pin triggers a clock event.

In addition, to the TIMER modes listed above, the TIMER also supports a 2x Count Mode. In this mode the counter increments/decrements by 2. The 2x Count Mode intended use is to generate 2x PWM frequency when the Compare/Capture channel is put in PWM mode. The 2x Count Mode can be enabled by setting the X2CNT bitfield in the TIMERn_CTRL register.

The counter value can be read or written by software at any time by accessing the CNT field in TIMERn_CNT.

19.3.1.1 Events

Overflow is set when the counter value shifts from TIMERn_TOP to the next value when counting up. In up-count mode the next value is 0. In up/down-count mode, the next value is TIMERn_TOP-1.

Underflow is set when the counter value shifts from 0 to the next value when counting down. In down-count mode, the next value is TIMERn_TOP. In up/down-count mode the next value is 1.

Update event is set on overflow in up-count mode and on underflow in down-count or up/down count mode. This event is used to time updates of buffered values.

19.3.1.2 Operation

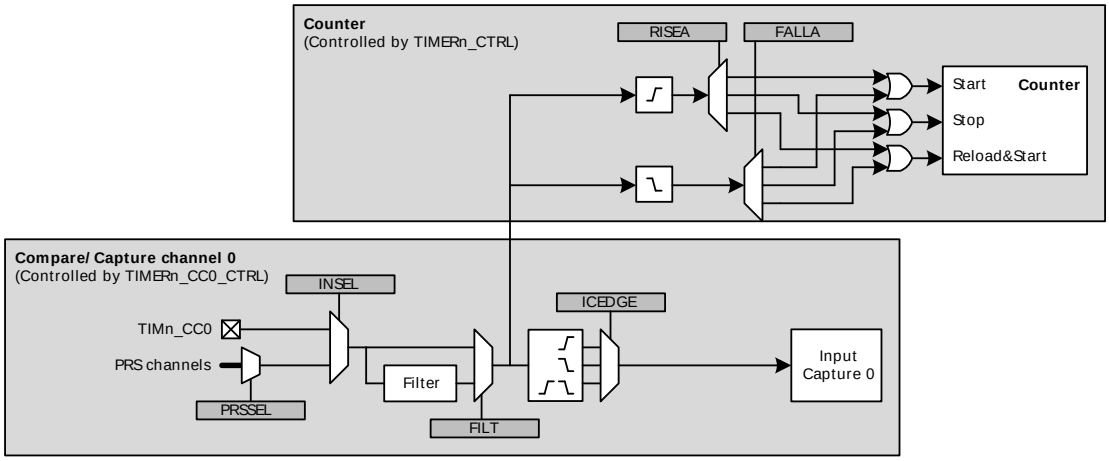
Figure 19.2 (p. 444) shows the hardware Timer/Counter control. Software can start or stop the counter by writing a 1 to the START or STOP bits in TIMERn_CMD. The counter value (CNT in TIMERn_CNT) can always be written by software to any 16-bit value.

It is also possible to control the counter through either an external pin or PRS input. This is done through the input logic for the Compare/Capture Channel 0. The Timer/Counter allows individual actions (start, stop, reload) to be taken for rising and falling input edges. This is configured in the RISEA and FALLA fields in TIMERn_CTRL. The reload value is 0 in up-count and up/down-count mode and TOP in down-count mode.

The RUNNING bit in TIMERn_STATUS indicates if the Timer is running or not. If the SYNC bit in TIMERn_CTRL is set, the Timer is started/stopped/reloaded (external pin or PRS) when any of the other timers are started/stopped/reloaded.

The DIR bit in TIMERn_STATUS indicates the counting direction of the Timer at any given time. The counter value can be read or written by software through the CNT field in TIMERn_CNT. In Up/Down-Count mode the count direction will be set to up if the CNT value is written by software.

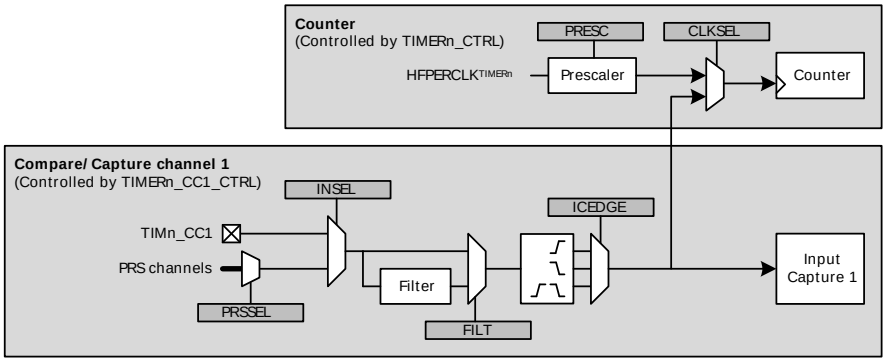
Figure 19.2. TIMER Hardware Timer/Counter Control



19.3.1.3 Clock Source

The counter can be clocked from several sources, which are all synchronized with the peripheral clock (HFPERCLK). See Figure 19.3 (p. 444) .

Figure 19.3. TIMER Clock Selection



19.3.1.3.1 Peripheral Clock (HFPERCLK)

The peripheral clock (HFPERCLK) can be used as a source with a configurable prescale factor of 2^PRESC, where PRESC is an integer between 0 and 10, which is set in PRESC in TIMERn_CTRL.

However, if 2x Count Mode is enabled and the Compare/Capture channels are put in PWM mode, the CC output is updated on both clock edges so prescaling the peripheral clock will result in incorrect result. The prescaler is stopped and reset when the timer is stopped.

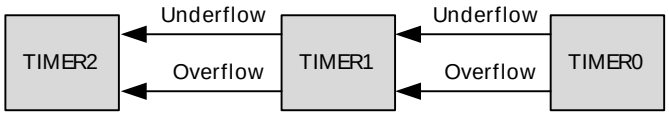
19.3.1.3.2 Compare/ Capture Channel 1 Input

The Timer can also be clocked by positive and/or negative edges on the Compare/Capture channel 1 input. This input can either come from the TIMn_CC1 pin or one of the PRS channels. The input signal must not have a higher frequency than $f_{HFPERCLK}/3$ when running from a pin input or a PRS input with FILT enabled in TIMERn_CCx_CTRL. When running from PRS without FILT, the frequency can be as high as $f_{HFPERCLK}$. Note that when clocking the Timer from the same pulse that triggers a start (through RISEA/FALLA in TIMERn_CTRL), the starting pulse will not update the Counter Value.

19.3.1.3.3 Underflow/Overflow from Neighboring Timer

All Timers are linked together (see Figure 19.4 (p. 445)), allowing timers to count on overflow/underflow from the lower numbered neighbouring timers to form a 32-bit or 48-bit timer. Note that all timers must be set to same count direction and less significant timer(s) can only be set to count up or down.

Figure 19.4. TIMER Connections



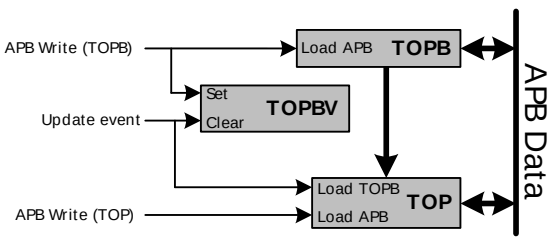
19.3.1.4 One-Shot Mode

By default, the counter counts continuously until it is stopped. If the OSMEN bit is set in the TIMERn_CTRL register, however, the counter is disabled by hardware on the first *update event*. Note that when the counter is running with CC1 as clock source (0b01 in CLKSEL in TIMERn_CTRL) and OSMEN is set, a CC1 capture event will not take place on the *update event* (CC1 rising edge) that stops the Timer.

19.3.1.5 Top Value Buffer

The TIMERn_TOP register can be altered either by writing it directly or by writing to the TIMER_TOPB (buffer) register. When writing to the buffer register the TIMERn_TOPB register will be written to TIMERn_TOP on the next update event. Buffering ensures that the TOP value is not set below the actual count value. The TOPBV flag in TIMERn_STATUS indicates whether the TIMERn_TOPB register contains data that have not yet been written to the TIMERn_TOP register (see Figure 19.5 (p. 445) .

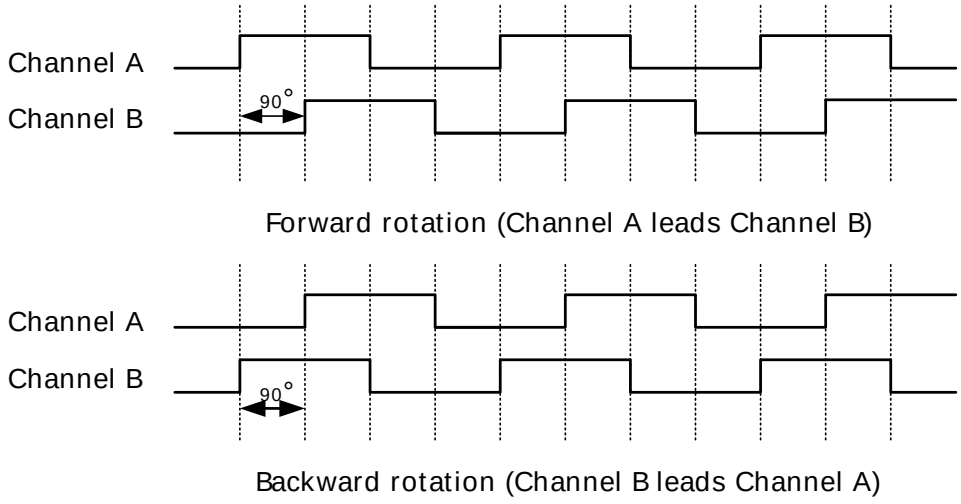
Figure 19.5. TIMER TOP Value Update Functionality



19.3.1.6 Quadrature Decoder

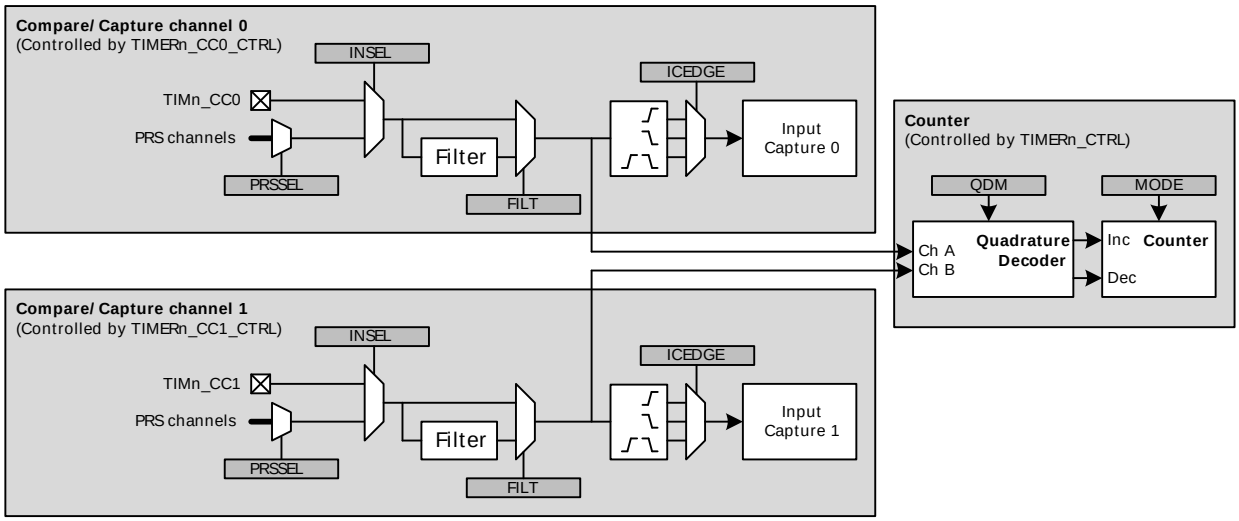
Quadrature Decoding mode is used to track motion and determine both rotation direction and position. The Quadrature Decoder uses two input channels that are 90 degrees out of phase (see Figure 19.6 (p. 446)).

Figure 19.6. TIMER Quadrature Encoded Inputs



In the Timer these inputs are tapped from the Compare/Capture channel 0 (Channel A) and 1 (Channel B) inputs before edge detection. The Timer/Counter then increments or decrements the counter, based on the phase relation between the two inputs. The Quadrature Decoder Mode supports two channels, but if a third channel (Z-terminal) is available, this can be connected to an external interrupt and trigger a counter reset from the interrupt service routine. By connecting a periodic signal from another timer as input capture on Compare/Capture Channel 2, it is also possible to calculate speed and acceleration.

Figure 19.7. TIMER Quadrature Decoder Configuration



The Quadrature Decoder can be set in either X2 or X4 mode, which is configured in the QDM bit in TIMERn_CTRL. See Figure 19.7 (p. 446)

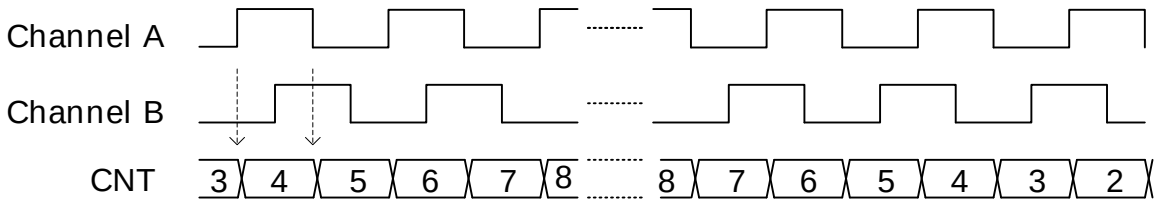
19.3.1.6.1 X2 Decoding Mode

In X2 Decoding mode, the counter increments or decrements on every edge of Channel A, see Table 19.1 (p. 447) and Figure 19.8 (p. 447) .

Table 19.1. TIMER Counter Response in X2 Decoding Mode

Channel B	Channel A	
	Rising	Falling
0	Increment	Decrement
1	Decrement	Increment

Figure 19.8. TIMER X2 Decoding Mode



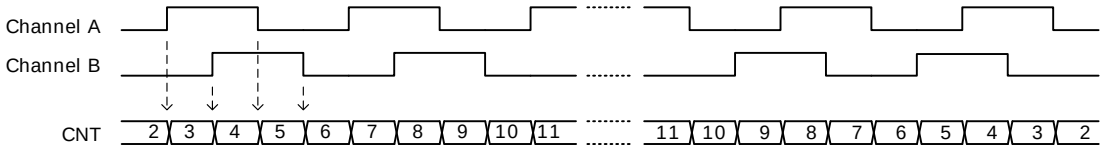
19.3.1.6.2 X4 Decoding Mode

In X4 Decoding mode, the counter increments or decrements on every edge of Channel A and Channel B, see Figure 19.9 (p. 447) and Table 19.2 (p. 447) .

Table 19.2. TIMER Counter Response in X4 Decoding Mode

Opposite Channel	Channel A		Channel B	
	Rising	Falling	Rising	Falling
Channel A = 0			Decrement	Increment
Channel A = 1			Increment	Decrement
Channel B = 0	Increment	Decrement		
Channel B = 1	Decrement	Increment		

Figure 19.9. TIMER X4 Decoding Mode



19.3.1.6.3 TIMER Rotational Position

To calculate a position Equation 19.1 (p. 447) can be used.

TIMER Rotational Position Equation

$$pos° = (CNT/X \times N) \times 360° \quad (19.1)$$

where X = Encoding type and N = Number of pulses per revolution.

19.3.2 Compare/Capture Channels

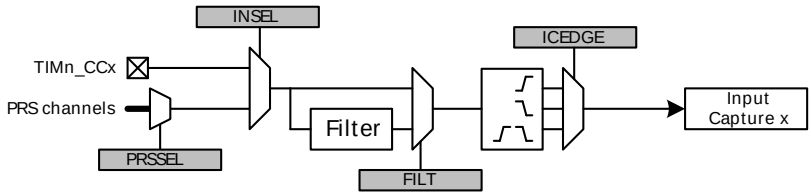
The Timer contains 3 Compare/Capture channels, which can be configured in the following modes:

1. Input Capture
2. Output Compare
3. PWM

19.3.2.1 Input Pin Logic

Each Compare/Capture channel can be configured as an input source for the Capture Unit or as external clock source for the Timer (see Figure 19.10 (p. 448)). Compare/Capture channels 0 and 1 are the inputs for the Quadrature Decoder Mode. The input channel can be filtered before it is used, which requires the input to remain stable for 5 cycles in a row before the input is propagated to the output.

Figure 19.10. TIMER Input Pin Logic



19.3.2.2 Compare/Capture Registers

The Compare/Capture channel registers are prefixed with TIMERn_CCx_, where the x stands for the channel number. Since the Compare/Capture channels serve three functions (input capture, compare, PWM), the behavior of the Compare/Capture registers (TIMERn_CCx_CCV) and buffer registers (TIMERn_CCx_CCVB) change depending on the mode the channel is set in.

19.3.2.2.1 Input Capture mode

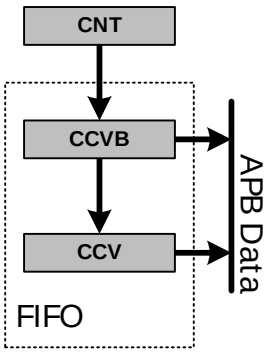
When running in Input Capture mode, TIMERn_CCx_CCV and TIMERn_CCx_CCVB form a FIFO buffer, and new capture values are added on a capture event, see Figure 19.11 (p. 449) . The first capture can always be read from TIMERn_CCx_CCV, and reading this address will load the next capture value into TIMERn_CCx_CCV from TIMERn_CCx_CCVB if it contains valid data. The CC value can be read without altering the FIFO contents by reading TIMERn_CCx_CCVP. TIMERn_CCx_CCVB can also be read without altering the FIFO contents. The ICV flag in TIMERn_STATUS indicates if there is a valid unread capture in TIMERn_CCx_CCV.

In case a capture is triggered while both CCV and CCVB contain unread capture values, the buffer overflow interrupt flag (ICBOF in TIMERn_IF) will be set. New capture values will on overflow overwrite the value in TIMERn_CCx_CCVB.

Note

In input capture mode, the timer will only trigger interrupts when it is running

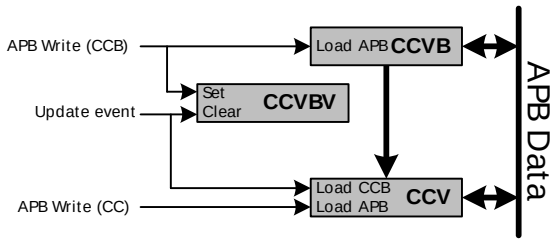
Figure 19.11. TIMER Input Capture Buffer Functionality



19.3.2.2.2 Compare and PWM Mode

When running in Output Compare or PWM mode, the value in TIMERn_CCx_CCV will be compared against the count value. In Compare mode the output can be configured to toggle, clear or set on compare match, overflow and underflow through the CMOA, COFOA and CUFOA fields in TIMERn_CCx_CTRL. TIMERn_CCx_CCV can be accessed directly or through the buffer register TIMERn_CCx_CCVB, see Figure 19.12 (p. 449) . When writing to the buffer register, the value in TIMERn_CCx_CCVB will be written to TIMERn_CCx_CCV on the next update event. This functionality ensures glitch free PWM outputs. The CCVBV flag in TIMERn_STATUS indicates whether the TIMERn_CCx_CCVB register contains data that have not yet been written to the TIMERn_CCx_CCV register. Note that when writing 0 to TIMERn_CCx_CCVB the CCV value is updated when the timer counts from 0 to 1. Thus, the compare match for the next period will not happen until the timer reaches 0 again on the way down.

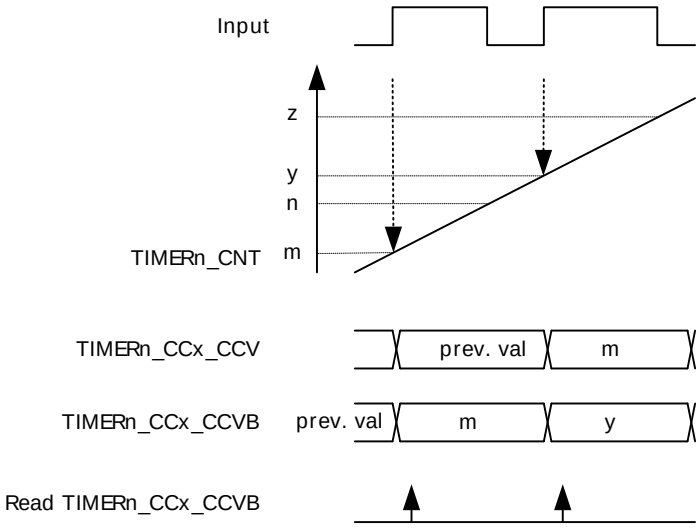
Figure 19.12. TIMER Output Compare/PWM Buffer Functionality



19.3.2.3 Input Capture

In Input Capture Mode, the counter value (TIMERn_CNT) can be captured in the Compare/Capture Register (TIMERn_CCx_CCV), see Figure 19.13 (p. 450) . In this mode, TIMERn_CCx_CCV is read-only. Together with the Compare/Capture Buffer Register (TIMERn_CCx_CCVB) the TIMERn_CCx_CCV form a double-buffered capture registers allowing two subsequent capture events to take place before a read-out is required. The CCPOL bits in TIMERn_STATUS indicate the polarity the edge that triggered the capture in TIMERn_CCx_CCV.

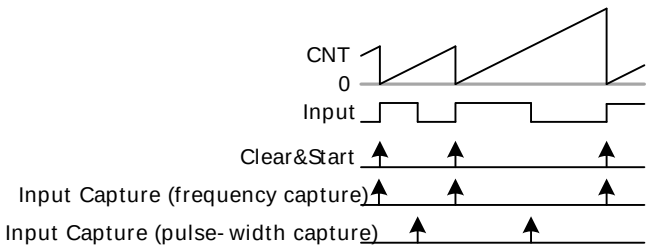
Figure 19.13. TIMER Input Capture



19.3.2.3.1 Period/Pulse-Width Capture

Period and/or pulse-width capture can be achieved by setting the RISEA field in TIMERn_CTRL to Clear&Start, and select the wanted input from either external pin or PRS, see Figure 19.14 (p. 450) . For period capture, the Compare/Capture Channel 0 should then be set to input capture on a rising edge of the same input signal. To capture the width of a high pulse, the channel should be set to capture on a falling edge of the input signal. To start the measuring period on either a falling edge or measure the low pulse-width of a signal, opposite polarities should be chosen.

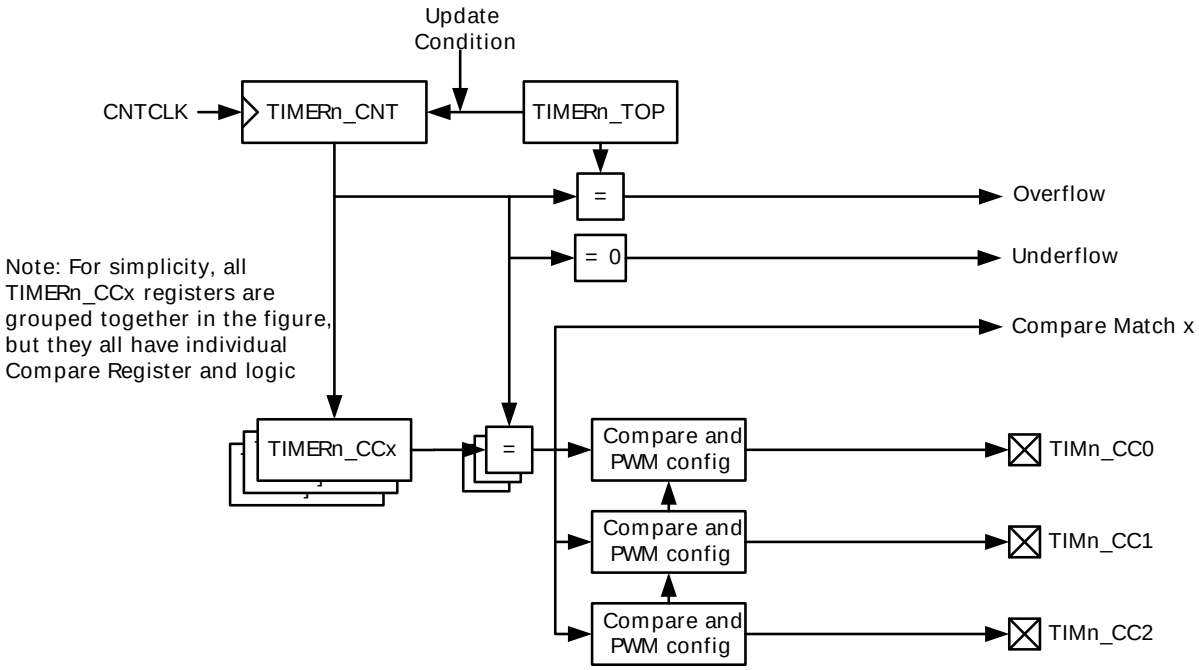
Figure 19.14. TIMER Period and/or Pulse width Capture



19.3.2.4 Compare

Each Compare/Capture channel contains a comparator which outputs a compare match if the contents of TIMERn_CCx_CCV matches the counter value, see Figure 19.15 (p. 451). In compare mode, each compare channel can be configured to either set, clear or toggle the output on an event (compare match, overflow or underflow). The output from each channel is represented as an alternative function on the port it is connected to, which needs to be enabled for the CC outputs to propagate to the pins.

Figure 19.15. TIMER Block Diagram Showing Comparison Functionality

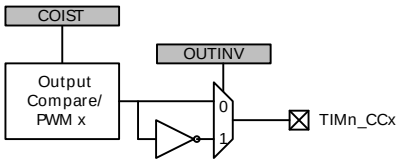


If occurring in the same cycle, match action will have priority over overflow or underflow action.

The input selected (through PRSSEL, INSEL and FILTSEL in TIMERn_CCx_CTRL) for the CC channel will also be sampled on compare match and the result is found in the CCPOL bits in TIMERn_STATUS. It is also possible to configure the CCPOL to always track the inputs by setting ATI in TIMERn_CTRL.

The COIST bit in TIMERn_CCx_CTRL is the initial state of the compare/PWM output. The COIST bit can also be used as an initial value to the compare outputs on a reload-start when RSSCOIST is set in TIMERn_CTRL. Also the resulting output can be inverted by setting OUTINV in TIMERn_CCx_CTRL. It is recommended to turn off the CC channel before configuring the output state to avoid any pulses on the output. The CC channel can be turned off by setting MODE to OFF in TIMER_CCx_CTRL.

Figure 19.16. TIMER Output Logic

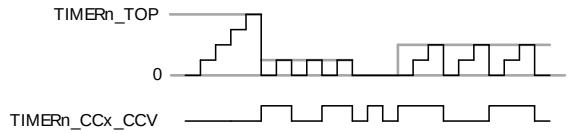


19.3.2.4.1 Frequency Generation (FRG)

Frequency generation (see Figure 19.17 (p. 452)) can be achieved in compare mode by:

- Setting the counter in up-count mode
- Enabling buffering of the TOP value.
- Setting the CC channels overflow action to toggle

Figure 19.17. TIMER Up-count Frequency Generation



The output frequency is given by Equation 19.2 (p. 452)

TIMER Up-count Frequency Generation Equation

$$f_{FRG} = f_{HFPERCLK} / (2^{(PRESC + 1)} \times (TOP + 1) \times 2) \quad (19.2)$$

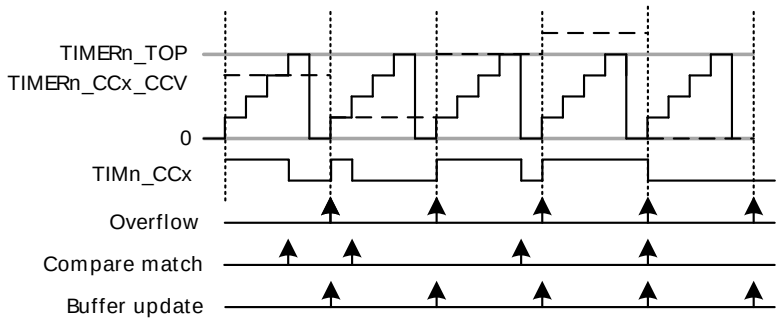
19.3.2.5 Pulse-Width Modulation (PWM)

In PWM mode, TIMERn_CCx_CCV is buffered to avoid glitches in the output. The settings in the Compare Output Action configuration bits are ignored in PWM mode and PWM generation is only supported for up-count and up/down-count mode.

19.3.2.6 Up-count (Single-slope) PWM

If the counter is set to up-count and the Compare/Capture channel is put in PWM mode, single slope PWM output will be generated (see Figure 19.18 (p. 452)). In up-count mode the PWM period is TOP +1 cycles and the PWM output will be high for a number of cycles equal to TIMERn_CCx_CCV. This means that a constant high output is achieved by setting TIMER_CCx to TOP+1 or higher. The PWM resolution (in bits) is then given by Equation 19.3 (p. 452) .

Figure 19.18. TIMER Up-count PWM Generation



TIMER Up-count PWM Resolution Equation

$$R_{PWM_{up}} = \log(TOP+1)/\log(2) \quad (19.3)$$

The PWM frequency is given by Equation 19.4 (p. 452) :

TIMER Up-count PWM Frequency Equation

$$f_{PWM_{up/down}} = f_{HFPERCLK} / (2^{PRESC} \times (TOP + 1) \quad (19.4)$$

The high duty cycle is given by Equation 19.5 (p. 453)

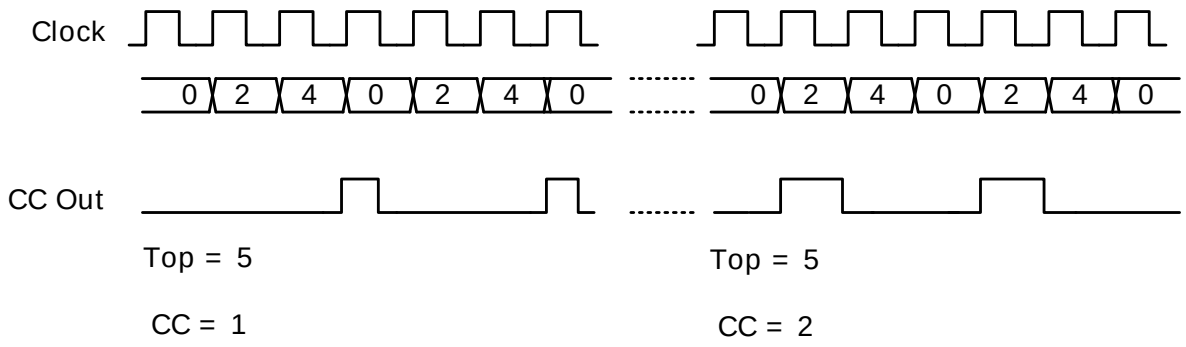
TIMER Up-count Duty Cycle Equation

$$DS_{up} = CCVx/TOP \quad (19.5)$$

19.3.2.6.1 2x Count Mode

When the Timer is set in 2x mode, the TIMER will count up by two. This will in effect make any odd Top value be rounded down to the closest even number. Similarly, any odd CC value will generate a match on the closest lower even value as shown in Figure 19.19 (p. 453)

Figure 19.19. TIMER CC out in 2x mode



The mode is enabled by setting the X2CNT field in TIMERn_CTRL register. The intended use of the 2x mode is to generate 2x PWM frequency when the Compare/Capture channel is put in PWM mode. Since the PWM output is updated on both edges of the clock, frequency prescaling will result in incorrect result in this mode. The PWM resolution (in bits) is then given by Equation 19.6 (p. 453) .

TIMER 2x PWM Resolution Equation

$$R_{PWM_{2xmode}} = \log(TOP/2+1)/\log(2) \quad (19.6)$$

The PWM frequency is given by Equation 19.7 (p. 453) :

TIMER 2x Mode PWM Frequency Equation(Up-count)

$$f_{PWM_{2xmode}} = 2 \times f_{HFPERCLK} / \text{floor}(TOP/2)+1 \quad (19.7)$$

The high duty cycle is given by Equation 19.8 (p. 453)

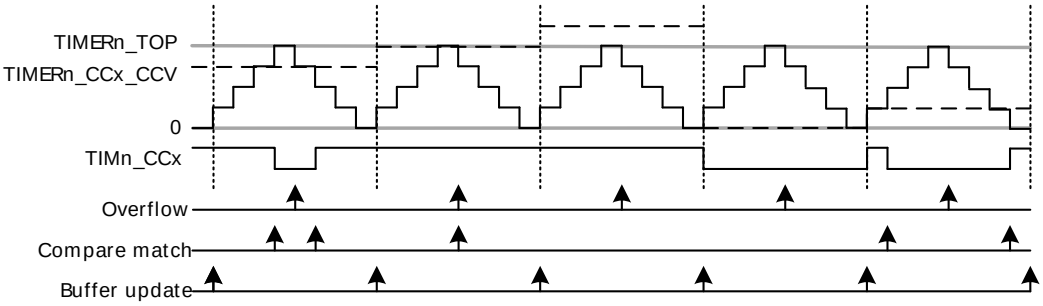
TIMER 2x Mode Duty Cycle Equation

$$DS_{2xmode} = CCVx/TOP \quad (19.8)$$

19.3.2.7 Up/Down-count (Dual-slope) PWM

If the counter is set to up-down count and the Compare/Capture channel is put in PWM mode, dual slope PWM output will be generated by Figure 19.20 (p. 454) .The resolution (in bits) is given by Equation 19.9 (p. 454) .

Figure 19.20. TIMER Up/Down-count PWM Generation



TIMER Up/Down-count PWM Resolution Equation

$$R_{PWM_{up/down}} = \log(TOP+1)/\log(2) \quad (19.9)$$

The PWM frequency is given by Equation 19.10 (p. 454) :

TIMER Up/Down-count PWM Frequency Equation

$$f_{PWM_{up/down}} = f_{HFPERCLK}/ (2\text{^}(PRESC+1) \times TOP) \quad (19.10)$$

The high duty cycle is given by Equation 19.11 (p. 454)

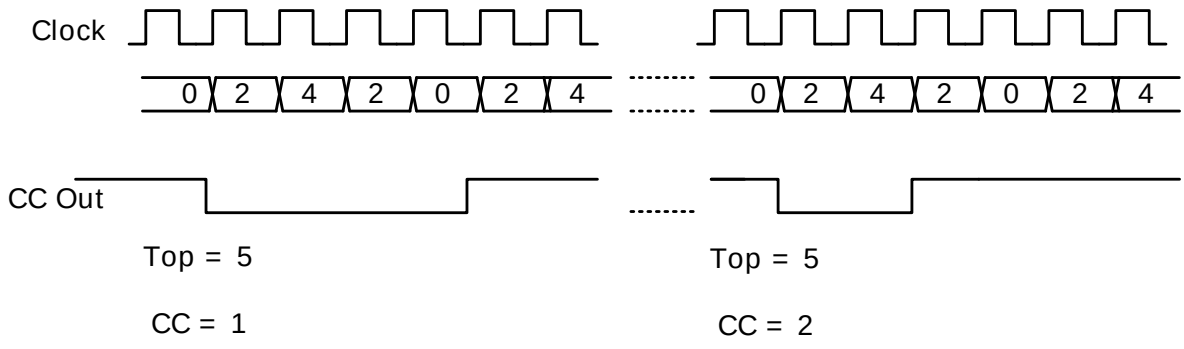
TIMER Up/Down-count Duty Cycle Equation

$$DS_{up/down} = CCVx/TOP \quad (19.11)$$

19.3.2.7.1 2x Count Mode

When the Timer is set in 2x mode, the TIMER will count up/down by two. This will in effect make any odd Top value be rounded down to the closest even number. Similarly, any odd CC value will generate a match on the closest lower even value as shown in Figure 19.21 (p. 454)

Figure 19.21. TIMER CC out in 2x mode



The mode is enabled by setting the X2CNT field in TIMERn_CTRL register. The intended use of the 2x mode is to generate 2x PWM frequency when the Compare/Capture channel is put in PWM mode. Since the PWM output is updated on both edges of the clock, frequency prescaling will result in incorrect result in this mode. The PWM resolution (in bits) is then given by Equation 19.12 (p. 454) .

TIMER 2x PWM Resolution Equation

$$R_{PWM_{2xmode}} = \log(TOP/2+1)/\log(2) \quad (19.12)$$

The PWM frequency is given by Equation 19.7 (p. 453) :

TIMER 2x Mode PWM Frequency Equation(Up/Down-count)

$$f_{PWM_{2xmode}} = f_{HFPERCLK} / TOP \quad (19.13)$$

The high duty cycle is given by Equation 19.14 (p. 455)

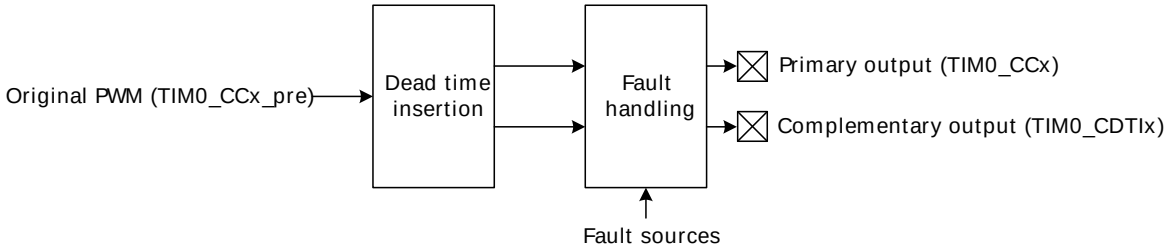
TIMER 2x Mode Duty Cycle Equation

$$DS_{2xmode} = CCVx/TOP \quad (19.14)$$

19.3.3 Dead-Time Insertion Unit (TIMER0 only)

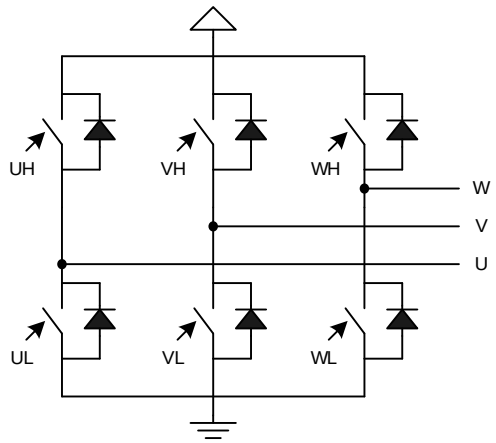
The Dead-Time Insertion Unit aims to make control of BLDC motors safer and more efficient by introducing complementary PWM outputs with dead-time insertion and fault handling, see Figure 19.22 (p. 455) .

Figure 19.22. TIMER Dead-Time Insertion Unit Overview



When used for motor control, the PWM outputs TIM0_CC0, TIM0_CC1 and TIM0_CC2 are often connected to the high-side transistors of a triple half-bridge setup (UH, VH and WH), and the complementary outputs connected to the respective low-side transistors (UL, VL, WL shown in Figure 19.23 (p. 455)). Transistors used in such a bridge often do not open/close instantaneously, and using the exact complementary inputs for the high and low side of a half-bridge may result in situations where both gates are open. This can give unnecessary current-draw and short circuit the power supply. The DTI unit provides dead-time insertion to deal with this problem.

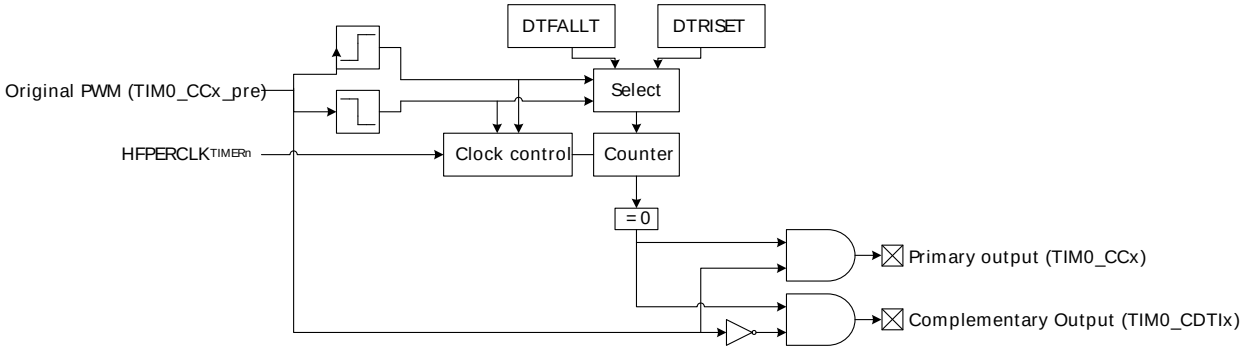
Figure 19.23. TIMER Triple Half-Bridge



For each of the 3 compare-match outputs of TIMER0, an additional complementary output is provided by the DTI unit. These outputs, named TIM0_CDTI0, TIM0_CDTI1 and TIM0_CDTI2 are provided to make

control of e.g. 3-channel BLDC or PMAC motors possible using only a single timer, see Figure 19.24 (p. 456) .

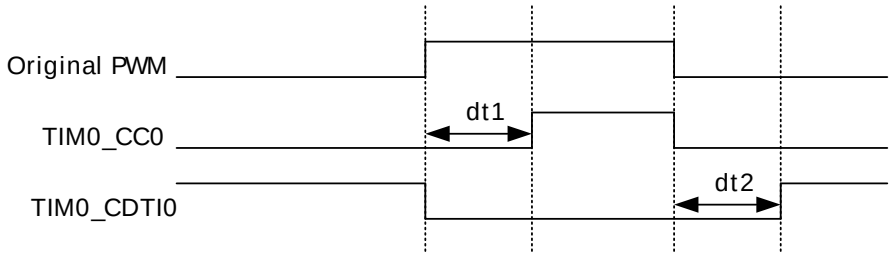
Figure 19.24. TIMER Overview of Dead-Time Insertion Block for a Single PWM channel



The DTI unit is enabled by setting DTEN in TIMER0_DTCTRL. In addition to providing the complementary outputs, the DTI unit then also overrides the compare match outputs from the timer.

The DTI unit gives the rising edges of the PWM outputs and the rising edges of the complementary PWM outputs a configurable time delay. By doing this, the DTI unit introduces a dead-time where both the primary and complementary outputs in a pair are inactive as seen in Figure 19.25 (p. 456) .

Figure 19.25. TIMER Polarity of Both Signals are Set as Active-High



Dead-time is specified individually for the rising and falling edge of the original PWM. These values are shared across all the three PWM channels of the DTI unit. A single prescaler value is provided for the DTI unit, meaning that both the rising and falling edge dead-times share prescaler value. The prescaler divides the $HFPERCLK_{TIMERn}$ by a configurable factor between 1 and 1024, which is set in the DTPRESC field in TIMER0_DTTIME. The rising and falling edge dead-times are configured in DTRISET and DTFALLT in TIMER0_DTTIME to any number between 1-64 $HFPERCLK_{TIMER0}$ cycles.

19.3.3.1 Output Polarity

The value of the primary and complementary outputs in a pair will never be set active at the same time by the DTI unit. The polarity of the outputs can be changed however, if this is required by the application. The active values of the primary and complementary outputs are set by two the TIMER0_DTCTRL register. The DTIPOL bit of this register specifies the base polarity. If DTIPOL =0, then the outputs are active-high, and if DTIPOL = 1 they are active-low. The relative phase of the primary and complementary outputs is not changed by DTIPOL, as the polarity of both outputs is changed, see Figure 19.26 (p. 457)

In some applications, it may be required that the primary outputs are active-high, while the complementary outputs are active-low. This can be accomplished by manipulating the DTCINV bit of the TIMER0_DTCTRL register, which inverts the polarity of the complementary outputs relative to the primary outputs.

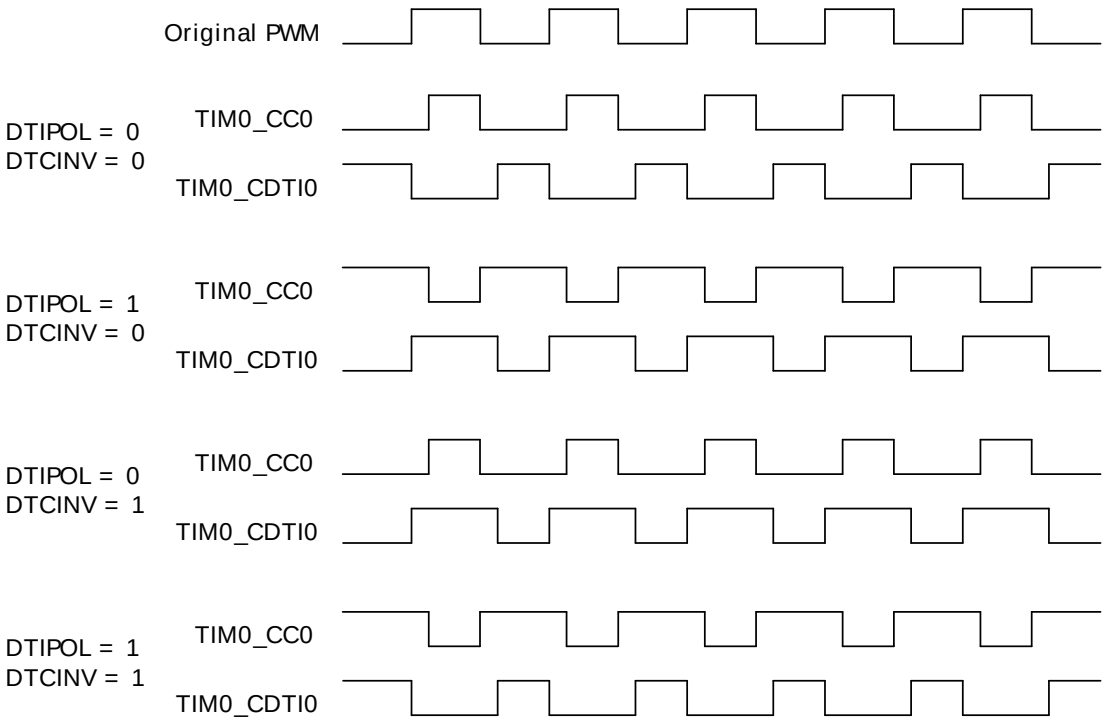
Example 19.1. TIMER DTI Example 1

DTIPOL = 0 and DTCINV = 0 results in outputs with opposite phase and active-high states.

Example 19.2. TIMER DTI Example 2

DTIPOL = 1 and DTCINV = 1 results in outputs with equal phase. The primary output will be active-high, while the complementary will be active-low

Figure 19.26. TIMER Output Polarities



Output generation on the individual DTI outputs can be disabled by configuring TIMER0_DTOGEN. When output generation on an output is disabled, it will go to and stay in its inactive state.

19.3.3.2 PRS Channel as Source

A PRS channel can optionally be used as input to the DTI module instead of the PWM output from the timer. Setting DTPRSEN in TIMER0_DTCTRL will override the source of the first DTI channel, driving TIM0_CC0 and TIM0_CDTI0, with the value on the PRS channel. The rest of the DTI channels will continue to be driven by the PWM output from the timer. The PRS channel to use is chosen by configuring DTPRSSEL in TIMER0_DTCTRL. Note that the timer must be running even when PRS is used as DTI source.

The DTI prescaler, set by DTPRESC in TIMER0_DTTIME determines with which accuracy the DTI can insert dead-time into a PRS signal. The maximum dead-time error equals $2^{DTPRESC}$ clock cycles. With zero prescaling, the inserted dead-times are therefore accurate, but they may be inaccurate for larger prescaler settings.

19.3.3.3 Fault Handling

The fault handling system of the DTI unit allows the outputs of the DTI unit to be put in a well-defined state in case of a fault. This hardware fault handling system makes a fast reaction to faults possible, reducing the possibility of damage to the system.

The fault sources which trigger a fault in the DTI module are determined by TIMER0_DTFSEN. Any combination of the available error sources can be selected:

- PRS source 0, determined by DTPRS0FSEL in TIMER0_DTFC
- PRS source 1, determined by DTPRS1FSEL in TIMER0_DTFC
- Debugger
- Core Lockup

One or two PRS channels can be used as an error source. When PRS source 0 is selected as an error source, DTPRS0FSEL determines which PRS channel is used for this source. DTPRS1FSEL determines which PRS channel is selected as PRS source 1. Please note that for Core Lockup, the LOCKUPRDIS in RMU_CTRL must be set. Otherwise this will generate a full reset of the EFM32.

19.3.3.3.1 Action on Fault

When a fault occurs, the bit representing the fault source is set in DTFS, and the outputs from the DTI unit are set to a well-defined state. The following options are available, and can be enabled by configuring DTFACT in TIMER0_DTFC:

- Set outputs to inactive level
- Clear outputs
- Tristate outputs

With the first option enabled, the output state in case of a fault depends on the polarity settings for the individual outputs. An output set to be active high will be set low if a fault is detected, while an output set to be active low will be driven high.

When a fault occurs, the fault source(s) can be read out of TIMER0_DTFS. TIMER0_DTFS is organized in the same way as DTFSEN, with one bit for each source.

19.3.3.3.2 Exiting Fault State

When a fault is triggered by the PRS system, software intervention is required to re-enable the outputs of the DTI unit. This is done by manually clearing TIMER0_DTFS. If the fault cause, determined by TIMER0_DTFS, is the debugger alone, the outputs can optionally be re-enabled when the debugger exits and the processor resumes normal operation. The corresponding bit in TIMER0_DTFS will in that case be cleared by hardware. The automatic start-up functionality can be enabled by setting DTDAS in TIMER0_DTCTRL. If more bits are still set in DTFS when the automatic start-up functionality has cleared the debugger bit, the DTI module does not exit the fault state. The fault state is only exited when all the bits in TIMER0_DTFS have been cleared.

19.3.3.4 Configuration Lock

To prevent software errors from making changes to the DTI configuration, a configuration lock is available. Writing any value but 0xCE80 to LOCKKEY in TIMER0_DTLOCK results in TIMER0_DTFC, TIMER0_DTCTRL, TIMER0_DTTIME and TIMER0_ROUTE being locked for writing. To unlock the registers, write 0xCE80 to LOCKKEY in TIMER0_DTLOCK. The value of TIMER0_DTLOCK is 1 when the lock is active, and 0 when the registers are unlocked.

19.3.4 Debug Mode

When the CPU is halted in debug mode, the timer can be configured to either continue to run or to be frozen. This is configured in DBGHALT in TIMERn_CTRL.

19.3.5 Interrupts, DMA and PRS Output

The Timer has 5 output events:

- Counter Underflow
- Counter Overflow
- Compare match or input capture (one per Compare/Capture channel)

Each of the events has its own interrupt flag. Also, there is one interrupt flag for each Compare/Capture channel which is set on buffer overflow in capture mode. Buffer overflow happens when a new capture pushes an old unread capture out of the TIMERn_CCx_CCV/TIMERn_CCx_CCVB register pair.

If the interrupt flags are set and the corresponding interrupt enable bits in TIMERn_IEN) are set high, the Timer will send out an interrupt request. Each of the events will also lead to a one $HFPERCLK_{TIMERn}$ cycle high pulse on individual PRS outputs. Setting PRSOCNF to LEVEL in TIMERn_CCx_CTRL will make the compare match PRS output follow the compare match output, instead of outputting one $HFPERCLK_{TIMERn}$ cycle high pulse.

Each of the events will also set a DMA request when they occur. The different DMA requests are cleared when certain acknowledge conditions are met, see Table 19.3 (p. 459) . If DMACLRACT is set in TIMERn_CTRL, the DMA request is cleared when the triggered DMA channel is active, without having to access any timer registers.

Table 19.3. TIMER Events

Event	Acknowledge
Underflow/Overflow	Read or write to TIMERn_CNT or TIMERn_TOPB
CC 0	Read or write to TIMERn_CC0_CCV or TIMERn_CC0_CCVB
CC 1	Read or write to TIMERn_CC1_CCV or TIMERn_CC1_CCVB
CC 2	Read or write to TIMERn_CC2_CCV or TIMERn_CC2_CCVB

19.3.6 GPIO Input/Output

The TIMn_CCx inputs/outputs and TIM0_CDTIx outputs are accessible as alternate functions through GPIO. Each pin connection can be enabled/disabled separately by setting the corresponding CCxPEN or CDTIxPEN bits in TIMERn_ROUTE. The LOCATION bits in the same register can be used to move all enabled pins to alternate pins.

19.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	TIMERn_CTRL	RW	Control Register
0x004	TIMERn_CMD	W1	Command Register
0x008	TIMERn_STATUS	R	Status Register
0x00C	TIMERn_IEN	RW	Interrupt Enable Register
0x010	TIMERn_IF	R	Interrupt Flag Register
0x014	TIMERn_IFS	W1	Interrupt Flag Set Register
0x018	TIMERn_IFC	W1	Interrupt Flag Clear Register
0x01C	TIMERn_TOP	RWH	Counter Top Value Register
0x020	TIMERn_TOPB	RW	Counter Top Value Buffer Register
0x024	TIMERn_CNT	RWH	Counter Value Register
0x028	TIMERn_ROUTE	RW	I/O Routing Register
0x030	TIMERn_CC0_CTRL	RW	CC Channel Control Register
0x034	TIMERn_CC0_CCV	RWH	CC Channel Value Register
0x038	TIMERn_CC0_CCVP	R	CC Channel Value Peek Register
0x03C	TIMERn_CC0_CCVB	RWH	CC Channel Buffer Register
0x040	TIMERn_CC1_CTRL	RW	CC Channel Control Register
0x044	TIMERn_CC1_CCV	RWH	CC Channel Value Register
0x048	TIMERn_CC1_CCVP	R	CC Channel Value Peek Register
0x04C	TIMERn_CC1_CCVB	RWH	CC Channel Buffer Register
0x050	TIMERn_CC2_CTRL	RW	CC Channel Control Register
0x054	TIMERn_CC2_CCV	RWH	CC Channel Value Register
0x058	TIMERn_CC2_CCVP	R	CC Channel Value Peek Register
0x05C	TIMERn_CC2_CCVB	RWH	CC Channel Buffer Register
0x070	TIMERn_DTCTRL	RW	DTI Control Register
0x074	TIMERn_DTTIME	RW	DTI Time Control Register
0x078	TIMERn_DTFC	RW	DTI Fault Configuration Register
0x07C	TIMERn_DTOGEN	RW	DTI Output Generation Enable Register
0x080	TIMERn_DTFAULT	R	DTI Fault Register
0x084	TIMERn_DTFAULTC	W1	DTI Fault Clear Register
0x088	TIMERn_DTLOCK	RW	DTI Configuration Lock Register

19.5 Register Description

19.5.1 TIMERn_CTRL - Control Register

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset			0	0	0x0										0x0				0		0x0		0x0		0	0	0	0	0		0x0	
Access			RW	RW	RW										RW				RW		RW		RW		RW	RW	RW	RW	RW		RW	
Name			RSSCOIST	ATI	PRESC										CLKSEL				X2CNT		FALLA		RISEA		DMACLRACT	DEBUGRUN	QDM	OSMEN	SYNC		MODE	

Bit	Name	Reset	Access	Description
31:30	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
29	RSSCOIST	0	RW	**Reload-Start Sets Compare Output initial State**
	When enabled, compare output is set to COIST value at Reload-Start event			
28	ATI	0	RW	**Always Track Inputs**
	Enable ATI makes CCPOL always track the polarity of the inputs			
27:24	PRESC	0x0	RW	**Prescaler Setting**
	These bits select the prescaling factor.			

Value	Mode	Description
0	DIV1	The HFPERCLK is undivided
1	DIV2	The HFPERCLK is divided by 2
2	DIV4	The HFPERCLK is divided by 4
3	DIV8	The HFPERCLK is divided by 8
4	DIV16	The HFPERCLK is divided by 16
5	DIV32	The HFPERCLK is divided by 32
6	DIV64	The HFPERCLK is divided by 64
7	DIV128	The HFPERCLK is divided by 128
8	DIV256	The HFPERCLK is divided by 256
9	DIV512	The HFPERCLK is divided by 512
10	DIV1024	The HFPERCLK is divided by 1024

23:18	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
17:16	CLKSEL	0x0	RW	**Clock Source Select**
	These bits select the clock source for the timer.			

Value	Mode	Description
0	PRESCHFPERCLK	Prescaled HFPERCLK
1	CC1	Compare/Capture Channel 1 Input
2	TIMEROUF	Timer is clocked by underflow(down-count) or overflow(up-count) in the lower numbered neighbor Timer

15:14	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
13	X2CNT	0	RW	**2x Count Mode**
	Enable 2x count mode			
12	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
11:10	FALLA	0x0	RW	**Timer Falling Input Edge Action**
	These bits select the action taken in the counter when a falling edge occurs on the input.			

Value	Mode	Description
0	NONE	No action
1	START	Start counter without reload

Bit	Name	Reset	Access	Description

Value	Mode	Description
2	STOP	Stop counter without reload
3	RELOADSTART	Reload and start counter

9:8 RISEA 0x0 RW **Timer Rising Input Edge Action**

These bits select the action taken in the counter when a rising edge occurs on the input.

Value	Mode	Description
0	NONE	No action
1	START	Start counter without reload
2	STOP	Stop counter without reload
3	RELOADSTART	Reload and start counter

7 DMACLRACT 0 RW **DMA Request Clear on Active**

When this bit is set, the DMA requests are cleared when the corresponding DMA channel is active. This enables the timer DMA requests to be cleared without accessing the timer.

6 DEBUGRUN 0 RW **Debug Mode Run Enable**

Set this bit to enable timer to run in debug mode.

Value	Description
0	Timer is frozen in debug mode
1	Timer is running in debug mode

5 QDM 0 RW **Quadrature Decoder Mode Selection**

This bit sets the mode for the quadrature decoder.

Value	Mode	Description
0	X2	X2 mode selected
1	X4	X4 mode selected

4 OSMEN 0 RW **One-shot Mode Enable**

Enable/disable one shot mode.

3 SYNC 0 RW **Timer Start/Stop/Reload Synchronization**

When this bit is set, the Timer is started/stopped/reloaded by start/stop/reload commands in the other timers

Value	Description
0	Timer is not started/stopped/reloaded by other timers
1	Timer is started/stopped/reloaded by other timers

2 Reserved *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

1:0 MODE 0x0 RW **Timer Mode**

These bit set the counting mode for the Timer. Note, when Quadrature Decoder Mode is selected (MODE = 'b11), the CLKSEL is don't care. The Timer is clocked by the Decoder Mode clock output.

Value	Mode	Description
0	UP	Up-count mode
1	DOWN	Down-count mode
2	UPDOWN	Up/down-count mode
3	QDEC	Quadrature decoder mode

19.5.2 TIMERn_CMD - Command Register

Offset	Bit Position																													
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															W1	W1
Name																															STOP	START

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	STOP	0	W1	**Stop Timer**
	Write a 1 to this bit to stop timer			
0	START	0	W1	**Start Timer**
	Write a 1 to this bit to start timer			

19.5.3 TIMERn_STATUS - Status Register

Offset	Bit Position																													
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset						0	0	0						0	0	0						0	0	0						0	0	0
Access						R	R	R						R	R	R						R	R	R						R	R	R
Name						CCPOL2	CCPOL1	CCPOL0						ICV2	ICV1	ICV0						CCVBV2	CCVBV1	CCVBV0						TOPBV	DIR	RUNNING

Bit	Name	Reset	Access	Description
31:27	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
26	CCPOL2	0	R	**CC2 Polarity**

In Input Capture mode, this bit indicates the polarity of the edge that triggered capture in TIMERn_CC2_CCV. In Compare/PWM mode, this bit indicates the polarity of the selected input to CC channel 2. These bits are cleared when CCMODE is written to 0b00 (Off).

Value	Mode	Description
0	LOWRISE	CC2 polarity low level/rising edge
1	HIGHFALL	CC2 polarity high level/falling edge

25 CCPOL1 0 R **CC1 Polarity**

In Input Capture mode, this bit indicates the polarity of the edge that triggered capture in TIMERn_CC1_CCV. In Compare/PWM mode, this bit indicates the polarity of the selected input to CC channel 1. These bits are cleared when CCMODE is written to 0b00 (Off).

Value	Mode	Description
0	LOWRISE	CC1 polarity low level/rising edge
1	HIGHFALL	CC1 polarity high level/falling edge

24 CCPOL0 0 R **CC0 Polarity**

In Input Capture mode, this bit indicates the polarity of the edge that triggered capture in TIMERn_CC0_CCV. In Compare/PWM mode, this bit indicates the polarity of the selected input to CC channel 0. These bits are cleared when CCMODE is written to 0b00 (Off).

Value	Mode	Description
0	LOWRISE	CC0 polarity low level/rising edge
1	HIGHFALL	CC0 polarity high level/falling edge

Bit	Name	Reset	Access	Description
23:19	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
18	ICV2	0	R	**CC2 Input Capture Valid**

This bit indicates that TIMERn_CC2_CCV contains a valid capture value. These bits are only used in input capture mode and are cleared when CCMODE is written to 0b00 (Off).

Value	Description
0	TIMERn_CC2_CCV does not contain a valid capture value(FIFO empty)
1	TIMERn_CC2_CCV contains a valid capture value(FIFO not empty)

17	ICV1	0	R	**CC1 Input Capture Valid**

This bit indicates that TIMERn_CC1_CCV contains a valid capture value. These bits are only used in input capture mode and are cleared when CCMODE is written to 0b00 (Off).

Value	Description
0	TIMERn_CC1_CCV does not contain a valid capture value(FIFO empty)
1	TIMERn_CC1_CCV contains a valid capture value(FIFO not empty)

16	ICV0	0	R	**CC0 Input Capture Valid**

This bit indicates that TIMERn_CC0_CCV contains a valid capture value. These bits are only used in input capture mode and are cleared when CCMODE is written to 0b00 (Off).

Value	Description
0	TIMERn_CC0_CCV does not contain a valid capture value(FIFO empty)
1	TIMERn_CC0_CCV contains a valid capture value(FIFO not empty)

15:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	CCVBV2	0	R	**CC2 CCVB Valid**

This field indicates that the TIMERn_CC2_CCVB registers contain data which have not been written to TIMERn_CC2_CCV. These bits are only used in output compare/pwm mode and are cleared when CCMODE is written to 0b00 (Off).

Value	Description
0	TIMERn_CC2_CCVB does not contain valid data
1	TIMERn_CC2_CCVB contains valid data which will be written to TIMERn_CC2_CCV on the next update event

9	CCVBV1	0	R	**CC1 CCVB Valid**

This field indicates that the TIMERn_CC1_CCVB registers contain data which have not been written to TIMERn_CC1_CCV. These bits are only used in output compare/pwm mode and are cleared when CCMODE is written to 0b00 (Off).

Value	Description
0	TIMERn_CC1_CCVB does not contain valid data
1	TIMERn_CC1_CCVB contains valid data which will be written to TIMERn_CC1_CCV on the next update event

8	CCVBV0	0	R	**CC0 CCVB Valid**

This field indicates that the TIMERn_CC0_CCVB registers contain data which have not been written to TIMERn_CC0_CCV. These bits are only used in output compare/pwm mode and are cleared when CCMODE is written to 0b00 (Off).

Value	Description
0	TIMERn_CC0_CCVB does not contain valid data
1	TIMERn_CC0_CCVB contains valid data which will be written to TIMERn_CC0_CCV on the next update event

7:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	TOPBV	0	R	**TOPB Valid**

This indicates that TIMERn_TOPB contains valid data that has not been written to TIMERn_TOP. This bit is also cleared when TIMERn_TOP is written.

Value	Description
0	TIMERn_TOPB does not contain valid data
1	TIMERn_TOPB contains valid data which will be written to TIMERn_TOP on the next update event

1	DIR	0	R	**Direction**

Indicates count direction.

Value	Mode	Description
0	UP	Counting up

Bit	Name	Reset	Access	Description
	Value	Mode	Description	
	1	DOWN	Counting down	
0	RUNNING	0	R	**Running**
	Indicates if timer is running or not.			

19.5.4 TIMERn_IEN - Interrupt Enable Register

Offset	Bit Position																												
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																						0	0	0		0	0	0			0	0
Access																						RW	RW	RW		RW	RW	RW			RW	RW
Name																						ICBOF2	ICBOF1	ICBOF0		CC2	CC1	CC0			UF	OF

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	ICBOF2	0	RW	**CC Channel 2 Input Capture Buffer Overflow Interrupt Enable**
	Enable/disable Compare/Capture ch 2 input capture buffer overflow interrupt.			
9	ICBOF1	0	RW	**CC Channel 1 Input Capture Buffer Overflow Interrupt Enable**
	Enable/disable Compare/Capture ch 1 input capture buffer overflow interrupt.			
8	ICBOF0	0	RW	**CC Channel 0 Input Capture Buffer Overflow Interrupt Enable**
	Enable/disable Compare/Capture ch 0 input capture buffer overflow interrupt.			
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	CC2	0	RW	**CC Channel 2 Interrupt Enable**
	Enable/disable Compare/Capture ch 2 interrupt.			
5	CC1	0	RW	**CC Channel 1 Interrupt Enable**
	Enable/disable Compare/Capture ch 1 interrupt.			
4	CC0	0	RW	**CC Channel 0 Interrupt Enable**
	Enable/disable Compare/Capture ch 0 interrupt.			
3:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	UF	0	RW	**Underflow Interrupt Enable**
	Enable/disable underflow interrupt.			
0	OF	0	RW	**Overflow Interrupt Enable**
	Enable/disable overflow interrupt.			

19.5.5 TIMERn_IF - Interrupt Flag Register

Offset	Bit Position																												
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																						0	0	0		0	0	0			0	0
Access																						R	R	R		R	R	R			R	R
Name																						ICBOF2	ICBOF1	ICBOF0		CC2	CC1	CC0			UF	OF

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	ICBOF2	0	R	**CC Channel 2 Input Capture Buffer Overflow Interrupt Flag**
	This bit indicates that a new capture value has pushed an unread value out of the TIMERn_CC2_CCV/TIMERn_CC2_CCVB register pair.			
9	ICBOF1	0	R	**CC Channel 1 Input Capture Buffer Overflow Interrupt Flag**
	This bit indicates that a new capture value has pushed an unread value out of the TIMERn_CC1_CCV/TIMERn_CC1_CCVB register pair.			
8	ICBOF0	0	R	**CC Channel 0 Input Capture Buffer Overflow Interrupt Flag**
	This bit indicates that a new capture value has pushed an unread value out of the TIMERn_CC0_CCV/TIMERn_CC0_CCVB register pair.			
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	CC2	0	R	**CC Channel 2 Interrupt Flag**
	This bit indicates that there has been an interrupt event on Compare/Capture channel 2.			
5	CC1	0	R	**CC Channel 1 Interrupt Flag**
	This bit indicates that there has been an interrupt event on Compare/Capture channel 1.			
4	CC0	0	R	**CC Channel 0 Interrupt Flag**
	This bit indicates that there has been an interrupt event on Compare/Capture channel 0.			
3:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	UF	0	R	**Underflow Interrupt Flag**
	This bit indicates that there has been an underflow.			
0	OF	0	R	**Overflow Interrupt Flag**
	This bit indicates that there has been an overflow.			

19.5.6 TIMERn_IFS - Interrupt Flag Set Register

Offset	Bit Position																												
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																						0	0	0		0	0	0			0	0
Access																						W1	W1	W1		W1	W1	W1			W1	W1
Name																						ICBOF2	ICBOF1	ICBOF0		CC2	CC1	CC0			UF	OF

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	ICBOF2	0	W1	**CC Channel 2 Input Capture Buffer Overflow Interrupt Flag Set**
	Writing a 1 to this bit will set Compare/Capture channel 2 input capture buffer overflow interrupt flag.			
9	ICBOF1	0	W1	**CC Channel 1 Input Capture Buffer Overflow Interrupt Flag Set**
	Writing a 1 to this bit will set Compare/Capture channel 1 input capture buffer overflow interrupt flag.			
8	ICBOF0	0	W1	**CC Channel 0 Input Capture Buffer Overflow Interrupt Flag Set**
	Writing a 1 to this bit will set Compare/Capture channel 0 input capture buffer overflow interrupt flag.			
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	CC2	0	W1	**CC Channel 2 Interrupt Flag Set**
	Writing a 1 to this bit will set Compare/Capture channel 2 interrupt flag.			
5	CC1	0	W1	**CC Channel 1 Interrupt Flag Set**

Bit	Name	Reset	Access	Description
	Writing a 1 to this bit will set Compare/Capture channel 1 interrupt flag.			
4	CC0	0	W1	**CC Channel 0 Interrupt Flag Set**
	Writing a 1 to this bit will set Compare/Capture channel 0 interrupt flag.			
3:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	UF	0	W1	**Underflow Interrupt Flag Set**
	Writing a 1 to this bit will set the underflow interrupt flag.			
0	OF	0	W1	**Overflow Interrupt Flag Set**
	Writing a 1 to this bit will set the overflow interrupt flag.			

19.5.7 TIMERn_IFC - Interrupt Flag Clear Register

Offset	Bit Position																												
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																						0	0	0		0	0	0			0	0
Access																						W1	W1	W1		W1	W1	W1			W1	W1
Name																						ICBOF2	ICBOF1	ICBOF0		CC2	CC1	CC0			UF	OF

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	ICBOF2	0	W1	**CC Channel 2 Input Capture Buffer Overflow Interrupt Flag Clear**
	Writing a 1 to this bit will clear Compare/Capture channel 2 input capture buffer overflow interrupt flag.			
9	ICBOF1	0	W1	**CC Channel 1 Input Capture Buffer Overflow Interrupt Flag Clear**
	Writing a 1 to this bit will clear Compare/Capture channel 1 input capture buffer overflow interrupt flag.			
8	ICBOF0	0	W1	**CC Channel 0 Input Capture Buffer Overflow Interrupt Flag Clear**
	Writing a 1 to this bit will clear Compare/Capture channel 0 input capture buffer overflow interrupt flag.			
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	CC2	0	W1	**CC Channel 2 Interrupt Flag Clear**
	Writing a 1 to this bit will clear Compare/Capture interrupt flag 2.			
5	CC1	0	W1	**CC Channel 1 Interrupt Flag Clear**
	Writing a 1 to this bit will clear Compare/Capture interrupt flag 1.			
4	CC0	0	W1	**CC Channel 0 Interrupt Flag Clear**
	Writing a 1 to this bit will clear Compare/Capture interrupt flag 0.			
3:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	UF	0	W1	**Underflow Interrupt Flag Clear**
	Writing a 1 to this bit will clear the underflow interrupt flag.			
0	OF	0	W1	**Overflow Interrupt Flag Clear**
	Writing a 1 to this bit will clear th overflow interrupt flag.			

19.5.8 TIMERn_TOP - Counter Top Value Register

Offset	Bit Position																															
0x01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0xFFFF															
Access																	RWH															
Name																	TOP															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	TOP	0xFFFF	RWH	**Counter Top Value**
	These bits hold the TOP value for the counter.			

19.5.9 TIMERn_TOPB - Counter Top Value Buffer Register

Offset	Bit Position																															
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	TOPB															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	TOPB	0x0000	RW	**Counter Top Value Buffer**
	These bits hold the TOP buffer value.			

19.5.10 TIMERn_CNT - Counter Value Register

Offset	Bit Position																															
0x024	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RWH															
Name																	CNT															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	CNT	0x0000	RWH	**Counter Value**
	These bits hold the counter value.			

19.5.11 TIMERn_ROUTE - I/O Routing Register

Offset	Bit Position																												
0x028	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset															0x0							0	0	0						0	0	0
Access															RW							RW	RW	RW						RW	RW	RW
Name															LOCATION							CDTI2PEN	CDTI1PEN	CDTI0PEN						CC2PEN	CC1PEN	CC0PEN

Bit	Name	Reset	Access	Description
31:19	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
18:16	LOCATION	0x0	RW	**I/O Location**
	Decides the location of the CC and CDTI pins.			

Value	Mode	Description
0	LOC0	Location 0
1	LOC1	Location 1
2	LOC2	Location 2
3	LOC3	Location 3
4	LOC4	Location 4
5	LOC5	Location 5
6	LOC6	Location 6

15:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	CDTI2PEN	0	RW	**CC Channel 2 Complementary Dead-Time Insertion Pin Enable**
	Enable/disable CC channel 2 complementary dead-time insertion output connection to pin.			
9	CDTI1PEN	0	RW	**CC Channel 1 Complementary Dead-Time Insertion Pin Enable**
	Enable/disable CC channel 1 complementary dead-time insertion output connection to pin.			
8	CDTI0PEN	0	RW	**CC Channel 0 Complementary Dead-Time Insertion Pin Enable**
	Enable/disable CC channel 0 complementary dead-time insertion output connection to pin.			
7:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	CC2PEN	0	RW	**CC Channel 2 Pin Enable**
	Enable/disable CC channel 2 output/input connection to pin.			
1	CC1PEN	0	RW	**CC Channel 1 Pin Enable**
	Enable/disable CC channel 1 output/input connection to pin.			
0	CC0PEN	0	RW	**CC Channel 0 Pin Enable**
	Enable/disable CC Channel 0 output/input connection to pin.			

19.5.12 TIMERn_CCx_CTRL - CC Channel Control Register

Offset	Bit Position																													
0x030	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset				0	0x0		0x0				0	0			0x0				0x0		0x0		0x0					0		0		0x0
Access				RW	RW		RW				RW	RW			RW				RW		RW		RW					RW		RW		RW
Name				PRSCONF	ICEVCTRL		ICEDGE				FILT	INSEL			PRSSEL				CUFOA		COFOA		CMOA					COIST		OUTINV		MODE

Bit	Name	Reset	Access	Description
31:29	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
28	PRSCONF	0	RW	**PRS Configuration**

Select PRS pulse or level.

Value	Mode	Description
0	PULSE	Each CC event will generate one HFPERCLK cycle high pulse
1	LEVEL	The PRS channel will follow CC out

27:26	ICEVCTRL	0x0	RW	**Input Capture Event Control**

These bits control when a Compare/Capture PRS output pulse, interrupt flag and DMA request is set.

Value	Mode	Description
0	EVERYEDGE	PRS output pulse, interrupt flag and DMA request set on every capture
1	EVERYSECONDEDGE	PRS output pulse, interrupt flag and DMA request set on every second capture
2	RISING	PRS output pulse, interrupt flag and DMA request set on rising edge only (if ICEDGE = BOTH)
3	FALLING	PRS output pulse, interrupt flag and DMA request set on falling edge only (if ICEDGE = BOTH)

25:24	ICEDGE	0x0	RW	**Input Capture Edge Select**

These bits control which edges the edge detector triggers on. The output is used for input capture and external clock input.

Value	Mode	Description
0	RISING	Rising edges detected
1	FALLING	Falling edges detected
2	BOTH	Both edges detected
3	NONE	No edge detection, signal is left as it is

23:22	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
21	FILT	0	RW	**Digital Filter**

Enable digital filter.

Value	Mode	Description
0	DISABLE	Digital filter disabled
1	ENABLE	Digital filter enabled

20	INSEL	0	RW	**Input Selection**

Select Compare/Capture channel input.

Value	Mode	Description
0	PIN	TIMERnCCx pin is selected
1	PRS	PRS input (selected by PRSSEL) is selected

19	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
18:16	PRSSEL	0x0	RW	**Compare/Capture Channel PRS Input Channel Selection**

Select PRS input channel for Compare/Capture channel.

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected as input

Bit	Name	Reset	Access	Description

Value	Mode	Description
1	PRSCH1	PRS Channel 1 selected as input
2	PRSCH2	PRS Channel 2 selected as input
3	PRSCH3	PRS Channel 3 selected as input
4	PRSCH4	PRS Channel 4 selected as input
5	PRSCH5	PRS Channel 5 selected as input

15:14 *Reserved* *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

13:12 CUFOA 0x0 RW **Counter Underflow Output Action**

Select output action on counter underflow.

Value	Mode	Description
0	NONE	No action on counter underflow
1	TOGGLE	Toggle output on counter underflow
2	CLEAR	Clear output on counter underflow
3	SET	Set output on counter underflow

11:10 COFOA 0x0 RW **Counter Overflow Output Action**

Select output action on counter overflow.

Value	Mode	Description
0	NONE	No action on counter overflow
1	TOGGLE	Toggle output on counter overflow
2	CLEAR	Clear output on counter overflow
3	SET	Set output on counter overflow

9:8 CMOA 0x0 RW **Compare Match Output Action**

Select output action on compare match.

Value	Mode	Description
0	NONE	No action on compare match
1	TOGGLE	Toggle output on compare match
2	CLEAR	Clear output on compare match
3	SET	Set output on compare match

7:5 *Reserved* *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

4 COIST 0 RW **Compare Output Initial State**

This bit is only used in Output Compare and PWM mode. When this bit is set in compare mode,the output is set high when the counter is disabled. When counting resumes, this value will represent the initial value for the output. If the bit is cleared, the output will be cleared when the counter is disabled. In PWM mode, the output will always be low when disabled, regardless of this bit. However, this bit will represent the initial value of the output, once it is enabled.

3 *Reserved* *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

2 OUTINV 0 RW **Output Invert**

Setting this bit inverts the output from the CC channel (Output compare,PWM).

1:0 MODE 0x0 RW **CC Channel Mode**

These bits select the mode for Compare/Capture channel.

Value	Mode	Description
0	OFF	Compare/Capture channel turned off
1	INPUTCAPTURE	Input capture
2	OUTPUTCOMPARE	Output compare
3	PWM	Pulse-Width Modulation

19.5.13 TIMERn_CCx_CCV - CC Channel Value Register

Offset	Bit Position																													
0x034	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RWH															
Name																	CCV															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	CCV	0x0000	RWH	**CC Channel Value**
	In input capture mode, this field holds the first unread capture value. When reading this register in input capture mode, then contents of the TIMERn_CCx_CCVB register will be written to TIMERn_CCx_CCV in the next cycle. In compare mode, this fields holds the compare value.			

19.5.14 TIMERn_CCx_CCVP - CC Channel Value Peek Register

Offset	Bit Position																													
0x038	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	R															
Name																	CCVP															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	CCVP	0x0000	R	**CC Channel Value Peek**
	This field is used to read the CC value without pulling data through the FIFO in capture mode.			

19.5.15 TIMERn_CCx_CCVB - CC Channel Buffer Register

Offset	Bit Position																													
0x03C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RWH															
Name																	CCVB															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	CCVB	0x0000	RWH	**CC Channel Value Buffer**
	In Input Capture mode, this field holds the last capture value if the TIMERn_CCx_CCV register already contains an earlier unread capture value. In Output Compare or PWM mode, this field holds the CC buffer value which will be written to TIMERn_CCx_CCV on an update event if TIMERn_CCx_CCVB contains valid data.			

19.5.16 TIMERn_DTCTRL - DTI Control Register

Offset	Bit Position																													
0x070	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset								0																		0x0			0	0	0	0
Access								RW																		RW			RW	RW	RW	RW
Name								DTPRSEN																		DTPRSSEL			DTCINV	DTIPOL	DTDAS	DTEN

Bit	Name	Reset	Access	Description
31:25	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
24	DTPRSEN	0	RW	**DTI PRS Source Enable**
	Enable/disable PRS as DTI input.			
23:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:4	DTPRSSEL	0x0	RW	**DTI PRS Source Channel Select**
	Select which PRS channel to listen to.			

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected as input
1	PRSCH1	PRS Channel 1 selected as input
2	PRSCH2	PRS Channel 2 selected as input
3	PRSCH3	PRS Channel 3 selected as input
4	PRSCH4	PRS Channel 4 selected as input
5	PRSCH5	PRS Channel 5 selected as input

3	DTCINV	0	RW	**DTI Complementary Output Invert.**
	Set to invert complementary outputs.			
2	DTIPOL	0	RW	**DTI Inactive Polarity**
	Set inactive polarity for outputs.			
1	DTDAS	0	RW	**DTI Automatic Start-up Functionality**
	Configure DTI restart on debugger exit.			

Value	Mode	Description
0	NORESTART	No DTI restart on debugger exit
1	RESTART	DTI restart on debugger exit

0	DTEN	0	RW	**DTI Enable**
	Enable/disable DTI.			

19.5.17 TIMERn_DTTIME - DTI Time Control Register

Offset	Bit Position																													
0x074	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset											0x00								0x00										0x0			
Access											RW								RW										RW			
Name											DTFALLT								DTRISET										DTPRESC			

Bit	Name	Reset	Access	Description
31:22	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
21:16	DTFALLT	0x00	RW	**DTI Fall-time**

Set time span for the falling edge.

Value	Description
DTFALLT	Fall time of DTFALLT+1 prescaled HFPERCLK cycles

15:14	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
13:8	DTRISET	0x00	RW	**DTI Rise-time**

Set time span for the rising edge.

Value	Description
DTRISET	Rise time of DTRISET+1 prescaled HFPERCLK cycles

7:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3:0	DTPRESC	0x0	RW	**DTI Prescaler Setting**

Select prescaler for DTI.

Value	Mode	Description
0	DIV1	The HFPERCLK is undivided
1	DIV2	The HFPERCLK is divided by 2
2	DIV4	The HFPERCLK is divided by 4
3	DIV8	The HFPERCLK is divided by 8
4	DIV16	The HFPERCLK is divided by 16
5	DIV32	The HFPERCLK is divided by 32
6	DIV64	The HFPERCLK is divided by 64
7	DIV128	The HFPERCLK is divided by 128
8	DIV256	The HFPERCLK is divided by 256
9	DIV512	The HFPERCLK is divided by 512
10	DIV1024	The HFPERCLK is divided by 1024

19.5.18 TIMERn_DTFC - DTI Fault Configuration Register

Offset	Bit Position																													
0x078	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset					0	0	0	0							0x0								0x0								0x0	
Access					RW	RW	RW	RW							RW								RW								RW	
Name					DTLOCKUPFEN	DTDBGFEN	DTPRS1FEN	DTPRS0FEN							DTFA								DTPRS1FSEL								DTPRS0FSEL	

Bit	Name	Reset	Access	Description
31:28	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
27	DTLOCKUPFEN	0	RW	**DTI Lockup Fault Enable**
	Set this bit to 1 to enable core lockup as a fault source			
26	DTDBGFEN	0	RW	**DTI Debugger Fault Enable**
	Set this bit to 1 to enable debugger as a fault source			
25	DTPRS1FEN	0	RW	**DTI PRS 1 Fault Enable**
	Set this bit to 1 to enable PRS source 1(PRS channel determined by DTPRS1FSEL) as a fault source			
24	DTPRS0FEN	0	RW	**DTI PRS 0 Fault Enable**
	Set this bit to 1 to enable PRS source 0(PRS channel determined by DTPRS0FSEL) as a fault source			
23:18	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
17:16	DTFA	0x0	RW	**DTI Fault Action**
	Select fault action.			

Value	Mode	Description
0	NONE	No action on fault
1	INACTIVE	Set outputs inactive
2	CLEAR	Clear outputs
3	TRISTATE	Tristate outputs

15:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	DTPRS1FSEL	0x0	RW	**DTI PRS Fault Source 1 Select**
	Select PRS channel for fault source 1.			

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected as fault source 1
1	PRSCH1	PRS Channel 1 selected as fault source 1
2	PRSCH2	PRS Channel 2 selected as fault source 1
3	PRSCH3	PRS Channel 3 selected as fault source 1
4	PRSCH4	PRS Channel 4 selected as fault source 1
5	PRSCH5	PRS Channel 5 selected as fault source 1
6	PRSCH6	PRS Channel 6 selected as fault source 1
7	PRSCH7	PRS Channel 7 selected as fault source 1

7:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2:0	DTPRS0FSEL	0x0	RW	**DTI PRS Fault Source 0 Select**
	Select PRS channel for fault source 0.			

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected as fault source 0
1	PRSCH1	PRS Channel 1 selected as fault source 0
2	PRSCH2	PRS Channel 2 selected as fault source 0
3	PRSCH3	PRS Channel 3 selected as fault source 0
4	PRSCH4	PRS Channel 4 selected as fault source 0
5	PRSCH5	PRS Channel 5 selected as fault source 0
6	PRSCH6	PRS Channel 6 selected as fault source 0
7	PRSCH7	PRS Channel 7 selected as fault source 0

19.5.19 TIMERn_DTOGEN - DTI Output Generation Enable Register

Offset	Bit Position																													
0x07C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																											0	0	0	0	0	0
Access																											RW	RW	RW	RW	RW	RW
Name																											DTOGCDTI2EN	DTOGCDTI1EN	DTOGCDTI0EN	DTOGCC2EN	DTOGCC1EN	DTOGCC0EN

Bit	Name	Reset	Access	Description
31:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5	DTOGCDTI2EN	0	RW	**DTI CDTI2 Output Generation Enable**
	This bit enables/disables output generation for the CDTI2 output from the DTI.			
4	DTOGCDTI1EN	0	RW	**DTI CDTI1 Output Generation Enable**
	This bit enables/disables output generation for the CDTI1 output from the DTI.			
3	DTOGCDTI0EN	0	RW	**DTI CDTI0 Output Generation Enable**
	This bit enables/disables output generation for the CDTI0 output from the DTI.			
2	DTOGCC2EN	0	RW	**DTI CC2 Output Generation Enable**
	This bit enables/disables output generation for the CC2 output from the DTI.			
1	DTOGCC1EN	0	RW	**DTI CC1 Output Generation Enable**
	This bit enables/disables output generation for the CC1 output from the DTI.			
0	DTOGCC0EN	0	RW	**DTI CC0 Output Generation Enable**
	This bit enables/disables output generation for the CC0 output from the DTI.			

19.5.20 TIMERn_DTFAULT - DTI Fault Register

Offset	Bit Position																													
0x080	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																													0	0	0	0
Access																													R	R	R	R
Name																													DTLOCKUPF	DTDBGF	DTPRS1F	DTPRS0F

Bit	Name	Reset	Access	Description
31:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	DTLOCKUPF	0	R	**DTI Lockup Fault**
	This bit is set to 1 if a core lockup fault has occurred and DTLOCKUPFEN is set to 1. The TIMER0_DTFAULTC register can be used to clear fault bits.			
2	DTDBGF	0	R	**DTI Debugger Fault**
	This bit is set to 1 if a debugger fault has occurred and DTDBGFEN is set to 1. The TIMER0_DTFAULTC register can be used to clear fault bits.			
1	DTPRS1F	0	R	**DTI PRS 1 Fault**
	This bit is set to 1 if a PRS 1 fault has occurred and DTPRS1FEN is set to 1. The TIMER0_DTFAULTC register can be used to clear fault bits.			

Bit	Name	Reset	Access	Description
0	DTPRS0F	0	R	**DTI PRS 0 Fault**
	This bit is set to 1 if a PRS 0 fault has occurred and DTPRS0FEN is set to 1. The TIMER0_DTFAULTC register can be used to clear fault bits.			

19.5.21 TIMERn_DTFAULTC - DTI Fault Clear Register

Offset	Bit Position																															
0x084	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																													0	0	0	0
Access																													W1	W1	W1	W1
Name																													TLOCKUPFC	DTDBGFC	DTPRS1FC	DTPRS0FC

Bit	Name	Reset	Access	Description
31:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	TLOCKUPFC	0	W1	**DTI Lockup Fault Clear**
	Write 1 to this bit to clear core lockup fault.			
2	DTDBGFC	0	W1	**DTI Debugger Fault Clear**
	Write 1 to this bit to clear debugger fault.			
1	DTPRS1FC	0	W1	**DTI PRS1 Fault Clear**
	Write 1 to this bit to clear PRS 1 fault.			
0	DTPRS0FC	0	W1	**DTI PRS0 Fault Clear**
	Write 1 to this bit to clear PRS 0 fault.			

19.5.22 TIMERn_DTLOCK - DTI Configuration Lock Register

Offset	Bit Position																															
0x088	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	LOCKKEY															

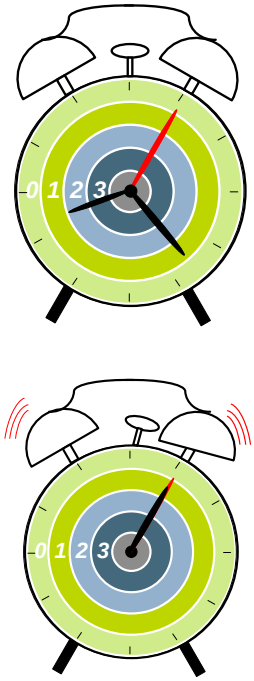
Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	LOCKKEY	0x0000	RW	**DTI Lock Key**

Write any other value than the unlock code to lock TIMER0_ROUTE, TIMER0_DTCTRL, TIMER0_DTTIME and TIMER0_DTFC from editing. Write the unlock code to unlock. When reading the register, bit 0 is set when the lock is enabled.

Mode	Value	Description
Read Operation		
UNLOCKED	0	TIMER DTI registers are unlocked
LOCKED	1	TIMER DTI registers are locked

Bit	Name	Reset	Access	Description
	Mode	Value		Description
	Write Operation			
	LOCK	0		Lock TIMER DTI registers
	UNLOCK	0xCE80		Unlock TIMER DTI registers

20 RTC - Real Time Counter



Quick Facts

What?

The Real Time Counter (RTC) ensures timekeeping in low energy modes. Combined with two low power oscillators (XTAL or RC), the RTC can run in EM2 with total current consumption less than 0.9 µA, and in EM3 with total current consumption less than 0.5 µA.

Why?

Timekeeping over long time periods is required in many applications, while using as little power as possible.

How?

Selectable 1 kHz and 32.768 Hz oscillators that can be used as clock source and two different compare registers that can trigger a wake-up. 24-bit resolution and selectable prescaling allow the system to stay in EM2 or EM3 for a long time and still maintain reliable timekeeping.

20.1 Introduction

The Real Time Counter (RTC) contains a 24-bit counter and is clocked either by a 32.768 Hz crystal oscillator, a 32.768 Hz RC oscillator, or a 1 kHz RC oscillator . In addition to energy modes EM0 and EM1, the RTC is also available in EM2. This makes it ideal for keeping track of time since the RTC is enabled in EM2 where most of the device is powered down. Using the 1 kHz ULFRCO as input clock, the RTC can be used for timekeeping all the way down to EM3.

Two compare channels are available in the RTC. These can be used to trigger interrupts and to wake the device up from a low energy mode. They can also be used with the LETIMER to generate various output waveforms.

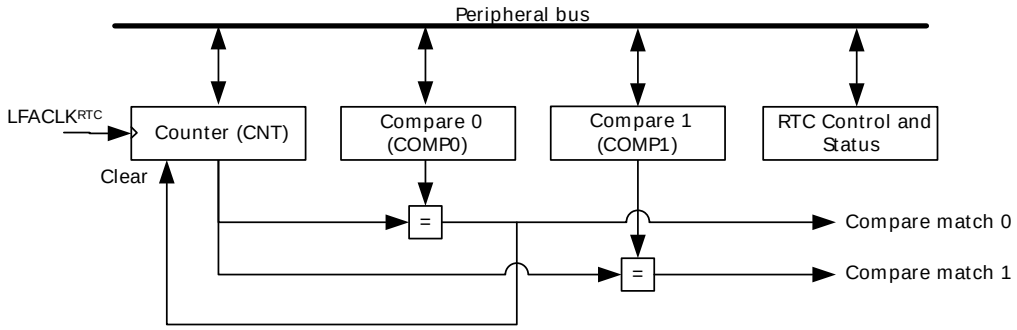
20.2 Features

- 24-bit Real Time Counter.
- Prescaler
  - 32.768 kHz/2^N, N = 0 - 15.
  - Overflow @ 0.14 hours for prescaler setting = 0.
  - Overflow @ 4660 hours (194 days) for prescaler setting = 15 (1 s tick).
- Two compare registers
  - A compare match can potentially wake-up the device from low energy modes EM1 and EM2.
  - Second compare register can be top value for RTC.
  - Both compare channels can trigger LETIMER.
  - Compare match events are available to other peripherals through the Peripheral Reflex System (PRS).

20.3 Functional Description

The RTC is a 24-bit counter with two compare channels. The RTC is closely coupled with the LETIMER, and can be configured to trigger it on a compare match on one or both compare channels. An overview of the RTC module is shown in Figure 20.1 (p. 480) .

Figure 20.1. RTC Overview



20.3.1 Counter

The RTC is enabled by setting the EN bit in the RTC_CTRL register. It counts up as long as it is enabled, and will on an overflow simply wrap around and continue counting. The RTC is cleared when it is disabled. The timer value is both readable and writable and the RTC always starts counting from 0 when enabled. The value of the counter can be read or modified using the RTC_CNT register.

20.3.1.1 Clock Source

The RTC clock source and its prescaler value are defined in the Register Description section of the Clock Management Unit (CMU). The clock used by the RTC has a frequency given by Equation 20.1 (p. 480) .

RTC Frequency Equation

$$f_{RTC} = f_{LFACLK}/2^{RTC\_PRESC} \quad (20.1)$$

where f_{LFACLK} is the LFACLK frequency (32.768 kHz) and RTC_PRESC is a 4 bit value. Table 20.1 (p. 481) shows the time of overflow and resolution of the RTC at the available prescaler values.

To use this module, the LE interface clock must be enabled in CMU_HFCORECLKEN0 in addition to the module clock

Table 20.1. RTC Resolution Vs Overflow

RTC_PRESC	Resolution	Overflow
0	30,5 µs	512 s
1	61,0 µs	1024 s
2	122 µs	2048 s
3	244 µs	1,14 hours
4	488 µs	2,28 hours
5	977 µs	4,55 hours
6	1,95 ms	9,10 hours
7	3,91 ms	18,2 hours
8	7,81 ms	1,52 days
9	15,6 ms	3,03 days
10	31,25 ms	6,07 days
11	62,5 ms	12,1 days
12	0,125 s	24,3 days
13	0,25 s	48,5 days
14	0,5 s	97,1 days
15	1 s	194 days

20.3.2 Compare Channels

Two compare channels are available in the RTC. The compare values can be set by writing to the RTC compare channel registers RTC_COMPn, and when RTC_CNT is equal to one of these, the respective compare interrupt flag COMPn is set.

If COMP0TOP is set, the compare value set for compare channel 0 is used as a top value for the RTC, and the timer is cleared on a compare match with compare channel 0. If using the COMP0TOP setting, make sure to set this bit prior to or at the same time the EN bit is set. Setting COMP0TOP after the EN bit is set may cause unintended operation (i.e. if CNT > COMP0).

20.3.2.1 LETIMER Triggers

A compare event on either of the compare channels can start the LETIMER. See the LETIMER documentation for more information on this feature.

20.3.2.2 PRS Sources

Both the compare channels of the RTC can be used as PRS sources. They will generate a pulse lasting one RTC clock cycle on a compare match.

20.3.3 Interrupts

The interrupts generated by the RTC are combined into one interrupt vector. If interrupts for the RTC is enabled, an interrupt will be made if one or more of the interrupt flags in RTC_IF and their corresponding bits in RTC_IEN are set. Interrupt events are overflow and compare match on either compare channels. Clearing of an interrupt flag is performed by writing to the corresponding bit in the RTC_IFC register.

20.3.4 Debugrun

By default, the RTC is halted when code execution is halted from the debugger. By setting the DEBUGRUN bit in the RTC_CTRL register, the RTC will continue to run even when the debugger is halted.

20.3.5 Using the RTC in EM3

The RTC can be enabled all the way down to EM3 by using the ULFRCO as clock source. This is done by clearing CMU_LFCLKSEL_LFA and setting CMU_LFCLKSEL_LFAE to 1. This will make the RTC use the internal 1 kHz ultra low frequency RC oscillator (ULFRCO), consuming very little energy. Please note that the ULFRCO is not accurate over temperature and voltage, and it should be verified that the ULFRCO fulfills the timekeeping needs of the application before using this in the design.

20.3.6 Register access

This module is a Low Energy Peripheral, and supports immediate synchronization. For description regarding immediate synchronization, the reader is referred to Section 6.3.1.1 (p. 61) .

20.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	RTC_CTRL	RW	Control Register
0x004	RTC_CNT	RWH	Counter Value Register
0x008	RTC_COMP0	RW	Compare Value Register 0
0x00C	RTC_COMP1	RW	Compare Value Register 1
0x010	RTC_IF	R	Interrupt Flag Register
0x014	RTC_IFS	W1	Interrupt Flag Set Register
0x018	RTC_IFC	W1	Interrupt Flag Clear Register
0x01C	RTC_IEN	RW	Interrupt Enable Register
0x020	RTC_FREEZE	RW	Freeze Register
0x024	RTC_SYNCBUSY	R	Synchronization Busy Register

20.5 Register Description

20.5.1 RTC_CTRL - Control Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																															
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																														0	0	0
Access																														RW	RW	RW
Name																														COMP0TOP	DEBUGRUN	EN

Bit	Name	Reset	Access	Description
31:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	COMP0TOP	0	RW	**Compare Channel 0 is Top Value**

When set, the counter is cleared in the clock cycle after a compare match with compare channel 0.

Value	Mode	Description
0	DISABLE	The top value of the RTC is 16777215 (0xFFFFFF)
1	ENABLE	The top value of the RTC is given by COMP0

1	DEBUGRUN	0	RW	**Debug Mode Run Enable**

Set this bit to enable the RTC to keep running in debug.

Value	Description
0	RTC is frozen in debug mode
1	RTC is running in debug mode

0	EN	0	RW	**RTC Enable**

When this bit is set, the RTC is enabled and counts up. When cleared, the counter register CNT is reset.

20.5.2 RTC_CNT - Counter Value Register

Offset	Bit Position																															
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset									0x000000																							
Access									RWH																							
Name									CNT																							

Bit	Name	Reset	Access	Description
31:24	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
23:0	CNT	0x000000	RWH	**Counter Value**
	Gives access to the counter value of the RTC.			

20.5.3 RTC_COMP0 - Compare Value Register 0 (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																															
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset									0x000000																							
Access									RW																							
Name									COMP0																							

Bit	Name	Reset	Access	Description
31:24	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
23:0	COMP0	0x000000	RW	**Compare Value 0**
	A compare match event occurs when CNT is equal to this value. This event sets the COMP0 interrupt flag, and can be used to start the LETIMER. It is also available as a PRS signal.			

20.5.4 RTC_COMP1 - Compare Value Register 1 (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																														
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset									0x000000																							
Access									RW																							
Name									COMP1																							

Bit	Name	Reset	Access	Description
31:24	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
23:0	COMP1	0x000000	RW	**Compare Value 1**
	A compare match event occurs when CNT is equal to this value. This event sets COMP1 interrupt flag, and can be used to start the LETIMER. It is also available as a PRS signal.			

20.5.5 RTC_IF - Interrupt Flag Register

Offset	Bit Position																														
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																														0	0	0
Access																														R	R	R
Name																														COMP1	COMP0	OF

Bit	Name	Reset	Access	Description
31:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	COMP1	0	R	**Compare Match 1 Interrupt Flag**
	Set on a compare match between CNT and COMP1.			
1	COMP0	0	R	**Compare Match 0 Interrupt Flag**
	Set on a compare match between CNT and COMP0.			
0	OF	0	R	**Overflow Interrupt Flag**
	Set on a CNT value overflow.			

20.5.6 RTC_IFS - Interrupt Flag Set Register

Offset	Bit Position																														
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																														0	0	0
Access																														W1	W1	W1
Name																														COMP1	COMP0	OF

Bit	Name	Reset	Access	Description
31:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	COMP1	0	W1	**Set Compare match 1 Interrupt Flag**
	Write to 1 to set the COMP1 interrupt flag.			
1	COMP0	0	W1	**Set Compare match 0 Interrupt Flag**
	Write to 1 to set the COMP0 interrupt flag.			
0	OF	0	W1	**Set Overflow Interrupt Flag**
	Write to 1 to set the OF interrupt flag.			

20.5.7 RTC_IFC - Interrupt Flag Clear Register

Offset	Bit Position																													
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																														0	0	0
Access																														W1	W1	W1
Name																														COMP1	COMP0	OF

Bit	Name	Reset	Access	Description
31:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	COMP1	0	W1	**Clear Compare match 1 Interrupt Flag**
	Write to 1 to clear the COMP1 interrupt flag.			
1	COMP0	0	W1	**Clear Compare match 0 Interrupt Flag**
	Write to 1 to clear the COMP0 interrupt flag.			
0	OF	0	W1	**Clear Overflow Interrupt Flag**
	Write to 1 to clear the OF interrupt flag.			

20.5.8 RTC_IEN - Interrupt Enable Register

Offset	Bit Position																													
0x01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																														0	0	0
Access																														RW	RW	RW
Name																														COMP1	COMP0	OF

Bit	Name	Reset	Access	Description
31:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	COMP1	0	RW	**Compare Match 1 Interrupt Enable**
	Enable interrupt on compare match 1.			
1	COMP0	0	RW	**Compare Match 0 Interrupt Enable**
	Enable interrupt on compare match 0.			
0	OF	0	RW	**Overflow Interrupt Enable**

Bit	Name	Reset	Access	Description
	Enable interrupt on overflow.			

20.5.9 RTC_FREEZE - Freeze Register

Offset	Bit Position																														
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																RW
Name																																REGFREEZE

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	REGFREEZE	0	RW	**Register Update Freeze**
	When set, the update of the RTC is postponed until this bit is cleared. Use this bit to update several registers simultaneously.			

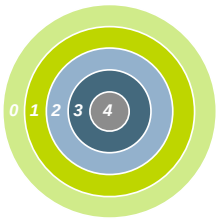
Value	Mode	Description
0	UPDATE	Each write access to an RTC register is updated into the Low Frequency domain as soon as possible.
1	FREEZE	The RTC is not updated with the new written value until the freeze bit is cleared.

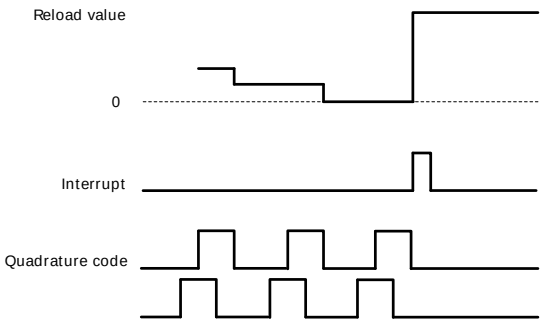
20.5.10 RTC_SYNCBUSY - Synchronization Busy Register

Offset	Bit Position																														
0x024	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																														0	0	0
Access																														R	R	R
Name																														COMP1	COMP0	CTRL

Bit	Name	Reset	Access	Description
31:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	COMP1	0	R	**COMP1 Register Busy**
	Set when the value written to COMP1 is being synchronized.			
1	COMP0	0	R	**COMP0 Register Busy**
	Set when the value written to COMP0 is being synchronized.			
0	CTRL	0	R	**CTRL Register Busy**
	Set when the value written to CTRL is being synchronized.			

21 PCNT - Pulse Counter





Quick Facts

What?

The Pulse Counter (PCNT) decodes incoming pulses. The module has a quadrature mode which may be used to decode the speed and direction of a mechanical shaft. PCNT can operate in EM0-EM3.

Why?

The PCNT generates an interrupt after a specific number of pulses (or rotations), eliminating the need for timing- or I/O interrupts and CPU processing to measure pulse widths, etc.

How?

21.1 Introduction

The Pulse Counter (PCNT) can be used for counting incoming pulses on a single input or to decode quadrature encoded inputs. It can run from the internal LFACLK (EM0-EM2) while counting pulses on the PCNTn_S0IN pin or using this pin as an external clock source (EM0-EM3) that runs both the PCNT counter and register access.

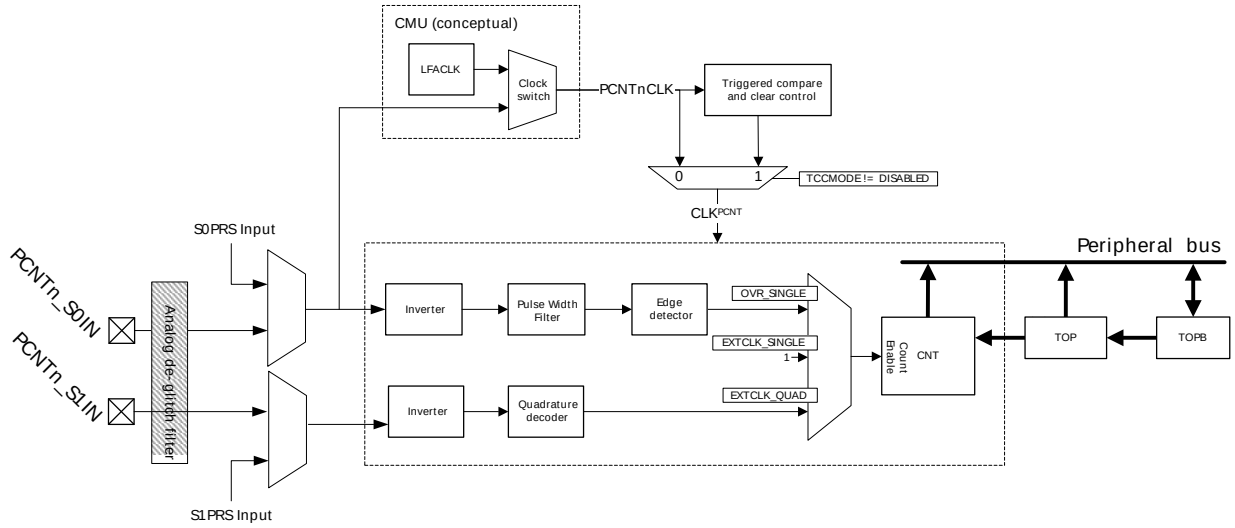
21.2 Features

-
- Auxiliary counter for counting a single direction
- Single input oversampling up/down counter mode (EM0-EM2)
- Externally clocked single input pulse up/down counter mode (EM0-EM3)
- Externally clocked quadrature decoder mode (EM0-EM3)
- Interrupt on counter underflow and overflow
- Interrupt when a direction change is detected (quadrature decoder mode only)
- Optional pulse width filter
- Optional input inversion/edge detect select
- PRS S0IN and S1IN input
- Asynchronously triggered compare and clear

21.3 Functional Description

An overview of the PCNT module is shown in Figure 21.1 (p. 489) .

Figure 21.1. PCNT Overview



21.3.1 Pulse Counter Modes

The pulse counter can operate in single input oversampling mode (OVSSINGLE), externally clocked single input counter mode (EXTCLKSINGLE) and externally clocked quadrature decoder mode (EXTCLKQUAD). The following sections describe operation of each of the three modes and how they are enabled. Input timing constraints are described in Section 21.3.6 (p. 493) and Section 21.3.7 (p. 493) .

21.3.1.1 Single Input Oversampling Mode

This mode is enabled by writing OVSSINGLE to the MODE field in the PCNTn_CTRL register and disabled by writing DISABLE to the same field. LFACLK is configured from the registers in the Clock Management Unit (CMU), Chapter 12 (p. 139) .

The optional pulse width filter is enabled by setting the FILT bit in the PCNTn_CTRL register. Additionally, the PCNTn_S0IN input may be inverted, so that falling edges are counted, by setting the EDGE bit in the PCNTn_CTRL register.

If S1CDIR is cleared, PCNTn_S0IN is the only observed input in this mode. The PCNTn_S0IN input is sampled by the LFACLK and the number of detected positive or negative edges on PCNTn_S0IN appears in PCNTn_CNT. The counter may be configured to count down by setting the CNTDIR bit in PCNTn_CTRL. Default is to count up.

The counting direction can also be controlled externally in this mode by setting S1CDIR in PCNTn_CTRL. This will make the input value on PCNTn_S1IN decide the direction counted on a PCNTn_S0IN edge. If PCNTn_S1IN is high, the count is done according to CNTDIR in PCNTn_CTRL. If low, the count direction is opposite.

21.3.1.2 Externally Clocked Single Input Counter Mode

This mode is enabled by writing EXTCLKSINGLE to the MODE field in the PCNTn_CTRL register and disabled by writing DISABLE to the same field. The external pin clock source must be configured from the registers in the CMU (Chapter 12 (p. 139)).

Positive edges on PCNTn_S0IN are used to clock the counter. Similar to the oversampled mode, PCNTn_S1IN is used to determine the count direction if S1CDIR in PCNTn_CTRL is set. If not, CNTDIR in PCNTn_CTRL solely defines count direction. As the LFACLK is not used in this mode, the PCNT module can operate in EM3.

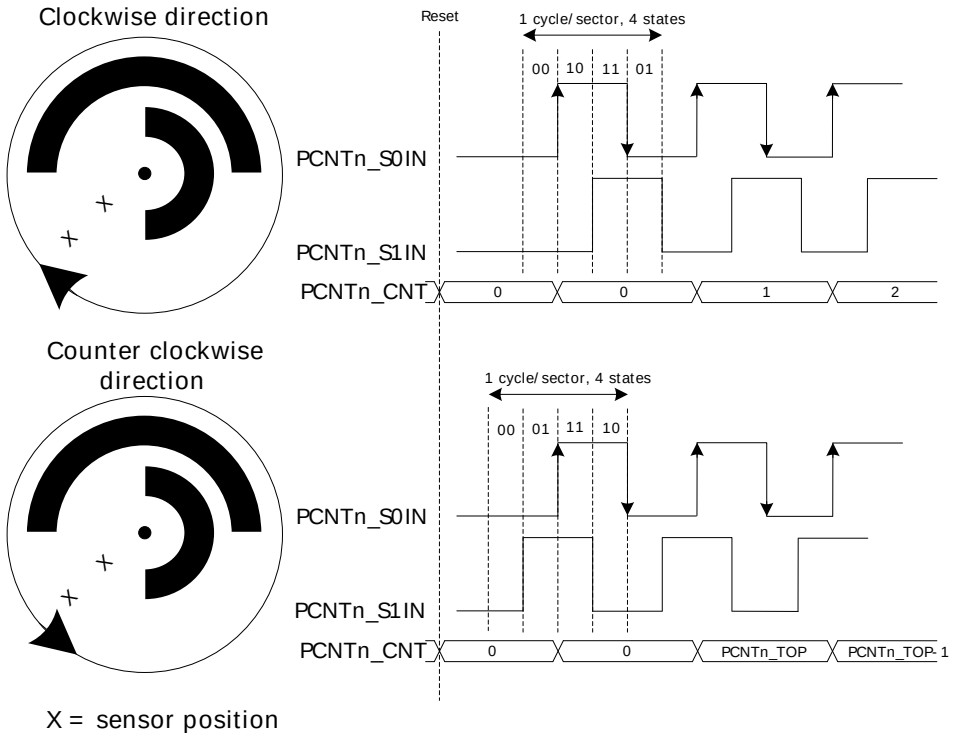
The digital pulse width filter is not available in this mode. The analog de-glitch filter in the GPIO pads is capable of removing some unwanted noise. However, this mode may be susceptible to spikes and unintended pulses from devices such as mechanical switches, and is therefore most suited to take input from electronic sensors etc. that generate single wire pulses.

21.3.1.3 Externally Clocked Quadrature Decoder Mode

This mode is enabled by writing EXTCLKQUAD to the MODE field in PCNTn_CTRL and disabled by writing DISABLE to the same field. The external pin clock source must be configured from the registers in the CMU, (Chapter 12 (p. 139)).

Both edges on PCNTn_S0IN pin are used to sample PCNTn_S1IN pin to decode the quadrature code. Consequently, this mode does not depend on the internal LFACLK and may be operated in EM3. A quadrature coded signal contains information about the relative speed and direction of a rotating shaft as illustrated by Figure 21.2 (p. 490) , hence the direction of the counter register PCNTn_CNT is controlled automatically.

Figure 21.2. PCNT Quadrature Coding



If PCNTn_S0IN leads PCNTn_S1IN in phase, the direction is clockwise, and if it lags in phase the direction is counter-clockwise. Although the direction is automatically detected, the detected direction may be inverted by writing 1 to the EDGE bit in the PCNTn_CTRL register. Default behavior is illustrated by Figure 21.2 (p. 490) .

The counter direction may be read from the DIR bit in the PCNTn_STATUS register. Additionally, the DIRCNG interrupt in the PCNTn_IF register is generated when a direction change is detected. When a change is detected, the DIR bit in the PCNTn_STATUS register must be read to determine the current new direction.

Note

The sector disc illustrated in the figure may be finer grained in some systems. Typically, they may generate 2-4 PCNTn_S0IN wave periods per 360° rotation.

The direction of the quadrature code and control of the counter is generated by the simple binary function outlined by Table 21.1 (p. 491) . Note that this function also filters some invalid inputs that may occur when the shaft changes direction or temporarily toggles direction.

Table 21.1. PCNT QUAD Mode Counter Control Function

Inputs		Control/Status	
S1IN posedge	S1IN negedge	Count Enable	CNTDIR status bit
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	0

Note

PCNTn_S1IN is sampled on both edges of PCNTn_S0IN.

21.3.2 Hysteresis

By default the pulse counter wraps to 0 when passing the configured top value, and wraps to the top value when counting down from 0. On these events, a system will likely want to wake up to store and track the overflow count. This is fine if the pulse counter is tracking a monotonic value or a value that does not change directions frequently. If you have the latter however, and the counter changes directions around the overflow/underflow point, the system will have to wake up a lot to keep track of the rotations, causing high current consumptions

To solve this, the pulse counter has a way of introducing hysteresis to the counter. When HYST in PCNTn_CTRL is set, the pulse counter will always wrap to TOP/2 on underflows and overflows. This takes the counter away from the area where it might overflow or underflow, removing the problem.

Given a starting value of 0 for the counter, the absolute count value when hysteresis is enabled can be calculated with the equations Equation 21.1 (p. 491) or Equation 21.2 (p. 491) , depending on whether the TOP value is even or odd.

Absolute position with hysteresis and even TOP value

$$CNT_{abs} = CNT - UF_{CNT} \times (TOP/2+1) + OF_{CNT} \times (TOP/2+1) \quad (21.1)$$

Absolute position with hysteresis and odd TOP value

$$CNT_{abs} = CNT - UF_{CNT} \times (TOP/2+1) + OF_{CNT} \times (TOP/2+2) \quad (21.2)$$

21.3.3 Auxiliary counter

To be able to keep explicit track of counting in one direction in addition to the regular counter which counts both up and down, the auxiliary counter can be used. The pulse counter can for instance be configured to keep track of the absolute rotation of the wheel, and at the same time the auxiliary counter can keep track of how much the wheel has reversed.

The auxiliary counter is enabled by configuring AUXCNTEV in PCNTn_CTRL. It will always count up, but it can be configured whether it should count up on up-events, down-events or both, keeping track of rotation either way or general movement. The value of the auxiliary counter can be read from the PCNTn_AUXCNT register.

Overflows on the auxiliary counter happen when the auxiliary counter passes the top value of the pulse counter, configured in PCNTn_TOP. In that event, the AUXOF interrupt flag is set, and the auxiliary counter wraps to 0.

As the auxiliary counter, the main counter can be configured to count only on certain events. This is done through CNTEV in PCNTn_CTRL, and it is possible like for the auxiliary counter, to make the main counter count on only up and down events. The difference between the counters is that where the auxiliary counter will only count up, the main counter will count up or down depending on the direction of the count event.

21.3.4 Triggered compare and clear

The pulse counter features triggered compare and clear. When enabled, a configurable trigger will induce a comparison between the main counter, PCNT_CNT, and the top value, PCNT_TOP. After the comparison, the counter is cleared. The trigger for a compare and clear event is configured in the TCCMODE bit-field in PCNT_CTRL. There are two options, LFA and PRS. If LFA is selected, the pulse counter will be compared with the top value, and cleared every 2^N LFA clock cycle. N is configured in TCCPRESC in PCNT_CTRL. If a PRS trigger is selected, the active PRS channel is configured in TCCPRSSEL in PCNT_CTRL. The PRS input can be inverted by setting TCCPRSPOL, triggering the compare and clear on the negative edge of the PRS input. The PRS input can also be used as a gate for the pulse counter clock. This is enabled by setting PRSGATEEN in PCNT_CTRL.

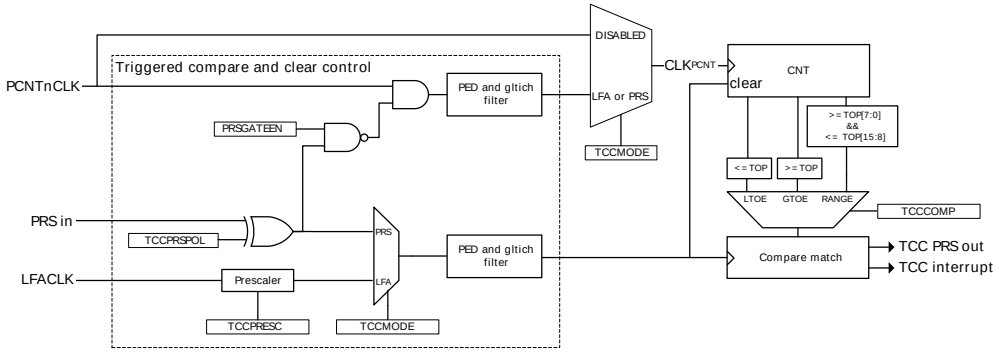
Note

When PRSGATEEN is set, the clock to the entire pulse counter will be gated by the PRS input, meaning that register writes will not take effect while the gated clock is inactive.

Comparison with PCNT_TOP can be performed in three ways; range, greater than or equal, and less than or equal. TCCCOMP in PCNT_CTRL configures comparison mode. Upon a compare match, the TCC interrupt is set, and the PRS output from the pulse counter is set. The PRS output will remain set until the next compare and clear event. Triggered compare and clear is intended for use when the pulse counter is configured to count up. In this mode, PCNT_CNT will not wrap to 0 when hitting PCNT_TOP, it will keep counting. In addition, the counter will not overflow, it will rather stop counting, just setting the overflow interrupt flag.

Figure 21.3 (p. 492) shows an overview of the control circuitry for triggered compare and clear. The control circuitry includes two positive edge detectors (PED) and glitch filters, used to generate clocks for the pulse counter. The two clock outputs are mutually exclusive: If both edge detectors receive a pulse at the same time, the output pulse from one of them will be postponed until the other edge detectors output pulse has completed.

Figure 21.3. PCNT Triggered compare and clear



Note

TCCMODE, TCCPRESC, PRSGATEEN, TCCPRSPOL, and TCCPRSSEL in PCNT_CTRL should only be altered when PCNT_CTRL_RSTEN is set.

21.3.5 Register Access

The counter-clock domain may be clocked externally. To update the counter-clock domain registers from software in this mode, 2-3 clock pulses on the external clock are needed to synchronize accesses

to the externally clocked domain. Clock source switching is controlled from the registers in the CMU (Chapter 12 (p. 139)).

When the RSTEN bit in the PCNTn_CTRL register is set to 1, the PCNT clock domain is asynchronously held in reset. The reset is synchronously released two PCNT clock edges after the RSTEN bit in the PCNTn_CTRL register is cleared by software. This asynchronous reset restores the reset values in PCNTn_TOP, PCNTn_CNT and other control registers in the PCNT clock domain.

AUXCNTRSTEN works in a similar manner as RSTEN, but only resetting the auxiliary counter, AUXCNT. Note that the auxiliary counter is also reset by RSTEN.

Since this module is a Low Energy Peripheral, and runs off a clock which is asynchronous to the HFCORECLK, special considerations must be taken when accessing registers. Please refer to Section 6.3 (p. 61) for a description on how to perform register accesses to Low Energy Peripherals.

Note

PCNTn_TOP and PCNTn_CNT are read-only registers. When writing to PCNTn_TOPB, make sure that the counter value, PCNTn_CNT, can not exceed the value written to PCNTn_TOPB within two clock cycles.

21.3.6 Clock Sources

The 32 kHz LFACLK is one of two possible clock sources. The clock select register is described in Chapter 12 (p. 139) . The default clock source is the LFACLK.

This PCNT module may also use PCNTn_S0IN as an external clock to clock the counter (EXTCLKSINGLE mode) and to sample PCNTn_S1IN (EXTCLKQUAD mode). Setup, hold and max frequency constraints for PCNTn_S0IN and PCNTn_S1IN for these modes are specified in the device datasheet.

To use this module, the LE interface clock must be enabled in CMU_HFCORECLKEN0, in addition to the module clock.

Note

PCNT Clock Domain Reset, RSTEN, should be set when changing clock source for PCNT. In addition to this, the PCNTn_SYNCBUSY value should be zero. If changing to an external clock source, the clock pin has to be enabled as input prior to de-asserting RSTEN. Changing clock source without asserting RSTEN results in undefined behaviour.

21.3.7 Input Filter

An optional pulse width filter is available in OVSSINGLE mode. The filter is enabled by writing 1 to the FILT bit in the PCNTn_CTRL register. When enabled, the high and low periods of PCNTn_S0IN must be stable for 5 consecutive clock cycles before the edge is passed to the edge detector.

In EXTCLKSINGLE and EXTCLKQUAD mode, there is no digital pulse width filter available.

21.3.8 Edge Polarity

The edge polarity can be set by configuring the EDGE bit in the PCNTn_CTRL register. When this bit is cleared, the pulse counter counts positive edges in OVSSINGLE mode and negative edges if the bit is set.

In EXTCLKQUAD mode, the EDGE bit in PCNTn_CTRL inverts the direction of the counter (which is automatically detected).

Note

The EDGE bit in PCNTn_CTRL has no effect in EXTCLKSINGLE mode.

21.3.9 PRS S0IN and S1IN Input

It is possible to receive input from PRS on both SOIN and S1IN by setting S0PRSEN or S1PRSEN in PCNTn_INPUT. The PRS channel used can be selected using S0PRSSEL in PCNTn_INPUT.

21.3.10 Interrupts

The interrupt generated by PCNT uses the PCNTn_INT interrupt vector. Software must read the PCNTn_IF register to determine which module interrupt that generated the vector invocation.

21.3.10.1 Underflow and Overflow Interrupts

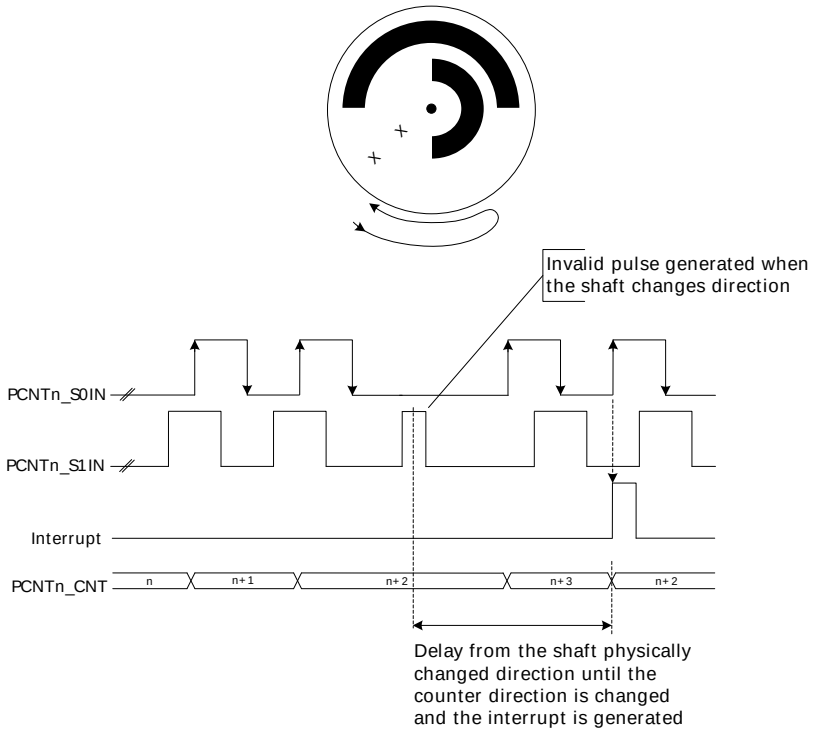
The underflow interrupt flag (UF) is set when the counter counts down from 0. I.e. when the value of the counter is 0 and a new pulse is received. The PCNTn_CNT register is loaded with the PCNTn_TOP value after this event.

The overflow interrupt flag (OF) is set when the counter counts up from the PCNTn_TOP (reload) value. I.e. if PCNTn_CNT = PCNTn_TOP and a new pulse is received. The PCNTn_CNT register is loaded with the value 0 after this event.

21.3.10.2 Direction Change Interrupt

The PCNTn_PCNT module sets the DIRCNG interrupt flag (PCNTn_IF register) when the direction of the quadrature code changes. The behavior of this interrupt is illustrated by Figure 21.4 (p. 494) .

Figure 21.4. PCNT Direction Change Interrupt (DIRCNG) Generation



21.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	PCNTn_CTRL	RW	Control Register
0x004	PCNTn_CMD	W1	Command Register
0x008	PCNTn_STATUS	R	Status Register
0x00C	PCNTn_CNT	R	Counter Value Register
0x010	PCNTn_TOP	R	Top Value Register
0x014	PCNTn_TOPB	RW	Top Value Buffer Register
0x018	PCNTn_IF	R	Interrupt Flag Register
0x01C	PCNTn_IFS	W1	Interrupt Flag Set Register
0x020	PCNTn_IFC	W1	Interrupt Flag Clear Register
0x024	PCNTn_IEN	RW	Interrupt Enable Register
0x028	PCNTn_ROUTE	RW	I/O Routing Register
0x02C	PCNTn_FREEZE	RW	Freeze Register
0x030	PCNTn_SYNCBUSY	R	Synchronization Busy Register
0x038	PCNTn_AUXCNT	RWH	Auxiliary Counter Value Register
0x03C	PCNTn_INPUT	RW	PCNT Input Register

21.5 Register Description

21.5.1 PCNTn_CTRL - Control Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x0			0	0	0x0			0x0				0x0				0x0				0x0		0	0		0	0	0	0	0	0x0	
Access	RW			RW	RW	RW			RW				RW				RW				RW		RW	RW		RW	RW	RW	RW	RW	RW	
Name	TCCPRSSEL			TCCPRSPOL	PRSGATEEN	TCCCOMP			TCCPRESC				TCCMODE				AUXCNTEV				CNTEV		S1CDIR	HYST		AUXCNTRSTEN	RSTEN	FILT	EDGE	CNTDIR	MODE	

Bit	Name	Reset	Access	Description
31:29	TCCPRSSEL	0x0	RW	**TCC PRS Channel Select**

Select PRS channel used as compare and clear trigger.

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected.
1	PRSCH1	PRS Channel 1 selected.
2	PRSCH2	PRS Channel 2 selected.
3	PRSCH3	PRS Channel 3 selected.
4	PRSCH4	PRS Channel 4 selected.
5	PRSCH5	PRS Channel 5 selected.

28	TCCPRSPOL	0	RW	**TCC PRS polarity select**

Configure which edge on the PRS input is used to trigger a compare and clear event

Bit	Name	Reset	Access	Description

Value	Mode	Description
0	RISING	Rising edge on PRS trigger compare and clear event.
1	FALLING	Falling edge on PRS trigger compare and clear event.

27 PRSGATEEN 0 RW **PRS gate enable**

When set, the clock input to the pulse counter will be gated when the selected PRS input is the inverse of TCCPRSPOL.

26:25 TCCCOMP 0x0 RW **Triggered compare and clear compare mode**

Selects the mode for comparison upon a compare and clear event.

Value	Mode	Description
0	LTOE	Compare match if PCNT_CNT is less than, or equal to PCNT_TOP.
1	GTOE	Compare match if PCNT_CNT is greater than or equal to PCNT_TOP.
2	RANGE	Compare match if PCNT_CNT is less than, or equal to PCNT_TOP[15:8]], and greater than, or equal to PCNT_TOP[7:0].

24 Reserved *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

23:22 TCCPRESC 0x0 RW **Set the LFA prescaler for triggered compare and clear**

Selects the prescaler value for LFA compare and clear events

Value	Mode	Description
0	DIV1	Compare and clear event each LFA cycle.
1	DIV2	Compare and clear performed on every other LFA cycle.
2	DIV4	Compare and clear performed on every 4th LFA cycle.
3	DIV8	Compare and clear performed on every 8th LFA cycle.

21:20 Reserved *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

19:18 TCCMODE 0x0 RW **Sets the mode for triggered compare and clear**

Selects whether compare and clear should be triggered on each LFA clock, or from PRS

Value	Mode	Description
0	DISABLED	Triggered compare and clear not enabled.
1	LFA	Compare and clear performed on each (optionally prescaled) LFA clock cycle.
2	PRS	Compare and clear performed on positive PRS edges.

17:16 Reserved *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

15:14 AUXCNTEV 0x0 RW **Controls when the auxiliary counter counts**

Selects whether the auxiliary counter responds to up-count events, down-count events or both

Value	Mode	Description
0	NONE	Never counts.
1	UP	Counts up on up-count events.
2	DOWN	Counts up on down-count events.
3	BOTH	Counts up on both up-count and down-count events.

13:12 Reserved *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

11:10 CNTEV 0x0 RW **Controls when the counter counts**

Selects whether the regular counter responds to up-count events, down-count events or both

Value	Mode	Description
0	BOTH	Counts up on up-count and down on down-count events.
1	UP	Only counts up on up-count events.
2	DOWN	Only counts down on down-count events.
3	NONE	Never counts.

9 S1CDIR 0 RW **Count direction determined by S1**

S1 gives the direction of counting when in the OVSSINGLE or EXTCLKSINGLE modes. When S1 is high, the count direction is given by CNTDIR, and when S1 is low, the count direction is the opposite

8 HYST 0 RW **Enable Hysteresis**

When hysteresis is enabled, the PCNT will always overflow and underflow to TOP/2.

Bit	Name	Reset	Access	Description
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	AUXCNTRSTEN	0	RW	**Enable AUXCNT Reset**
	The auxiliary counter, AUXCNT, is asynchronously held in reset when this bit is set. The reset is synchronously released two PCNT clock edges after this bit is cleared. If external clock used the reset should be performed by setting and clearing the bit without pending for SYNCBUSY bit.			
5	RSTEN	0	RW	**Enable PCNT Clock Domain Reset**
	The PCNT clock domain is asynchronously held in reset when this bit is set. The reset is synchronously released two PCNT clock edges after this bit is cleared. If external clock used the reset should be performed by setting and clearing the bit without pending for SYNCBUSY bit.			
4	FILT	0	RW	**Enable Digital Pulse Width Filter**
	The filter passes all high and low periods that are at least 5 clock cycles long. This filter is only available in OVSSINGLE mode.			
3	EDGE	0	RW	**Edge Select**
	Determines the polarity of the incoming edges. This bit should be written when PCNT is in DISABLE mode, otherwise the behavior is unpredictable. This bit is ignored in EXTCLKSINGLE mode.			

Value	Mode	Description
0	POS	Positive edges on the PCNTn_S0IN inputs are counted in OVSSINGLE mode.
1	NEG	Negative edges on the PCNTn_S0IN inputs are counted in OVSSINGLE mode, and the counter direction is inverted in EXTCLKQUAD mode.

2	CNTDIR	0	RW	**Non-Quadrature Mode Counter Direction Control**
	The direction of the counter must be set in the OVSSINGLE and EXTCLKSINGLE modes. This bit is ignored in EXTCLKQUAD mode as the direction is automatically detected.			

Value	Mode	Description
0	UP	Up counter mode.
1	DOWN	Down counter mode.

1:0	MODE	0x0	RW	**Mode Select**
	Selects the mode of operation. The corresponding clock source must be selected from the CMU.			

Value	Mode	Description
0	DISABLE	The module is disabled.
1	OVSSINGLE	Single input LFACLK oversampling mode (available in EM0-EM2).
2	EXTCLKSINGLE	Externally clocked single input counter mode (available in EM0-EM3).
3	EXTCLKQUAD	Externally clocked quadrature decoder mode (available in EM0-EM3).

21.5.2 PCNTn_CMD - Command Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																															
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															W1	W1
Name																															LTOPBIM	LCNTIM

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	LTOPBIM	0	W1	**Load TOPB Immediately**
	This bit has no effect since TOPB is not buffered and it is loaded directly into TOP.			
0	LCNTIM	0	W1	**Load CNT Immediately**
	Load PCNTn_TOP into PCNTn_CNT on the next counter clock cycle.			

21.5.3 PCNTn_STATUS - Status Register

Offset	Bit Position																														
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																R
Name																																DIR

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	DIR	0	R	**Current Counter Direction**

Current direction status of the counter. This bit is valid in EXTCLKQUAD mode only.

Value	Mode	Description
0	UP	Up counter mode (clockwise in EXTCLKQUAD mode with the NEDGE bit in PCNTn_CTRL set to 0).
1	DOWN	Down counter mode.

21.5.4 PCNTn_CNT - Counter Value Register

Offset	Bit Position																														
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	R															
Name																	CNT															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	CNT	0x0000	R	**Counter Value**

Gives read access to the counter.

21.5.5 PCNTn_TOP - Top Value Register

Offset	Bit Position																														
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x00FF															
Access																	R															
Name																	TOP															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	TOP	0x00FF	R	**Counter Top Value**
	When counting down, this value is reloaded into PCNTn_CNT when counting past 0. When counting up, 0 is written to the PCNTn_CNT register when counting past this value.			

21.5.6 PCNTn_TOPB - Top Value Buffer Register (Async Reg)

For more information about Asynchronous Registers please see Section 6.3 (p. 61) .

Offset	Bit Position																													
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x00FF															
Access																	RW															
Name																	TOPB															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	TOPB	0x00FF	RW	**Counter Top Buffer**
	Loaded automatically to TOP when written.			

21.5.7 PCNTn_IF - Interrupt Flag Register

Offset	Bit Position																													
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																												0	0	0	0	0
Access																												R	R	R	R	R
Name																												TCC	AUXOF	DIRCNG	OF	UF

Bit	Name	Reset	Access	Description
31:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	TCC	0	R	**Triggered compare Interrupt Read Flag**
	Set upon triggered compare match			
3	AUXOF	0	R	**Overflow Interrupt Read Flag**
	Set when an Auxiliary CNT overflow occurs			
2	DIRCNG	0	R	**Direction Change Detect Interrupt Flag**
	Set when the count direction changes. Set in EXTCLKQUAD mode only.			
1	OF	0	R	**Overflow Interrupt Read Flag**
	Set when a CNT overflow occurs			
0	UF	0	R	**Underflow Interrupt Read Flag**

Bit	Name	Reset	Access	Description
	Set when a CNT underflow occurs			

21.5.8 PCNTn_IFS - Interrupt Flag Set Register

Offset	Bit Position																														
0x01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																												0	0	0	0	0
Access																												W1	W1	W1	W1	W1
Name																												TCC	AUXOF	DIRCNG	OF	UF

Bit	Name	Reset	Access	Description
31:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	TCC	0	W1	**Triggered compare Interrupt Set**
	Write to 1 to set the triggered compare interrupt flag			
3	AUXOF	0	W1	**Auxiliary Overflow Interrupt Set**
	Write to 1 to set the auxiliary overflow interrupt flag			
2	DIRCNG	0	W1	**Direction Change Detect Interrupt Set**
	Write to 1 to set the direction change interrupt flag			
1	OF	0	W1	**Overflow Interrupt Set**
	Write to 1 to set the overflow interrupt flag			
0	UF	0	W1	**Underflow interrupt set**
	Write to 1 to set the underflow interrupt flag			

21.5.9 PCNTn_IFC - Interrupt Flag Clear Register

Offset	Bit Position																														
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																												0	0	0	0	0
Access																												W1	W1	W1	W1	W1
Name																												TCC	AUXOF	DIRCNG	OF	UF

Bit	Name	Reset	Access	Description
31:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	TCC	0	W1	**Triggered compare Interrupt Clear**
	Write to 1 to clear the triggered compare interrupt flag			
3	AUXOF	0	W1	**Auxiliary Overflow Interrupt Clear**
	Write to 1 to clear the auxiliary overflow interrupt flag			
2	DIRCNG	0	W1	**Direction Change Detect Interrupt Clear**
	Write to 1 to clear the direction change detect interrupt flag			
1	OF	0	W1	**Overflow Interrupt Clear**

Bit	Name	Reset	Access	Description
	Write to 1 to clear the overflow interrupt flag			
0	UF	0	W1	**Underflow Interrupt Clear**
	Write to 1 to clear the underflow interrupt flag			

21.5.10 PCNTn_IEN - Interrupt Enable Register

Offset	Bit Position																														
0x024	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																												0	0	0	0	0
Access																												RW	RW	RW	RW	RW
Name																												TCC	AUXOF	DIRCNG	OF	UF

Bit	Name	Reset	Access	Description
31:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	TCC	0	RW	**Triggered compare Interrupt Enable**
	Enable the triggered compare interrupt			
3	AUXOF	0	RW	**Auxiliary Overflow Interrupt Enable**
	Enable the auxiliary overflow interrupt			
2	DIRCNG	0	RW	**Direction Change Detect Interrupt Enable**
	Enable the direction change detect interrupt.			
1	OF	0	RW	**Overflow Interrupt Enable**
	Enable the overflow interrupt			
0	UF	0	RW	**Underflow Interrupt Enable**
	Enable the underflow interrupt			

21.5.11 PCNTn_ROUTE - I/O Routing Register

Offset	Bit Position																														
0x028	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0x0									
Access																							RW									
Name																							LOCATION									

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	LOCATION	0x0	RW	**I/O Location**

Defines the location of the PCNT input pins. E.g. PCNTn_S0#0, #1 or #2.

Value	Mode	Description
0	LOC0	Location 0
1	LOC1	Location 1

Bit	Name	Reset	Access	Description

Value	Mode	Description
2	LOC2	Location 2
3	LOC3	Location 3
4	LOC4	Location 4

7:0	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

21.5.12 PCNTn_FREEZE - Freeze Register

Offset	Bit Position																													
0x02C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																RW
Name																																REGFREEZE

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	REGFREEZE	0	RW	**Register Update Freeze**

When set, the update of the PCNT clock domain is postponed until this bit is cleared. Use this bit to update several registers simultaneously.

Value	Mode	Description
0	UPDATE	Each write access to a PCNT register is updated into the Low Frequency domain as soon as possible.
1	FREEZE	The PCNT clock domain is not updated with the new written value.

21.5.13 PCNTn_SYNCBUSY - Synchronization Busy Register

Offset	Bit Position																													
0x030	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																														0	0	0
Access																														R	R	R
Name																														TOPB	CMD	CTRL

Bit	Name	Reset	Access	Description
31:3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	TOPB	0	R	**TOPB Register Busy**

Set when the value written to TOPB is being synchronized.

1	CMD	0	R	**CMD Register Busy**

Set when the value written to CMD is being synchronized.

0	CTRL	0	R	**CTRL Register Busy**

Set when the value written to CTRL is being synchronized.

21.5.14 PCNTn_AUXCNT - Auxiliary Counter Value Register

Offset	Bit Position																															
0x038	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RWH															
Name																	AUXCNT															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	AUXCNT	0x0000	RWH	**Auxiliary Counter Value**
	Gives read access to the auxiliary counter.			

21.5.15 PCNTn_INPUT - PCNT Input Register

Offset	Bit Position																															
0x03C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																						0		0x0				0			0x0	
Access																						RW		RW				RW			RW	
Name																						S1PRSEN		S1PRSSEL				S0PRSEN			S0PRSSEL	

Bit	Name	Reset	Access	Description
31:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10	S1PRSEN	0	RW	**S1IN PRS Enable**
	When set, the PRS channel is selected as input to S1IN.			
9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8:6	S1PRSSEL	0x0	RW	**S1IN PRS Channel Select**
	Select PRS channel as input to S1IN.			

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected.
1	PRSCH1	PRS Channel 1 selected.
2	PRSCH2	PRS Channel 2 selected.
3	PRSCH3	PRS Channel 3 selected.
4	PRSCH4	PRS Channel 4 selected.
5	PRSCH5	PRS Channel 5 selected.

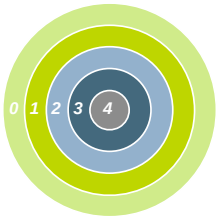
5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	S0PRSEN	0	RW	**S0IN PRS Enable**
	When set, the PRS channel is selected as input to S0IN.			
3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2:0	S0PRSSEL	0x0	RW	**S0IN PRS Channel Select**

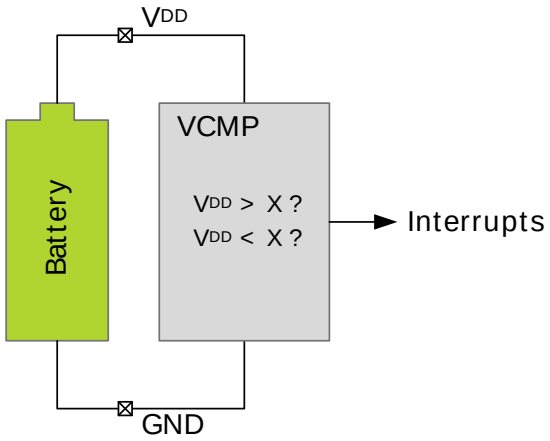
Bit	Name	Reset	Access	Description

Select PRS channel as input to S0IN.

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected.
1	PRSCH1	PRS Channel 1 selected.
2	PRSCH2	PRS Channel 2 selected.
3	PRSCH3	PRS Channel 3 selected.
4	PRSCH4	PRS Channel 4 selected.
5	PRSCH5	PRS Channel 5 selected.

22 VCMP - Voltage Comparator





Quick Facts

What?

The Voltage Supply Comparator (VCMP) monitors the input voltage supply and generates software interrupts on events using as little as 100 nA.

Why?

The VCMP can be used for simple power supply monitoring, e.g. for a battery level indicator.

How?

The scaled power supply is compared to a programmable reference voltage, and an interrupt can be generated when the supply is higher or lower than the reference. The VCMP can also be duty-cycled by software to further reduce the energy consumption.

22.1 Introduction

The Voltage Supply Comparator is used to monitor the supply voltage from software. An interrupt can be generated when the supply falls below or rises above a programmable threshold.

Note

Note that VCMP comes in addition to the Power-on Reset and Brown-out Detector peripherals, that both generate reset signals when the voltage supply is insufficient for reliable operation. VCMP does not generate reset, only interrupt. Also note that the ADC is capable of sampling the input voltage supply.

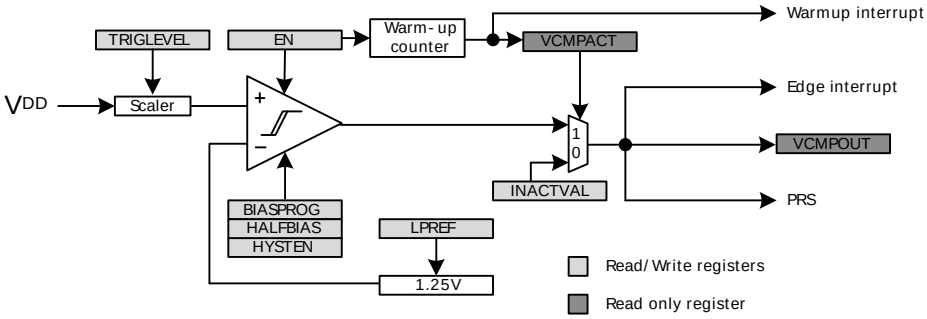
22.2 Features

- Voltage supply monitoring
- Scalable V_{DD} in 64 steps selectable as positive comparator input
- Internal 1.25 V bandgap reference
- Low power mode for internal V_{DD} and bandgap references
- Selectable hysteresis
  - 0 or ±20 mV
- Selectable response time
- Asynchronous interrupt generation on selectable edges
  - Rising edge
  - Falling edge
  - Rising and Falling edges
- Operational in EM0-EM3
- Comparator output direct on PRS
- Configurable output when inactive to avoid unwanted interrupts

22.3 Functional Description

An overview of the VCMP is shown in Figure 22.1 (p. 506) .

Figure 22.1. VCMP Overview



The comparator has two analog inputs, one positive and one negative. When the comparator is active, the output indicates which of the two input voltages is higher. When the voltage on the positive input is higher than the negative input voltage, the digital output is high and vice versa.

The output of the comparator can be read in the VCMPOUT bit in VCMP_STATUS. Configuration registers should only be changed while the comparator is disabled.

22.3.1 Warm-up Time

VCMP is enabled by setting the EN bit in VCMP_CTRL. When this bit is set, the comparator must stabilize before becoming active and the outputs can be used. This time period is called the warm-up time. The warm-up time is a configurable number of HFPERCLK cycles, set in WARMTIME, which should be set to at least 10 µs. When the comparator is enabled and warmed up, the VCMPACT bit in VCMP_STATUS will be set to indicate that the comparator is active.

As long as the comparator is not enabled or not warmed up, VCMPACT will be cleared and the comparator output value is set to the value in INACTVAL in VCMP_CTRL.

One should wait until the warm-up period is over before entering EM2 or EM3, otherwise no comparator interrupts will be detected. EM1 can still be entered during warm-up. After the warm-up period is completed, interrupts will be detected in EM2 and EM3.

22.3.2 Response Time

There is a delay from when the actual input voltage changes polarity, to when the output toggles. This period is called the response time and can be altered by increasing or decreasing the bias current to the comparator through the BIAS and HALFBIAS fields in VCMP_CTRL as shown in Table 22.1 (p. 506) . Setting a lower bias current will result in lower power consumption, but a longer response time.

Table 22.1. Bias Configuration

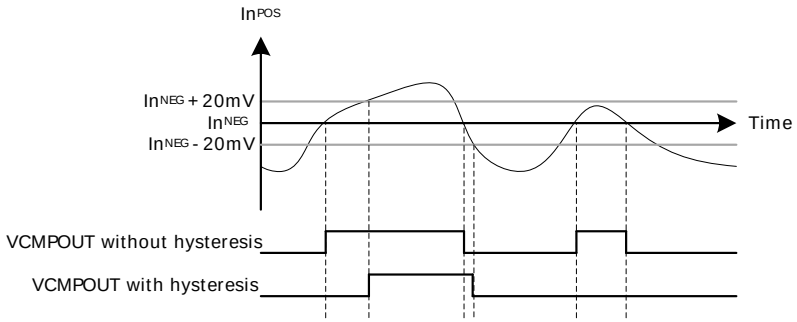
BIAS	Bias Current (µA)	
	HALFBIAS=0	HALFBIAS=1
0b0000	0.1	0.05
0b0001	0.2	0.1
0b0010	0.4	0.2
0b0011	0.6	0.3

BIAS	Bias Current (µA)	
	HALFBIAS=0	HALFBIAS=1
0b0100	0.8	0.4
0b0101	1.0	0.5
0b0110	1.2	0.6
0b0111	1.4	0.7
0b1000	2.0	1.0
0b1001	2.2	1.1
0b1010	2.4	1.2
0b1011	2.6	1.3
0b1100	2.8	1.4
0b1101	3.0	1.5
0b1110	3.2	1.6
0b1111	3.4	1.7

22.3.3 Hysteresis

In the voltage supply comparator, hysteresis can be enabled by setting HYSTEN in VCMP_CTRL. When HYSTEN is set, the digital output will not toggle until the positive input voltage is at least 20mV above or below the negative input voltage. This feature can be used to filter out uninteresting input fluctuations around zero and only show changes that are big enough to breach the hysteresis threshold.

Figure 22.2. VCMP 20 mV Hysteresis Enabled



22.3.4 Input Selection

The positive comparator input is always connected to the scaled power supply input. The negative comparator input is connected to the internal 1.25 V bandgap reference. The V_{DD} trigger level can be configured by setting the TRIGLEVEL field in VCMP_CTRL according to the following formula:

VCMP V_{DD} Trigger Level

$$V_{DD\ Trigger\ Level} = 1.667V + 0.034V \times TRIGLEVEL \quad (22.1)$$

A low power reference mode can be enabled by setting the LPREF bit in VCMP_INPUTSEL. In this mode, the power consumption in the reference buffer (V_{DD} and bandgap) is lowered at the cost of accuracy.

22.3.5 Interrupts and PRS Output

The VCMP includes an edge triggered interrupt flag (EDGE in VCMP_IF). If either IRISE and/or IFALL in VCMPn_CTRL is set, the EDGE interrupt flag will be set on rising and/or falling edge of the comparator output respectively. An interrupt request will be sent if the EDGE interrupt flag in VCMP_IF is set and enabled through the EDGE bit in VCMPn_IEN. The edge interrupt can also be used to wake up the device from EM3-EM1. VCMP also includes an interrupt flag, WARMUP in VCMP_IF, which is set when a warm-up sequence has finished. An interrupt request will be sent if the WARMUP interrupt flag in VCMP_IF is set and enabled through the WARMUP bit in VCMPn_IEN. The synchronized comparator output is also available as a PRS output signal.

22.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	VCMP_CTRL	RW	Control Register
0x004	VCMP_INPUTSEL	RW	Input Selection Register
0x008	VCMP_STATUS	R	Status Register
0x00C	VCMP_IEN	RW	Interrupt Enable Register
0x010	VCMP_IF	R	Interrupt Flag Register
0x014	VCMP_IFS	W1	Interrupt Flag Set Register
0x018	VCMP_IFC	W1	Interrupt Flag Clear Register

22.5 Register Description

22.5.1 VCMP_CTRL - Control Register

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset		1			0x7										0	0						0x0						0		0		0
Access		RW			RW										RW	RW						RW						RW		RW		RW
Name		HALFBIAS			BIASPROG										IFALL	IRISE						WARMTIME						HYSTEN		INACTVAL		EN

Bit	Name	Reset	Access	Description
31	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
30	HALFBIAS	1	RW	**Half Bias Current**
	Set this bit to 1 to halve the bias current. Table 22.1 (p. 506) .			
29:28	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
27:24	BIASPROG	0x7	RW	**VCMP Bias Programming Value**
	These bits control the bias current level. Table 22.1 (p. 506) .			
23:18	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
17	IFALL	0	RW	**Falling Edge Interrupt Sense**
	Set this bit to 1 to set the EDGE interrupt flag on falling edges of comparator output.			
16	IRISE	0	RW	**Rising Edge Interrupt Sense**
	Set this bit to 1 to set the EDGE interrupt flag on rising edges of comparator output.			
15:11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	WARMTIME	0x0	RW	**Warm-Up Time**
	Set warm-up time			

Value	Mode	Description
0	4CYCLES	4 HFPERCLK cycles
1	8CYCLES	8 HFPERCLK cycles
2	16CYCLES	16 HFPERCLK cycles
3	32CYCLES	32 HFPERCLK cycles
4	64CYCLES	64 HFPERCLK cycles
5	128CYCLES	128 HFPERCLK cycles
6	256CYCLES	256 HFPERCLK cycles

Bit	Name	Reset	Access	Description

Value	Mode	Description
7	512CYCLES	512 HFPERCLK cycles

7:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	HYSTEN	0	RW	**Hysteresis Enable**

Enable hysteresis.

Value	Description
0	No hysteresis
1	+-20 mV hysteresis

3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	INACTVAL	0	RW	**Inactive Value**

Configure the output value when the comparator is inactive.

1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	EN	0	RW	**Voltage Supply Comparator Enable**

Enable/disable voltage supply comparator.

22.5.2 VCMP_INPUTSEL - Input Selection Register

Offset	Bit Position																															
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																								0			0x00					
Access																								RW			RW					
Name																								LPREF			TRIGLEVEL					

Bit	Name	Reset	Access	Description
31:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8	LPREF	0	RW	**Low Power Reference**

Enable/disable low power mode for VDD and bandgap reference. When using this bit, always leave it as 0 during warm-up and then set it to 1 if desired when the warm-up is done.

7:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
5:0	TRIGLEVEL	0x00	RW	**Trigger Level**

Select VDD trigger level. V_{trig} = 1.667V+0.034V×TRIGLEVEL.

22.5.3 VCMP_STATUS - Status Register

Offset	Bit Position																															
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															R	R
Name																															VCMPOUT	VCMPACT

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	VCMPOUT	0	R	**Voltage Supply Comparator Output**
	Voltage supply comparator output value			
0	VCMPACT	0	R	**Voltage Supply Comparator Active**
	Voltage supply comparator active status.			

22.5.4 VCMP_IEN - Interrupt Enable Register

Offset	Bit Position																													
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															RW	RW
Name																															WARMUP	EDGE

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	WARMUP	0	RW	**Warm-up Interrupt Enable**
	Enable/disable interrupt on finished warm-up.			
0	EDGE	0	RW	**Edge Trigger Interrupt Enable**
	Enable/disable edge triggered interrupt.			

22.5.5 VCMP_IF - Interrupt Flag Register

Offset	Bit Position																													
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															R	R
Name																															WARMUP	EDGE

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	WARMUP	0	R	**Warm-up Interrupt Flag**
	Indicates that warm-up has finished.			
0	EDGE	0	R	**Edge Triggered Interrupt Flag**
	Indicates that there has been a rising and/or falling edge on the VCMP output.			

22.5.6 VCMP_IFS - Interrupt Flag Set Register

Offset	Bit Position																													
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															W1	W1
Name																															WARMUP	EDGE

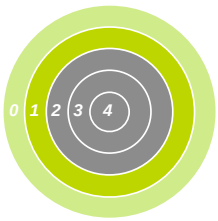
Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	WARMUP	0	W1	**Warm-up Interrupt Flag Set**
	Write to 1 to set warm-up finished interrupt flag			
0	EDGE	0	W1	**Edge Triggered Interrupt Flag Set**
	Write to 1 to set edge triggered interrupt flag			

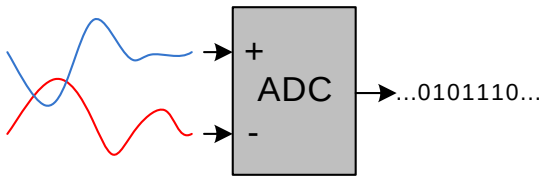
22.5.7 VCMP_IFC - Interrupt Flag Clear Register

Offset	Bit Position																													
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															W1	W1
Name																															WARMUP	EDGE

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	WARMUP	0	W1	**Warm-up Interrupt Flag Clear**
	Write to 1 to clear warm-up finished interrupt flag			
0	EDGE	0	W1	**Edge Triggered Interrupt Flag Clear**
	Write to 1 to clear edge triggered interrupt flag			

23 ADC - Analog to Digital Converter





Quick Facts

What?

The ADC is used to convert analog signals into a digital representation and features 8 external input channels

Why?

In many applications there is a need to measure analog signals and record them in a digital representation, without exhausting your energy source.

How?

A low power Successive Approximation Register ADC samples up to 8 input channels in a programmable sequence. With the help of PRS and DMA, the ADC can operate without CPU intervention, minimizing the number of powered up resources. The ADC can further be duty-cycled to reduce the energy consumption.

23.1 Introduction

The ADC is a Successive Approximation Register (SAR) architecture, with a resolution of up to 12 bits at up to one million samples per second. The integrated input mux can select inputs from 8 external pins and 6 internal signals.

23.2 Features

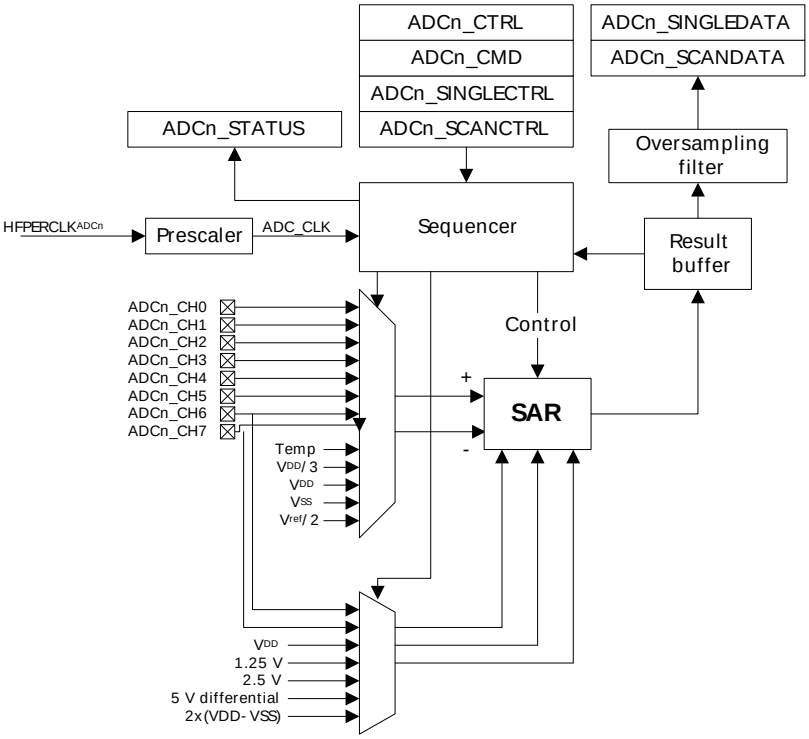
- Programmable resolution (6/8/12-bit)
  - 13 prescaled clock (ADC_CLK) cycles per conversion
  - Maximum 1 MSPS @ 12-bit
  - Maximum 1.86 MSPS @ 6-bit
- Configurable acquisition time
- Integrated prescaler
  - Selectable clock division factor from 1 to 128
- 13 MHz to 32 kHz allowed for ADC_CLK
- 18 input channels
  - 8 external single ended channels
  - 6 internal single ended channels
    - Including temperature sensor
  - 4 external differential channels
- Integrated input filter
  - Low pass RC filter
  - Decoupling capacitor
- Left or right adjusted results
  - Results in 2’s complement representation
  - Differential results sign extended to 32-bit results

- Programmable scan sequence
  - Up to 8 configurable samples in scan sequence
  - Mask to select which pins are included in the sequence
  - Triggered by software or PRS input
  - One shot or repetitive mode
  - Oversampling available
  - Overflow interrupt flag set when overwriting unread results
  - Conversion tailgating support for predictable periodic scans
- Programmable single conversion
  - Triggered by software or PRS input
  - Can be interleaved between two scan sequences
  - One shot or repetitive mode
  - Oversampling available
  - Overflow interrupt flag set when overwriting unread results
- Hardware oversampling support
  - 1st order accumulate and dump filter
  - From 2 to 4096 oversampling ratio (OSR)
  - Results in 16-bit representation
  - Enabled individually for scan sequence and single sample mode
  - Common OSR select
- Individually selectable voltage reference for scan and single mode
  - Internal 1.25V reference
  - Internal 2.5V reference
  - V_{DD}
  - Internal 5 V differential reference
  - Single ended external reference
  - Differential external reference
  - Unbuffered $2xV_{DD}$
- Support for offset and gain calibration
- Interrupt generation and/or DMA request
  - Finished single conversion
  - Finished scan conversion
  - Single conversion results overflow
  - Scan sequence results overflow

23.3 Functional Description

An overview of the ADC is shown in Figure 23.1 (p. 515) .

Figure 23.1. ADC Overview



23.3.1 Clock Selection

The ADC has an internal prescaler (PRESC bits in ADCn_CTRL) which can divide the peripheral clock (HFPERCLK) by any factor between 1 and 128. Note that the resulting ADC_CLK should not be set to a higher frequency than 13 MHz and not lower than 32 kHz.

23.3.2 Conversions

A conversion consists of two phases. The input is sampled in the acquisition phase before it is converted to digital representation during the approximation phase. The acquisition time can be configured independently for scan and single conversions (see Section 23.3.7 (p. 519)) by setting AT in ADCn_SINGLECTRL/ADCn_SCANCTRL. The acquisition times can be set to any integer power of 2 from 1 to 256 ADC_CLK cycles.

Note

For high impedance sources the acquisition time should be adjusted to allow enough time for the internal sample capacitor to fully charge. The minimum acquisition time for the internal temperature sensor and $V_{dd}/3$ is given in the electrical characteristics for the device.

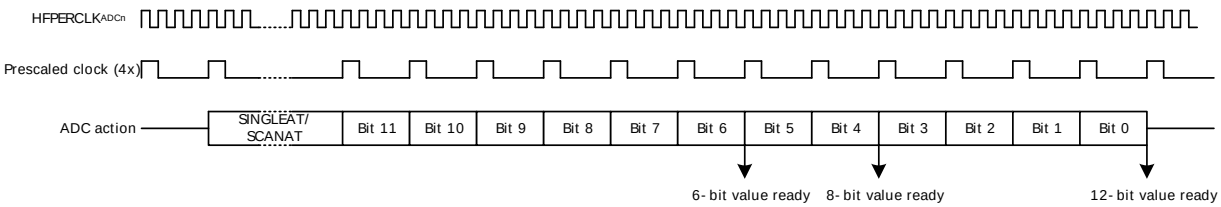
The analog to digital converter core uses one clock cycle per output bit in the approximation phase.

ADC Total Conversion Time (in ADC_CLK cycles) Per Output

$$T_{conv} = (T_A + N) \times OSR \quad (23.1)$$

T_A equals the number of acquisition cycles and N is the resolution. OSR is the oversampling ratio (see Section 23.3.7.7 (p. 521)). The minimum conversion time is 7 ADC_CYCLES with 6 bit resolution and 13 ADC_CYCLES with 12 bit resolution. The maximum conversion time is 1097728 ADC_CYCLES with the longest acquisition time, 12 bit resolution and highest oversampling rate.

Figure 23.2. ADC Conversion Timing



23.3.3 Warm-up Time

The ADC needs to be warmed up some time before a conversion can take place. This time period is called the warm-up time. When enabling the ADC or changing references between samples, the ADC is automatically warmed up for 1µs and an additional 5 µs if the bandgap is selected as reference.

Normally, the ADC will be warmed up only when samples are requested and is shut off when there are no more samples waiting. However, if lower latency is needed, configuring the WARMUPMODE field in ADCn_CTRL allows the ADC and/or reference to stay warm between samples, eliminating the need for warm-up. Figure 23.3 (p. 517) shows the analog power consumption in scenarios using the different WARMUPMODE settings.

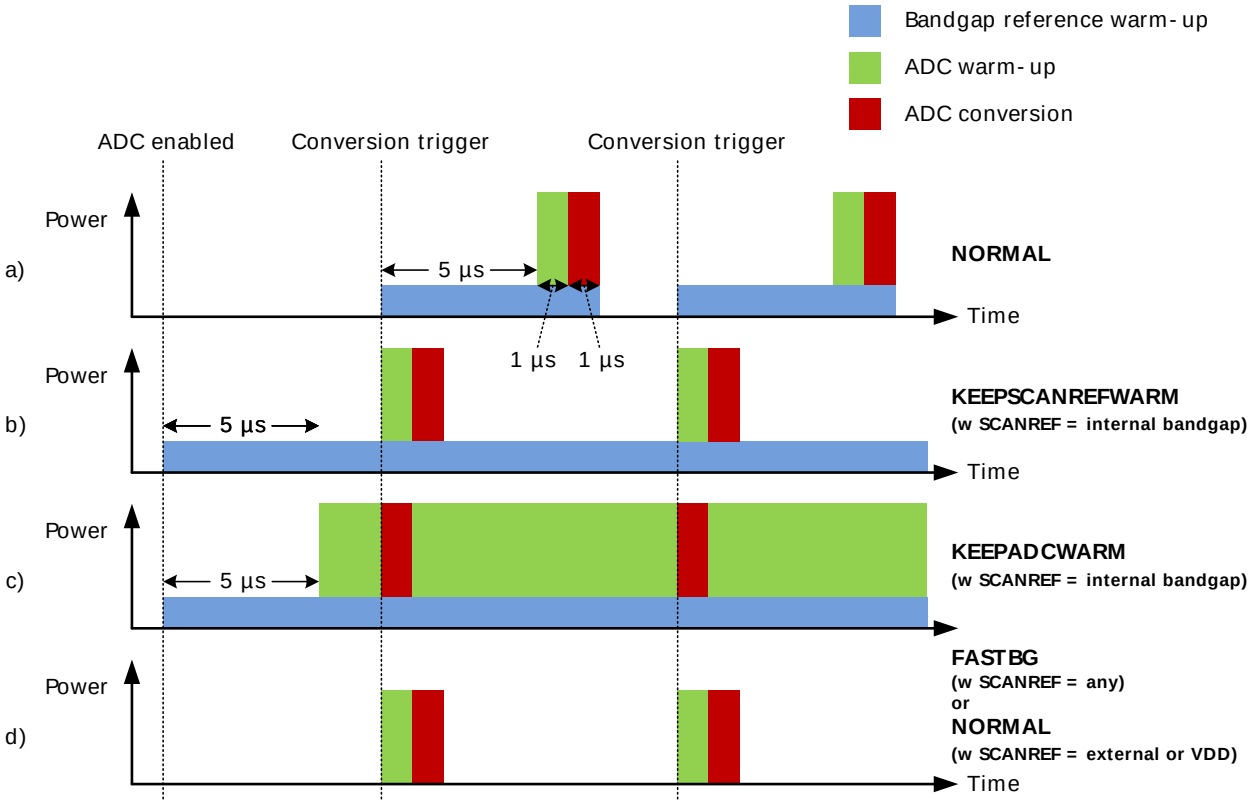
Only the bandgap reference selected for scan mode can be kept warm. If a different bandgap reference is selected for single mode, the warm-up time still applies.

- NORMAL: ADC and references are shut off when there are no samples waiting. a) in Figure 23.3 (p. 517) shows this mode used with an internal bandgap reference. Figure d) shows this mode when using VDD or an external reference.
- FASTBG: Bandgap warm-up is eliminated, but with reduced reference accuracy. d) in Figure 23.3 (p. 517) shows this mode used with an internal bandgap reference.
- KEEPSCANREFWARM: The reference selected for scan mode is kept warm. The ADC will still need to be warmed up before conversion. b) in Figure 23.3 (p. 517) shows this mode used with an internal bandgap reference.
- KEEPADCWARM: The ADC and the reference selected for scan mode is kept warm. c) in Figure 23.3 (p. 517) shows this mode used with an internal bandgap reference.

The minimum warm-up times are given in µs. The timing is done automatically by the ADC, given that a proper time base is given in the TIMEBASE bits in ADCn_CTRL. The TIMEBASE must be set to the number of HFPERCLK which corresponds to at least 1 µs. The TIMEBASE only affects the timing of the warm-up sequence and not the ADC_CLK.

When entering Energy Modes 2 or 3, the ADC must be stopped and WARMUPMODE in ADCn_CTRL written to 0.

Figure 23.3. ADC Analog Power Consumption With Different WARMUPMODE Settings



23.3.4 Input Selection

The ADC is connected to 8 external input pins, which can be selected as 8 different single ended inputs or 4 differential inputs. In addition, 6 single ended internal inputs can be selected. The available selections are given in the register description for ADCn_SINGLECTRL and ADCn_SCANCTRL.

For offset calibration purposes it is possible to internally short the differential ADC inputs and thereby measure a 0 V differential. Differential 0 V is selected by writing the DIFF bit to 1 and INPUTSEL to 4 in ADCn_SINGLECTRL. Calibration is described in detail in Section 23.3.10 (p. 522) .

Note

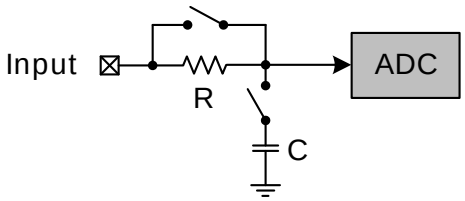
When VDD/3 is sampled, the acquisition time should be above a lower limit. The reader is referred to the datasheet for minimum VDD/3 acquisition time.

23.3.4.1 Input Filtering

The selected input signal can be filtered, either through an internal low pass RC filter or an internal decoupling capacitor. The different filter configurations can be enabled through the LPFMODE bits in ADCn_CTRL. For maximum SNR, LPFMODE is recommended set to DECAP, with a cutoff frequency of 31.5 MHz.

The RC input filter configuration is given in Figure 23.4 (p. 518). The resistance and capacitance values are given in the electrical characteristics for the device, named $R_{ADCFILT}$ and $C_{ADCFILT}$ respectively.

Figure 23.4. ADC RC Input Filter Configuration



23.3.4.2 Temperature Measurement

The ADC includes an internal temperature sensor. This sensor is characterized during production and the temperature readout from the ADC at production temperature, ADC0_TEMP_0_READ_1V25, is given in the Device Information (DI) page. The production temperature, CAL_TEMP_0, is also given in this page. The temperature gradient, TGRAD_ADCTH (mV/degree Celsius), for the sensor is found in the datasheet for the devices. By selecting 1.25 V internal reference and measuring the internal temperature sensor with 12 bit resolution, the temperature can be calculated according to the following formula:

ADC Temperature Measurement

$$T_{CELSIUS}=CAL\_TEMP\_0-(ADC0\_TEMP\_0\_READ\_1V25-ADC\_result)\times Vref/(4096\times TGRAD\_ADCTH) \quad (23.2)$$

Note

The minimum acquisition time for the temperature reference is found in the electrical characteristics for the device.

23.3.5 Reference Selection

The reference voltage can be selected from these sources:

- 1.25 V internal bandgap.
- 2.5 V internal bandgap.
- V_{DD}.
- 5 V internal differential bandgap.
- External single ended input from Ch. 6.
- Differential input, 2x(Ch. 6 - Ch. 7).
- Unbuffered $2xV_{DD}$.
- The 2.5 V reference needs a supply voltage higher than 2.5 V.
- The differential 5 V reference needs a supply voltage higher than 2.75 V.

Since the $2xV_{DD}$ differential reference is unbuffered, it is directly connected to the ADC supply voltage and more susceptible to supply noise. The V_{DD} reference is buffered both in single ended and differential mode.

If a differential reference with a larger range than the supply voltage is combined with single ended measurements, for instance the 5 V internal reference, the full ADC range will not be available because the maximum input voltage is limited by the maximum electrical ratings.

Note

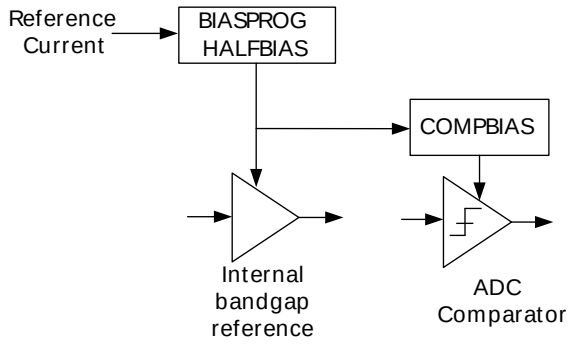
Single ended measurements with the external differential reference are not supported.

23.3.6 Programming of Bias Current

The bias current of the bandgap reference and the ADC comparator can be scaled by the BIASPROG, HALFBIAS and COMPBIAS bit fields of the ADCn_BIASPROG register. The BIASPROG and HALFBIAS

bitfields scale the current of ADC bandgap reference, and the COMPBIAS bits provide an additional bias programming for the ADC comparator as illustrated in Figure 23.5 (p. 519) . The electrical characteristics given in the datasheet require the bias configuration to be set to the default values, where no other bias values are given.

Figure 23.5. ADC Bias Programming



The minimum value of the BIASPROG and COMPBIAS bitfields of the ADCn_BIASPROG register (i.e. BIASPROG=0b0000, COMPBIAS=0b0000) represent the minimum bias currents. Similarly BIASPROG=0b1111 and COMPBIAS=0b1111 represent the maximum bias currents. Additionally, the bias current defined by the BIASPROG setting can be halved by setting the HALFBIAS bit of the ADCn_BIASPROG register.

The bias current settings should only be changed while the ADC is disabled.

23.3.7 ADC Modes

The ADC contains two separate programmable modes, one single sample mode and one scan mode. Both modes have separate configuration and result registers and can be set up to run only once per trigger or repetitively. The scan mode has priority over the single sample mode. However, if scan sequence is running, a triggered single sample will be interleaved between two scan samples.

23.3.7.1 Single Sample Mode

The single sample mode can be used to convert a single sample either once per trigger or repetitively. The configuration of the single sample mode is done in the ADCn_SINGLECTRL register and the results are found in the ADCn_SINGLEDATA register. The SINGLEDV bit in ADCn_STATUS is set high when there is valid data in the result register and is cleared when the data is read. The single mode results can also be read through ADCn_SINGLEDATAP without SINGLEDV being cleared. DIFF in ADCn_SINGLECTRL selects whether differential or single ended inputs are used and INPUTSEL selects input pin(s).

23.3.7.2 Scan mode

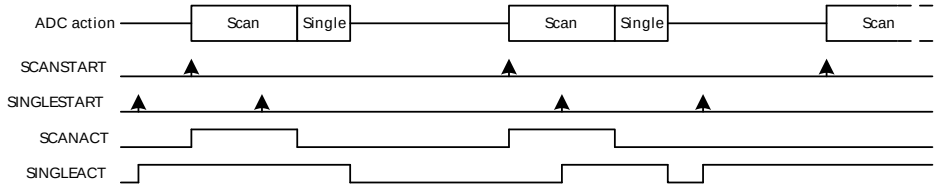
The scan mode is used to perform sweeps of the inputs. The configuration of the scan sequence is done in the ADCn_SCANCTRL register and the results are found in the ADCn_SCANDATA register. The SCANDV bit in ADCn_STATUS is set high when there is valid data in the result register and is cleared when the data is read. The scan mode results can also be read through ADCn_SCANDATAP without SCANDV being cleared. The inputs included in the sequence are defined by a the mask in INPUTMASK in ADCn_SCANCTRL. When the scan sequence is triggered, the sequence samples all inputs that are included in the mask, starting at the lowest pin number. DIFF in ADCn_SCANCTRL selects whether single ended or differential inputs are used.

23.3.7.3 Conversion Tailgating

The scan sequence has priority over the single sample mode. However, a scan trigger will not interrupt in the middle of a single conversion. If a scan sequence is triggered by a timer on a periodic basis,

single sample just before a scan trigger can delay the start of the scan sequence, thus causing jitter in sample rate. To solve this, conversion tailgating can be chosen by setting TAILGATE in ADCn_CTRL. When this bit is set, any triggered single samples will wait for the next scan sequence to finish before activating (see Figure 23.6 (p. 520)). The single sample will then follow immediately after the scan sequence. In this way, the scan sequence will always start immediately when triggered, if the period between the scan triggers is big enough to allow any single samples that might be triggered to finish in between the scan sequences.

Figure 23.6. ADC Conversion Tailgating



23.3.7.4 Conversion Trigger

The conversion modes can be activated by writing a 1 to the SINGLESTART or SCANSTART bit in the ADCn_CMD register. The conversions can be stopped by writing a 1 to the SINGLESTOP or SCANSTOP bit in the ADCn_CMD register. A START command will have priority over a stop command. When the ADC is stopped in the middle of a conversion, the result buffer is cleared. The SINGLEACT and SCANACT bits in ADCn_STATUS are set high when the modes are actively converting or have pending conversions.

It is also possible to trigger conversions from PRS signals. The system requires one HFPERCLK cycle pulses to trigger conversions. Setting PRSEN in ADCn_SINGLECTRL/ADCn_SCANCTRL enables triggering from PRS input. Which PRS channel to listen to is defined by PRSSEL in ADCn_SINGLECTRL/ADCn_SCANCTRL. When PRS trigger is selected, it is still possible to trigger the conversion from software. The reader is referred to the PRS datasheet for more information on how to set up the PRS channels.

Note

The conversion settings should not be changed while the ADC is running as this can lead to unpredictable behavior.

The prescaled clock phase is always reset by a triggered conversion as long as a conversion is not ongoing. This gives predictable latency from the time of the trigger to the time the conversion starts, regardless of when in the prescaled clock cycle the trigger occur.

23.3.7.5 Results

The results are presented in 2's complement form and the format for differential and single ended mode is given in Table 23.1 (p. 520) and Table 23.2 (p. 521). If differential mode is selected, the results are sign extended up to 32-bit (shown in Table 23.4 (p. 522)).

Table 23.1. ADC Single Ended Conversion

Input/Reference	Results	
	Binary	Hex value
1	111111111111	FFF
0.5	011111111111	7FF
1/4096	000000000001	001
0	000000000000	000

Table 23.2. ADC Differential Conversion

Input/Reference	Results	
	Binary	Hex value
0.5	011111111111	7FF
0.25	001111111111	3FF
1/2048	000000000001	001
0	000000000000	000
-1/2048	111111111111	FFF
-0.25	101111111111	BFF
-0.5	100000000000	800

23.3.7.6 Resolution

The ADC gives out 12-bit results, by default. However, if full 12-bit resolution is not needed, it is possible to speed up the conversion by selecting a lower resolution (N = 6 or 8 bits). For more information on the accuracy of the ADC, the reader is referred to the electrical characteristics section for the device.

23.3.7.7 Oversampling

To achieve higher accuracy, hardware oversampling can be enabled individually for each mode (Set RES in ADCn_SINGLECTRL/ADCn_SCANCTRL to 0x3). The oversampling rate (OVSRSEL in ADCn_CTRL) can be set to any integer power of 2 from 2 to 4096 and the configuration is shared between the scan and single sample mode (OVSRSEL field in ADCn_CTRL).

With oversampling, each selected input is sampled a number (given by the OVSR) of times, and the results are filtered by a first order accumulate and dump filter to form the end result. The data presented in the ADCn_SINGLEDATA and ADCn_SCANDATA registers are the direct contents of the accumulation register (sum of samples). However, if the oversampling ratio is set higher than 16x, the accumulated results are shifted to fit the MSB in bit 15 as shown in Table 23.3 (p. 521) .

Table 23.3. Oversampling Result Shifting and Resolution

Oversampling setting	# right shifts	Result Resolution # bits
2x	0	13
4x	0	14
8x	0	15
16x	0	16
32x	1	16
64x	2	16
128x	3	16
256x	4	16
512x	5	16
1024x	6	16
2048x	7	16
4096x	8	16

23.3.7.8 Adjustment

By default, all results are right adjusted, with the LSB of the result in bit position 0 (zero). In differential mode the signed bit is extended up to bit 31, but in single ended mode the bits above the result are read as 0. By setting ADJ in ADCn_SINGLECTRL/ADCn_SCANCTRL, the results are left adjusted as shown in Table 23.4 (p. 522) . When left adjusted, the MSB is always placed on bit 15 and sign extended to bit 31. All bits below the conversion result are read as 0 (zero).

Table 23.4. ADC Results Representation

Adjustment	Resolution	Bit																													
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Right	12	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	10	9	8	7	6	5	4	3	2	1	0
	8	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	6	5	4	3	2	1	0
	6	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	4	3	2	1	0
	OVS	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Left	12	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	10	9	8	7	6	5	4	3	2	1	0	-	-	-	-
	8	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	6	5	4	3	2	1	0	-	-	-	-	-	-	-	-	-
	6	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	4	3	2	1	0	-	-	-	-	-	-	-	-	-	-	-
	OVS	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

23.3.8 Interrupts, PRS Output

The single and scan modes have separate interrupt flags indicating finished conversions. Setting one of these flags will result in an ADC interrupt if the corresponding interrupt enable bit is set in ADCn_IEN.

In addition to the finished conversion flags, there is a scan and single sample result overflow flag which signalizes that a result from a scan sequence or single sample has been overwritten before being read.

A finished conversion will result in a one HFPERCLK cycle pulse which is output to the Peripheral Reflex System (PRS).

23.3.9 DMA Request

The ADC has two DMA request lines, SINGLE and SCAN, which are set when a single or scan conversion has completed. The request are cleared when the corresponding single or scan result register is read.

23.3.10 Calibration

The ADC supports offset and gain calibration to correct errors due to process and temperature variations. This must be done individually for each reference used. The ADC calibration (ADCn_CAL) register contains four register fields for calibrating offset and gain for both single and scan mode. The gain and offset calibration are done in single mode, but the resulting calibration values can be used for both single and scan mode.

Gain and offset for the 1V25, 2V5 and VDD references are calibrated during production and the calibration values for these can be found in the Device Information page. During reset, the gain and offset calibration registers are loaded with the production calibration values for the 1V25 reference.

The SCANGAIN and SINGLEGAIN calibration fields are not used when the unbuffered differential 2xVDD reference is selected.

The effects of changing the calibration register values are given in Table 23.5 (p. 523) . Step by step calibration procedures for offset and gain are given in Section 23.3.10.1 (p. 523) and Section 23.3.10.2 (p. 523) .

Table 23.5. Calibration Register Effect

Calibration Register	ADC Result	Calibration Binary Value	Calibration Hex Value
Offset	Lowest Output	0111111	3F
	Highest Output	1000000	40
Gain	Lowest Output	0000000	00
	Highest Output	1111111	7F

The offset calibration register expects a signed 2's complement value with negative effect. A high value gives a low ADC reading.

The gain calibration register expects an unsigned value with positive effect. A high value gives a high ADC reading.

23.3.10.1 Offset Calibration

Offset calibration must be performed prior to gain calibration. Follow these steps for the offset calibration in single mode:

1. Select wanted reference by setting the REF bitfield of the ADCn_SINGLECTRL register.
2. Set the AT bitfield of the ADCn_SINGLECTRL register to 16CYCLES.
3. Set the INPUTSEL bitfield of the ADCn_SINGLECTRL register to DIFF0, and set the DIFF bitfield to 1 for enabling differential input. Since the input voltage is 0, the expected ADC output is the half of the ADC code range as it is in differential mode.
4. A binary search is used to find the offset calibration value. Set the SINGLESTART bit in the ADCn_CMD register and read the ADCn_SINGLEDATA register. The result of the binary search is written to the SINGLEOFFSET field of the ADCn_CAL register.

23.3.10.2 Gain Calibration

Offset calibration must be performed prior to gain calibration. The Gain Calibration is done in the following manner:

1. Select an external ADC channel (a differential channel can also be used).
2. Apply an external voltage on the selected ADC input channel. This voltage should correspond to the top of the ADC range.
3. A binary search is used to find the gain calibration value. Set the SINGLESTART bit in the ADCn_CTRL register and read the ADCn_SINGLEDATA register. The target value is ideally the top of the ADC range, but it is recommended to use a value a couple of LSBs below in order to avoid overshooting. The result of the binary search is written to the SINGLEGAIN field of the ADCn_CAL register.

23.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	ADCn_CTRL	RW	Control Register
0x004	ADCn_CMD	W1	Command Register
0x008	ADCn_STATUS	R	Status Register
0x00C	ADCn_SINGLECTRL	RW	Single Sample Control Register
0x010	ADCn_SCANCTRL	RW	Scan Control Register
0x014	ADCn_IEN	RW	Interrupt Enable Register
0x018	ADCn_IF	R	Interrupt Flag Register
0x01C	ADCn_IFS	W1	Interrupt Flag Set Register
0x020	ADCn_IFC	W1	Interrupt Flag Clear Register
0x024	ADCn_SINGLEDATA	R	Single Conversion Result Data
0x028	ADCn_SCANDATA	R	Scan Conversion Result Data
0x02C	ADCn_SINGLEDATAP	R	Single Conversion Result Data Peek Register
0x030	ADCn_SCANDATAP	R	Scan Sequence Result Data Peek Register
0x034	ADCn_CAL	RW	Calibration Register
0x03C	ADCn_BIASPROG	RW	Bias Programming Register

23.5 Register Description

23.5.1 ADCn_CTRL - Control Register

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset				0	0x0					0x1F								0x00									0x0		0		0x0	
Access				RW	RW					RW								RW									RW		RW		RW	
Name				CHCONIDLE	OVSRSEL					TIMEBASE								PRESC									LPFMODE		TAILGATE		WARMUPMODE	

Bit	Name	Reset	Access	Description
31:29	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
28	CHCONIDLE	0	RW	**Input channel connected when ADC is IDLE**

Input channel Preference

Value	Mode	Description
0	DISCONNECT	Disconnect the input channel at the end of the conversion
1	KEEPCON	Keeps the current channel selected by INPUTSEL connected when ADC is IDLE

27:24	OVSRSEL	0x0	RW	**Oversample Rate Select**

Select oversampling rate. Oversampling must be enabled for each mode for this setting to take effect.

Value	Mode	Description
0	X2	2 samples for each conversion result
1	X4	4 samples for each conversion result

Bit	Name	Reset	Access	Description

Value	Mode	Description
2	X8	8 samples for each conversion result
3	X16	16 samples for each conversion result
4	X32	32 samples for each conversion result
5	X64	64 samples for each conversion result
6	X128	128 samples for each conversion result
7	X256	256 samples for each conversion result
8	X512	512 samples for each conversion result
9	X1024	1024 samples for each conversion result
10	X2048	2048 samples for each conversion result
11	X4096	4096 samples for each conversion result

23 *Reserved* *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

22:16 TIMEBASE 0x1F RW **Time Base**

Set time base used for ADC warm up sequence according to the HFPERCLK frequency. The time base is defined as a number of HFPERCLK cycles which should be set equal to or higher than 1us.

Value	Description
TIMEBASE	ADC warm-up is set to TIMEBASE+1 HFPERCLK clock cycles and bandgap warm-up is set to 5x(TIMEBASE+1) HFPERCLK cycles.

15 *Reserved* *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

14:8 PRESC 0x00 RW **Prescaler Setting**

Select clock division factor.

Value	Description
PRESC	Clock division factor of PRESC+1.

7:6 *Reserved* *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

5:4 LPFMODE 0x0 RW **Low Pass Filter Mode**

These bits control the filtering of the ADC input. Details on the filter characteristics can be found in the device datasheets.

Value	Mode	Description
0	BYPASS	No filter or decoupling capacitor
1	DECAP	On chip decoupling capacitor selected
2	RCFILT	On chip RC filter selected

3 TAILGATE 0 RW **Conversion Tailgating**

Enable/disable conversion tailgating.

Value	Description
0	Scan sequence has priority, but can be delayed by ongoing single samples.
1	Scan sequence has priority and single samples will only start immediately after scan sequence.

2 *Reserved* *To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

1:0 WARMUPMODE 0x0 RW **Warm-up Mode**

Select Warm-up Mode for ADC

Value	Mode	Description
0	NORMAL	ADC is shut down after each conversion
1	FASTBG	Bandgap references do not need warm up, but have reduced accuracy.
2	KEEPSCANREFWARM	Reference selected for scan mode is kept warm.
3	KEEPADCWARM	ADC is kept warmed up and scan reference is kept warm

23.5.2 ADCn_CMD - Command Register

Offset	Bit Position																															
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																													0	0	0	0
Access																													W1	W1	W1	W1
Name																													SCANSTOP	SCANSTART	SINGLESTOP	SINGLESTART

Bit	Name	Reset	Access	Description
31:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3	SCANSTOP	0	W1	**Scan Sequence Stop**
	Write a 1 to stop scan sequence.			
2	SCANSTART	0	W1	**Scan Sequence Start**
	Write a 1 to start scan sequence.			
1	SINGLESTOP	0	W1	**Single Conversion Stop**
	Write a 1 to stop single conversion.			
0	SINGLESTART	0	W1	**Single Conversion Start**
	Write to 1 to start single conversion.			

23.5.3 ADCn_STATUS - Status Register

Offset	Bit Position																															
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset						0x0									0	0				0			0	0							0	0
Access						R									R	R				R			R	R							R	R
Name						SCANDATASRC									SCANDV	SINGLEDV				WARM			SCANREFWARM	SINGLEREFWARM							SCANACT	SINGLEACT

Bit	Name	Reset	Access	Description
31:27	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
26:24	SCANDATASRC	0x0	R	**Scan Data Source**

This value indicates from which input channel the results in the ADCn_SCANDATA register originates.

Value	Mode	Description
0	CH0	Single ended mode: SCANDATA result originates from ADCn_CH0. Differential mode: SCANDATA result originates from ADCn_CH0-ADCn_CH1
1	CH1	Single ended mode: SCANDATA result originates from ADCn_CH1. Differential mode: SCANDATA result originates from ADCn_CH2_ADCn_CH3
2	CH2	Single ended mode: SCANDATA result originates from ADCn_CH2. Differential mode: SCANDATA result originates from ADCn_CH4-ADCn_CH5
3	CH3	Single ended mode: SCANDATA result originates from ADCn_CH3. Differential mode: SCANDATA result originates from ADCn_CH6-ADCn_CH7
4	CH4	SCANDATA result originates from ADCn_CH4
5	CH5	SCANDATA result originates from ADCn_CH5
6	CH6	SCANDATA result originates from ADCn_CH6
7	CH7	SCANDATA result originates from ADCn_CH7

Bit	Name	Reset	Access	Description
23:18	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
17	SCANDV	0	R	**Scan Data Valid**
	Scan conversion data is valid.			
16	SINGLEDV	0	R	**Single Sample Data Valid**
	Single conversion data is valid.			
15:13	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
12	WARM	0	R	**ADC Warmed Up**
	ADC is warmed up.			
11:10	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
9	SCANREFWARM	0	R	**Scan Reference Warmed Up**
	Reference selected for scan mode is warmed up.			
8	SINGLEREFWARM	0	R	**Single Reference Warmed Up**
	Reference selected for single mode is warmed up.			
7:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	SCANACT	0	R	**Scan Conversion Active**
	Scan sequence is active or has pending conversions.			
0	SINGLEACT	0	R	**Single Conversion Active**
	Single conversion is active or has pending conversions.			

23.5.4 ADCn_SINGLECTRL - Single Sample Control Register

Offset	Bit Position																													
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset		0x0						0	0x0					0x0							0x0						0x0			0	0	0
Access		RW						RW	RW					RW							RW						RW			RW	RW	RW
Name		PRSSEL						PRSEN	AT					REF							INPUTSEL						RES			ADJ	DIFF	REP

Bit	Name	Reset	Access	Description
31	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
30:28	PRSSEL	0x0	RW	**Single Sample PRS Trigger Select**
	Select PRS trigger for single sample.			

Value	Mode	Description
0	PRSCH0	PRS ch 0 triggers single sample
1	PRSCH1	PRS ch 1 triggers single sample
2	PRSCH2	PRS ch 2 triggers single sample
3	PRSCH3	PRS ch 3 triggers single sample
4	PRSCH4	PRS ch 4 triggers single sample
5	PRSCH5	PRS ch 5 triggers single sample

27:25	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
24	PRSEN	0	RW	**Single Sample PRS Trigger Enable**
	Enabled/disable PRS trigger of single sample.			

Value	Description
0	Single sample is not triggered by PRS input
1	Single sample is triggered by PRS input selected by PRSSEL

Bit	Name	Reset	Access	Description
23:20	AT	0x0	RW	**Single Sample Acquisition Time**

Select the acquisition time for single sample.

Value	Mode	Description
0	1CYCLE	1 ADC_CLK cycle acquisition time for single sample
1	2CYCLES	2 ADC_CLK cycles acquisition time for single sample
2	4CYCLES	4 ADC_CLK cycles acquisition time for single sample
3	8CYCLES	8 ADC_CLK cycles acquisition time for single sample
4	16CYCLES	16 ADC_CLK cycles acquisition time for single sample
5	32CYCLES	32 ADC_CLK cycles acquisition time for single sample
6	64CYCLES	64 ADC_CLK cycles acquisition time for single sample
7	128CYCLES	128 ADC_CLK cycles acquisition time for single sample
8	256CYCLES	256 ADC_CLK cycles acquisition time for single sample

19	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
18:16	REF	0x0	RW	**Single Sample Reference Selection**

Select reference to ADC single sample mode.

Value	Mode	Description
0	1V25	Internal 1.25 V reference
1	2V5	Internal 2.5 V reference
2	VDD	Buffered VDD
3	5VDIFF	Internal differential 5 V reference
4	EXTSINGLE	Single ended external reference from ADCn_CH6
5	2XEXTDIFF	Differential external reference, 2x(ADCn_CH6 - ADCn_CH7)
6	2XVDD	Unbuffered 2xVDD

15:12	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
11:8	INPUTSEL	0x0	RW	**Single Sample Input Selection**

Select input to ADC single sample mode in either single ended mode or differential mode.

DIFF = 0		
Mode	Value	Description
CH0	0	ADCn_CH0
CH1	1	ADCn_CH1
CH2	2	ADCn_CH2
CH3	3	ADCn_CH3
CH4	4	ADCn_CH4
CH5	5	ADCn_CH5
CH6	6	ADCn_CH6
CH7	7	ADCn_CH7
TEMP	8	Temperature reference
VDDDIV3	9	VDD/3
VDD	10	VDD
VSS	11	VSS
VREFDIV2	12	VREF/2
DAC0OUT0	13	DAC0 output 0
DAC0OUT1	14	DAC0 output 1
DIFF = 1		
Mode	Value	Description
CH0CH1	0	Positive input: ADCn_CH0 Negative input: ADCn_CH1
CH2CH3	1	Positive input: ADCn_CH2 Negative input: ADCn_CH3
CH4CH5	2	Positive input: ADCn_CH4 Negative input: ADCn_CH5
CH6CH7	3	Positive input: ADCn_CH6 Negative input: ADCn_CH7
DIFF0	4	Differential 0 (Short between positive and negative inputs)

7:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

Bit	Name	Reset	Access	Description
5:4	RES	0x0	RW	**Single Sample Resolution Select**

Select single sample conversion resolution.

Value	Mode	Description
0	12BIT	12-bit resolution
1	8BIT	8-bit resolution
2	6BIT	6-bit resolution
3	OVS	Oversampling enabled. Oversampling rate is set in OVSRSEL

3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

2	ADJ	0	RW	**Single Sample Result Adjustment**

Select single sample result adjustment.

Value	Mode	Description
0	RIGHT	Results are right adjusted
1	LEFT	Results are left adjusted

1	DIFF	0	RW	**Single Sample Differential Mode**

Select single ended or differential input.

Value	Description
0	Single ended input
1	Differential input

0	REP	0	RW	**Single Sample Repetitive Mode**

Enable/disable repetitive single samples.

Value	Description
0	Single conversion mode is deactivated after one conversion
1	Single conversion mode is converting continuously until SINGLESTOP is written

23.5.5 ADCn_SCANCTRL - Scan Control Register

Offset	Bit Position																													
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset		0x0						0	0x0					0x0			0x00										0x0			0	0	0
Access		RW						RW	RW					RW			RW										RW			RW	RW	RW
Name		PRSSEL						PRSEN	AT					REF			INPUTMASK										RES			ADJ	DIFF	REP

Bit	Name	Reset	Access	Description
31	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
30:28	PRSSEL	0x0	RW	**Scan Sequence PRS Trigger Select**

Select PRS trigger for scan sequence.

Value	Mode	Description
0	PRSCH0	PRS ch 0 triggers scan sequence
1	PRSCH1	PRS ch 1 triggers scan sequence
2	PRSCH2	PRS ch 2 triggers scan sequence
3	PRSCH3	PRS ch 3 triggers scan sequence
4	PRSCH4	PRS ch 4 triggers scan sequence
5	PRSCH5	PRS ch 5 triggers scan sequence

27:25	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*

Bit	Name	Reset	Access	Description
24	PRSEN	0	RW	**Scan Sequence PRS Trigger Enable**

Enabled/disable PRS trigger of scan sequence.

Value	Description
0	Scan sequence is not triggered by PRS input
1	Scan sequence is triggered by PRS input selected by PRSSEL

Bit	Name	Reset	Access	Description
23:20	AT	0x0	RW	**Scan Sample Acquisition Time**

Select the acquisition time for scan samples.

Value	Mode	Description
0	1CYCLE	1 ADC_CLK cycle acquisition time for scan samples
1	2CYCLES	2 ADC_CLK cycles acquisition time for scan samples
2	4CYCLES	4 ADC_CLK cycles acquisition time for scan samples
3	8CYCLES	8 ADC_CLK cycles acquisition time for scan samples
4	16CYCLES	16 ADC_CLK cycles acquisition time for scan samples
5	32CYCLES	32 ADC_CLK cycles acquisition time for scan samples
6	64CYCLES	64 ADC_CLK cycles acquisition time for scan samples
7	128CYCLES	128 ADC_CLK cycles acquisition time for scan samples
8	256CYCLES	256 ADC_CLK cycles acquisition time for scan samples

Bit	Name	Reset	Access	Description
19	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
18:16	REF	0x0	RW	**Scan Sequence Reference Selection**

Select reference to ADC scan sequence.

Value	Mode	Description
0	1V25	Internal 1.25 V reference
1	2V5	Internal 2.5 V reference
2	VDD	VDD
3	5VDIFF	Internal differential 5 V reference
4	EXTSINGLE	Single ended external reference from ADCn_CH6
5	2XEXTDIFF	Differential external reference, 2x(ADCn_CH6 - ADCn_CH7)
6	2XVDD	Unbuffered 2xVDD

Bit	Name	Reset	Access	Description
15:8	INPUTMASK	0x00	RW	**Scan Sequence Input Mask**

Set one or more bits in this mask to select which inputs are included the scan sequence in either single ended or differential mode.

DIFF = 0		
Mode	Value	Description
CH0	00000001	ADCn_CH0 included in mask
CH1	00000010	ADCn_CH1 included in mask
CH2	00000100	ADCn_CH2 included in mask
CH3	00001000	ADCn_CH3 included in mask
CH4	00010000	ADCn_CH4 included in mask
CH5	00100000	ADCn_CH5 included in mask
CH6	01000000	ADCn_CH6 included in mask
CH7	10000000	ADCn_CH7 included in mask
DIFF = 1		
Mode	Value	Description
CH0CH1	00000001	(Positive input: ADCn_CH0 Negative input: ADCn_CH1) included in mask
CH2CH3	00000010	(Positive input: ADCn_CH2 Negative input: ADCn_CH3) included in mask
CH4CH5	00000100	(Positive input: ADCn_CH4 Negative input: ADCn_CH5) included in mask
CH6CH7	00001000	(Positive input: ADCn_CH6 Negative input: ADCn_CH7) included in mask
	0001xxxx-1111xxxx	Reserved

Bit	Name	Reset	Access	Description
7:6	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		

Bit	Name	Reset	Access	Description
5:4	RES	0x0	RW	**Scan Sequence Resolution Select**

Select scan sequence conversion resolution.

Value	Mode	Description
0	12BIT	12-bit resolution
1	8BIT	8-bit resolution
2	6BIT	6-bit resolution
3	OVS	Oversampling enabled. Oversampling rate is set in OVSRSEL

3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2	ADJ	0	RW	**Scan Sequence Result Adjustment**

Select scan sequence result adjustment.

Value	Mode	Description
0	RIGHT	Results are right adjusted
1	LEFT	Results are left adjusted

1	DIFF	0	RW	**Scan Sequence Differential Mode**

Select single ended or differential input.

Value	Description
0	Single ended input
1	Differential input

0	REP	0	RW	**Scan Sequence Repetitive Mode**

Enable/disable repetitive scan sequence.

Value	Description
0	Scan conversion mode is deactivated after one sequence
1	Scan conversion mode is converting continuously until SCANSTOP is written

23.5.6 ADCn_IEN - Interrupt Enable Register

Offset	Bit Position																													
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0	0							0	0
Access																							RW	RW							RW	RW
Name																							SCANOF	SINGLEOF							SCAN	SINGLE

Bit	Name	Reset	Access	Description
31:10	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
9	SCANOF	0	RW	**Scan Result Overflow Interrupt Enable**

Enable/disable scan result overflow interrupt.

8	SINGLEOF	0	RW	**Single Result Overflow Interrupt Enable**

Enable/disable single result overflow interrupt.

7:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	SCAN	0	RW	**Scan Conversion Complete Interrupt Enable**

Enable/disable scan conversion complete interrupt.

0	SINGLE	0	RW	**Single Conversion Complete Interrupt Enable**

Enable/disable single conversion complete interrupt.

23.5.7 ADCn_IF - Interrupt Flag Register

Offset	Bit Position																													
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0	0							0	0
Access																							R	R							R	R
Name																							SCANOF	SINGLEOF							SCAN	SINGLE

Bit	Name	Reset	Access	Description
31:10	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
9	SCANOF	0	R	**Scan Result Overflow Interrupt Flag**
	Indicates scan result overflow when this bit is set.			
8	SINGLEOF	0	R	**Single Result Overflow Interrupt Flag**
	Indicates single result overflow when this bit is set.			
7:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	SCAN	0	R	**Scan Conversion Complete Interrupt Flag**
	Indicates scan conversion complete when this bit is set.			
0	SINGLE	0	R	**Single Conversion Complete Interrupt Flag**
	Indicates single conversion complete when this bit is set.			

23.5.8 ADCn_IFS - Interrupt Flag Set Register

Offset	Bit Position																													
0x01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0	0							0	0
Access																							W1	W1							W1	W1
Name																							SCANOF	SINGLEOF							SCAN	SINGLE

Bit	Name	Reset	Access	Description
31:10	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
9	SCANOF	0	W1	**Scan Result Overflow Interrupt Flag Set**
	Write to 1 to set scan result overflow interrupt flag			
8	SINGLEOF	0	W1	**Single Result Overflow Interrupt Flag Set**
	Write to 1 to set single result overflow interrupt flag.			
7:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	SCAN	0	W1	**Scan Conversion Complete Interrupt Flag Set**
	Write to 1 to set scan conversion complete interrupt flag.			
0	SINGLE	0	W1	**Single Conversion Complete Interrupt Flag Set**
	Write to 1 to set single conversion complete interrupt flag.			

23.5.9 ADCn_IFC - Interrupt Flag Clear Register

Offset	Bit Position																														
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																							0	0							0	0
Access																							W1	W1							W1	W1
Name																							SCANOF	SINGLEOF							SCAN	SINGLE

Bit	Name	Reset	Access	Description
31:10	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
9	SCANOF	0	W1	**Scan Result Overflow Interrupt Flag Clear**
	Write to 1 to clear scan result overflow interrupt flag.			
8	SINGLEOF	0	W1	**Single Result Overflow Interrupt Flag Clear**
	Write to 1 to clear single result overflow interrupt flag.			
7:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	SCAN	0	W1	**Scan Conversion Complete Interrupt Flag Clear**
	Write to 1 to clear scan conversion complete interrupt flag.			
0	SINGLE	0	W1	**Single Conversion Complete Interrupt Flag Clear**
	Write to 1 to clear single conversion complete interrupt flag.			

23.5.10 ADCn_SINGLEDATA - Single Conversion Result Data

Offset	Bit Position																														
0x024	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	R																															
Name	DATA																															

Bit	Name	Reset	Access	Description
31:0	DATA	0x00000000	R	**Single Conversion Result Data**
	The register holds the results from the last single conversion. Reading this field clears the SINGLEDV bit in the ADCn_STATUS register.			

23.5.11 ADCn_SCANDATA - Scan Conversion Result Data

Offset	Bit Position																														
0x028	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	R																															
Name	DATA																															

Bit	Name	Reset	Access	Description
31:0	DATA	0x00000000	R	**Scan Conversion Result Data**
	The register holds the results from the last scan conversion. Reading this field clears the SCANDV bit in the ADCn_STATUS register.			

23.5.12 ADCn_SINGLEDATAP - Single Conversion Result Data Peek Register

Offset	Bit Position																														
0x02C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	R																															
Name	DATAP																															

Bit	Name	Reset	Access	Description
31:0	DATAP	0x00000000	R	**Single Conversion Result Data Peek**
	The register holds the results from the last single conversion. Reading this field will not clear SINGLEDV in ADCn_STATUS or SINGLE DMA request.			

23.5.13 ADCn_SCANDATAP - Scan Sequence Result Data Peek Register

Offset	Bit Position																													
0x030	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	R																															
Name	DATAP																															

Bit	Name	Reset	Access	Description
31:0	DATAP	0x00000000	R	**Scan Conversion Result Data Peek**
	The register holds the results from the last scan conversion. Reading this field will not clear SCANDV in ADCn_STATUS or single DMA request.			

23.5.14 ADCn_CAL - Calibration Register

Offset	Bit Position																													
0x034	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset		0x3F								0x00								0x3F								0x00						
Access		RW								RW								RW								RW						
Name		SCANGAIN								SCANOFFSET								SINGLEGAIN								SINGLEOFFSET						

Bit	Name	Reset	Access	Description
31	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
30:24	SCANGAIN	0x3F	RW	**Scan Mode Gain Calibration Value**
	This register contains the gain calibration value used with scan conversions. This field is set to the production gain calibration value for the 1V25 internal reference during reset, hence the reset value might differ from device to device. The field is unsigned. Higher values lead to higher ADC results.			
23	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
22:16	SCANOFFSET	0x00	RW	**Scan Mode Offset Calibration Value**
	This register contains the offset calibration value used with scan conversions. This field is set to the production offset calibration value for the 1V25 internal reference during reset, hence the reset value might differ from device to device. The field is encoded as a signed 2's complement number. Higher values lead to lower ADC results.			
15	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
14:8	SINGLEGAIN	0x3F	RW	**Single Mode Gain Calibration Value**
	This register contains the gain calibration value used with single conversions. This field is set to the production gain calibration value for the 1V25 internal reference during reset, hence the reset value might differ from device to device. The field is unsigned. Higher values lead to higher ADC results.			
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:0	SINGLEOFFSET	0x00	RW	**Single Mode Offset Calibration Value**

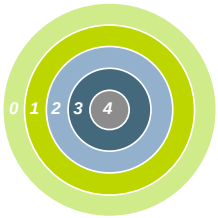
Bit	Name	Reset	Access	Description
	This register contains the offset calibration value used with single conversions. This field is set to the production offset calibration value for the 1V25 internal reference during reset, hence the reset value might differ from device to device. The field is encoded as a signed 2's complement number. Higher values lead to lower ADC results.			

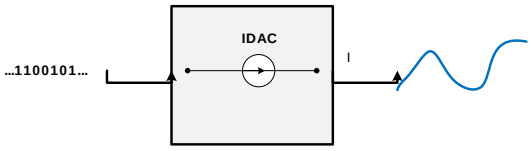
23.5.15 ADCn_BIASPROG - Bias Programming Register

Offset	Bit Position																												
0x03C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																					0x7					1			0x7			
Access																					RW					RW			RW			
Name																					COMPBIAS					HALFBIAS			BIASPROG			

Bit	Name	Reset	Access	Description
31:12	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
11:8	COMPBIAS	0x7	RW	**Comparator Bias Value**
	These bits are used to adjust the bias current to the ADC Comparator.			
7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	HALFBIAS	1	RW	**Half Bias Current**
	Set this bit to halve the bias current.			
5:4	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
3:0	BIASPROG	0x7	RW	**Bias Programming Value**
	These bits are used to adjust the bias current.			

24 IDAC - Current Digital to Analog Converter





Quick Facts

What?

The IDAC can sink, or source a configurable constant current.

Why?

The IDAC can be used to bias external circuits or with the ADC measure capacitance by injecting a controlled current into a component.

How?

In addition to providing a constant current, the IDAC can be switched on and off with a PRS signal all the way down to EM3.

24.1 Introduction

The current digital to analog converter (IDAC) can source or sink a configurable constant current, which can be output on, or sinked from pin or ADC. The current is configurable with several ranges of various step sizes.

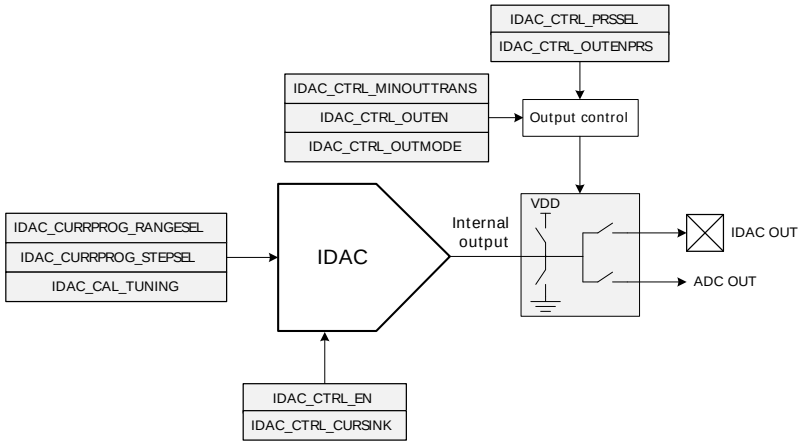
24.2 Features

- Can source and sink current
- Programmable constant output current
  - Selectable current range between 0.05 and 64 µA
  - Each range is linearly programmable in 32 steps
  - Support for current calibration
- Can charge ADC channels
- Support for manual and PRS triggered output enable
- Available in EM0-EM3

24.3 Functional Description

An overview of the IDAC module is shown in Figure 24.1 (p. 538) . The IDAC is designed to source or sink a programmable current which can be controlled by setting the range and the step in RANGESEL and STEPSEL bitfields in IDAC_CURRPROG register. The IDAC output enable to pin and ADC can be controlled by software or PRS. Output enable is controlled by software by setting OUTEN, or by PRS by setting OUTENPRS in IDAC_CTRL. The OUTMODE bitfield in IDAC_CTRL can be configured to choose either pin or ADC. The IDAC is enabled by setting IDACEN in IDAC_CTRL.

Figure 24.1. IDAC Overview



24.3.1 Current Programming

The 4 different current ranges can be selected by configuring the RANGESEL bitfield in IDAC_CURRPROG. Each range is linearly programmable in 32 steps, which is configured by the STEPSEL bitfield in IDAC_CURRPROG. These current ranges with their step sizes are shown in Table 24.1 (p. 538) .

Table 24.1. Range Selection

Range Select	Range Value [µA]	Step Size [nA]	Step Counts
0	0.05 - 1.6	50	32
1	1.6 - 4.7	100	32
2	0.5 - 16	500	32
3	2 - 64	2000	32

24.3.2 Output Control

The IDAC output can be controlled either by software or PRS. After configuring the desired output mode, setting OUTENPRS in IDAC_CTRL enables PRS control over outenable, while setting OUTEN in IDAC_CTRL for enabling via software.

24.3.3 Output Modes

The IDAC can output current either to pin, or to the currently selected ADC channel. Setting OUTMODE to PIN in IDAC_CTRL will output current to the IDAC_OUT pin, while setting OUTMODE to ADC will direct the current to one of the ADC channels. In ADC mode, the pin being charged depends on the channel selected for sampling by the ADC. Thus, if channel 1 is being sampled by ADC, the current from IDAC will charge the same pin.

24.3.4 Minimizing Output Transition

If the internal output of the IDAC is at a different voltage than the output pin, enabling the output can cause an unwanted output transition. To minimize this output transition it is possible to charge or discharge the internal output before enabling the output. Setting MINOUTTRANS in IDAC_CTRL when the IDAC is sourcing lowers the internal node to GND. When sinking, the internal output node is risen to VDD. Setting OUTEN when MINOUTTRANS is set will stop the charge/discharging until OUTEN is cleared, or when MINOUTTRANS is cleared.

24.3.5 Duty Cycle Configuration

The references for the IDAC can be duty-cycled at 4 Hz, meaning that it can source current at very low overhead current consumption at the cost of response time. By default duty-cycling is enabled in EM2 and EM3 and disabled in EM0 and EM1 but this is configurable. Setting DUTYCYCLEEN in IDAC_DUTYCONFIG will force duty cycling on in all energy modes, while setting EM2DUTYCYCLEDIS in the same register will disable duty cycling in EM2 and EM3, if DUTYCYCLEEN is not set. Note that sinking current can not be done with duty-cycled references so measures needs to be taken to always disable duty-cycling while sinking current.

24.3.6 Calibration

The IDAC can be calibrated to accurately compensate for process, supply voltage and temperature variations. During the production test, the middle step of each range is calibrated at room temperature. The TUNING bitfield in the IDAC_CAL register can be used to do further calibration of each step with an external resistor connected to IDAC_OUT. The calibrated tuning value for each band can be read from the Device Information (DI) page.

24.3.7 PRS Input

The IDAC can be configured to control output enable directly from the PRS channel by setting OUTENPRS in IDAC_CTRL. Also, the desired outmode (pin or ADC) must be configured in IDAC_CTRL. The PRS channel is selected using PRSSEL in the IDAC_CTRL register.

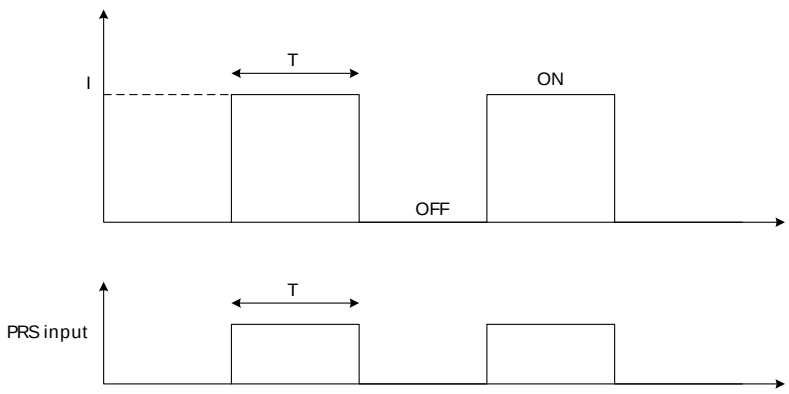
24.3.8 PRS Triggered Charge Injection

The amount of charge sourced or sinked by the IDAC can be controlled through PRS (e.g. with a Timer as producer) via the output switch. Figure 24.2 (p. 539) shows a case where the IDAC is configured to periodically supply charge using the PRS. The amount of charge injected is proportional to the the period the IDAC is on. The total charge injected is the current times the time the output switch is enabled.

The PRS system is enabled by setting OUTENPRS in IDAC_CTRL, and the PRS channel is selected by PRSSEL in IDAC_CTRL. Also OUTMODE must be set to ADC in IDAC_CTRL. To generate the periodic control signal, the TIMER module can be used, by configuring for example a CC channel to compare match with PRSLEVEL selected.

It is possible to observe the charge injection on the corresponding pin on the ADC inputmux. However, during normal ADC operations, the inputmux is shutdown between conversions, making it not possible to observe the charge injection correctly. Setting CHCONIDLE to KEEPCON in ADC_CTRL will enable the inputmux between conversions as well. The ADC_CTRL register description can be found Section 23.4 (p. 524)

Figure 24.2. IDAC Charge Injection Example



24.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	IDAC_CTRL	RW	Control Register
0x004	IDAC_CURPROG	RW	Current Programming Register
0x008	IDAC_CAL	RW	Calibration Register
0x00C	IDAC_DUTYCONFIG	RW	Duty Cycle Configauration Register

24.5 Register Description

24.5.1 IDAC_CTRL - Control Register

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset											0x0			0														0	0	0	0	0
Access											RW			RW														RW	RW	RW	RW	RW
Name											PRSSEL			OUTENPRS														OUTMODE	OUTEN	MINOUTTRANS	CURSINK	EN

Bit	Name	Reset	Access	Description
31:23	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
22:20	PRSSEL	0x0	RW	**IDAC Output PRS channnel Select**

Selects which PRS channel to use, when OUTENPRS is set.

Value	Mode	Description
0	PRSCH0	PRS Channel 0 selected.
1	PRSCH1	PRS Channel 1 selected.
2	PRSCH2	PRS Channel 2 selected.
3	PRSCH3	PRS Channel 3 selected.
4	PRSCH4	PRS Channel 4 selected.
5	PRSCH5	PRS Channel 5 selected.

19	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
18	OUTENPRS	0	RW	**PRS Controlled Output Enable**

Enable PRS Control of IDAC output enable.

Value	Description
0	IDAC_OUTMODE controlled by IDAC_OUTEN.
1	IDAC_OUTMODE controlled by PRS input selected by PRSSEL.

17:5	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
4	OUTMODE	0	RW	**Output Modes**

Select output mode.

Value	Mode	Description
0	PIN	IDAC output to pin enabled.
1	ADC	IDAC output to pin disabled. IDAC output to ADC enabled.

3	OUTEN	0	RW	**Output Enable**

Set to enable IDAC output if OUTENPRS is not set.

Bit	Name	Reset	Access	Description
2	MINOUTTRANS	0	RW	**Minimum Output Transition Enable**
	Set to enable Minimum output transition mode for the IDAC.			
1	CURSINK	0	RW	**Current Sink Enable**
	Set to enable the IDAC as current sink. By default, the IDAC sources current. Note that while sinking current, duty cycling needs to be disabled.			
0	EN	0	RW	**Current DAC Enable**
	Set to enable the IDAC.			

24.5.2 IDAC_CURPROG - Current Programming Register

Offset	Bit Position																														
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																					0x00										0x0	
Access																					RW										RW	
Name																					STEPSEL										RANGESEL	

Bit	Name	Reset	Access	Description
31:13	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
12:8	STEPSEL	0x00	RW	**Current Step Size Select**
	Select the step within each range. Please see Table 24.1 (p. 538) for step details.			
7:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1:0	RANGESEL	0x0	RW	**Current Range Select**
	Selects current range of the output.			

Value	Mode	Description
0	RANGE0	Current range set to 0 - 1.6 uA.
1	RANGE1	Current range set to 1.6 - 4.7 uA.
2	RANGE2	Current range set to 0.5 - 16 uA.
3	RANGE3	Current range set to 2 - 64 uA.

24.5.3 IDAC_CAL - Calibration Register

Offset	Bit Position																														
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																										0x00						
Access																										RW						
Name																										TUNING						

Bit	Name	Reset	Access	Description
31:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:0	TUNING	0x00	RW	**Tune the current to given accuracy**

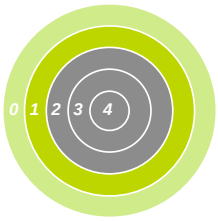
Bit	Name	Reset	Access	Description
	In production test the middle step (16) of each range is calibrated and can be read from the Device Information (DI) page.			

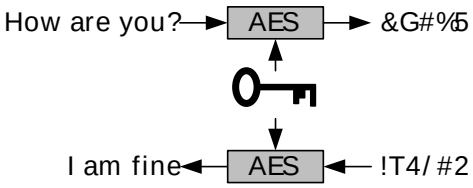
24.5.4 IDAC_DUTYCONFIG - Duty Cycle Configauration Register

Offset	Bit Position																													
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															RW	RW
Name																															EM2DUTYCYCLEDIS	DUTYCYCLEEN

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	EM2DUTYCYCLEDIS	0	RW	**EM2/EM3 Duty Cycle Disable.**
	Set to disable duty cycling in EM2 and EM3.			
0	DUTYCYCLEEN	0	RW	**Duty Cycle Enable.**
	Set to always enable duty cycling. Will override EM2DUTYCYCLEDIS.			

25 AES - Advanced Encryption Standard Accelerator





Quick Facts

What?

A fast and energy efficient hardware accelerator for AES-128 encryption and decryption.

Why?

Efficient encryption/decryption with little or no CPU intervention helps to meet the speed and energy demands of the application.

How?

High AES throughput allows the EZR32HG to spend more time in lower energy modes. In addition, specialized data access functions allow autonomous DMA/AES operation in both EM0 and EM1.

25.1 Introduction

The Advanced Encryption Standard (FIPS-197) is a symmetric block cipher operating on 128-bit blocks of data and 128-bit keys.

The AES accelerator performs AES encryption and decryption with 128-bit keys. Encrypting or decrypting one 128-bit data block takes 54 HFCORECLK cycles with 128-bit keys. The AES module is an AHB slave which enables efficient access to the data and key registers. All write accesses to the AES module must be 32-bit operations, i.e. 8- or 16-bit operations are not supported.

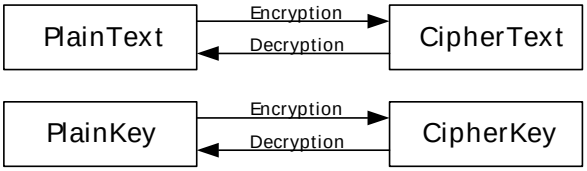
25.2 Features

- AES hardware encryption/decryption
  - 128-bit key (54 HFCORECLK cycles)
- Efficient CPU/DMA support
- Interrupt on finished encryption/decryption
- DMA request on finished encryption/decryption
- Optional XOR on Data write
- Configurable byte ordering

25.3 Functional Description

Some data and a key must be loaded into the KEY and DATA registers before an encryption or decryption can take place. The input data before encryption is called the PlainText and output from the encryption is called CipherText. For encryption, the key is called PlainKey. After one encryption, the resulting key in the KEY registers is the CipherKey. This key must be loaded into the KEY registers before every decryption. After one decryption, the resulting key will be the PlainKey. The resulting PlainKey/CipherKey is only dependent on the value in the KEY registers before encryption/decryption. The resulting keys and data are shown in Figure 25.1 (p. 545) .

Figure 25.1. AES Key and Data Definitions



25.3.1 Encryption/Decryption

The AES module can be set to encrypt or decrypt by clearing/setting the DECRYPT bit in AES_CTRL. The AES_CTRL register should not be altered while AES is running, as this may lead to unpredictable behaviour.

An AES encryption/decryption can be started in the following ways:

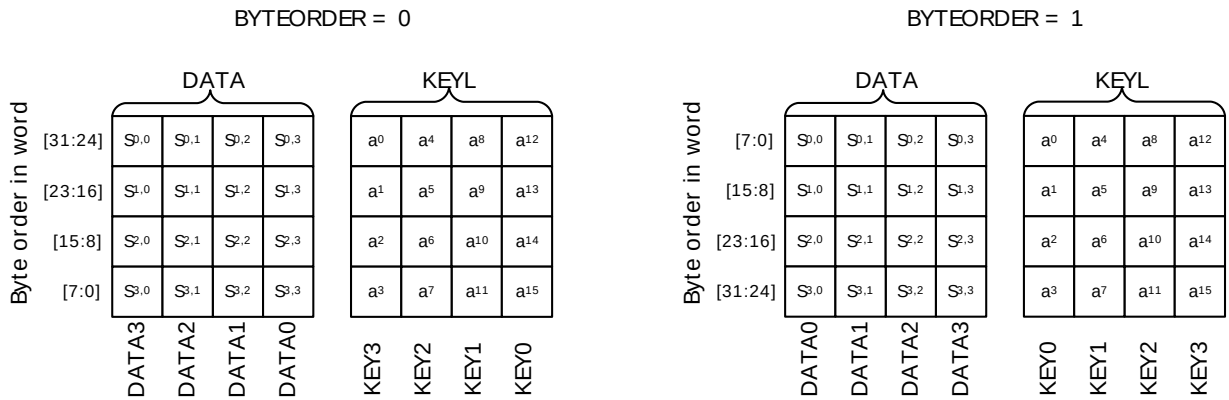
- Writing a 1 to the START bit in AES_CMD
- Writing 4 times 32 bits to AES_DATA when the DATASTART control bit is set
- Writing 4 times 32 bits to AES_XORDATA when the XORSTART control bit is set

An AES encryption/decryption can be stopped by writing a 1 to the STOP bit in AES_CMD. The RUNNING bit in AES_STATUS indicates that an AES encryption/decryption is ongoing.

25.3.2 Data and Key Access

The AES module contains a 128-bit DATA (State) register and a 128-bit KEY register defined as DATA3-DATA0 and KEY3-KEY0 (KEYL). The AES module has configurable byte ordering which is configured in BYTEORDER in AES_CTRL. Figure 25.2 (p. 545) illustrates how data written to the AES registers is mapped to the key and state defined in the Advanced Encryption Standard (FIPS-197). AES encryption/decryption takes two extra cycles when BYTEORDER is set. BYTEORDER has to be set prior to loading the data and key registers.

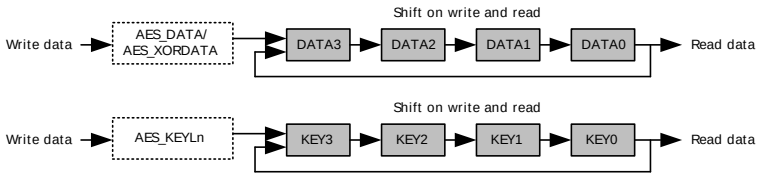
Figure 25.2. AES Data and Key Orientation as Defined in the Advanced Encryption Standard



The registers DATA3-DATA0, are not memory mapped directly, but can be written/read by accessing AES_DATA or AES_XORDATA. The same applies for the key registers, KEY3-KEY0 which are accessed through AES_KEYLn (n=A, B, C or D). Writing DATA3-DATA0 is then done through 4

consecutive writes to AES_DATA (or AES_XORDATA), starting with the word which is to be written to DATA0. For each write, the words will be word wise barrel shifted towards the least significant word. Accessing the KEY registers is done in the same fashion through KEYLn. See Figure 25.3 (p. 546) . Note that KEYLA, KEYLB, KEYLC and KEYLD are really the same register, just mapped to four different addresses. You can then choose freely which of these addresses you want to use to update the KEY3-KEY0 registers. Mapping the same registers to multiple addresses like this, allows the DMA controller to write a full 128-bit key in one sweep, when incrementing the address between each word write.

Figure 25.3. AES Data and Key Register Operation



Note

When encrypting multiple blocks of data in a row, the PlainKey must be written to the key register between each encryption, since the contents of the key registers will be turned into the CipherKey during the encryption. The opposite applies when decrypting, where you have to re-supply the CipherKey between each block.

25.3.2.1 Data Write XOR

The AES module contains an array of XOR gates connected to the DATA registers, which can be used during a data write to XOR the existing contents of the registers with the new data written. To use the XOR function, the data must be written to AES_XORDATA location.

Reading data from AES_XORDATA is equivalent to reading data from AES_DATA.

25.3.2.2 Start on Data Write

The AES module can be configured to start an encryption/decryption when the new data has been written to AES_DATA and/or AES_XORDATA. A 2-bit counter is incremented each time the AES_DATA or AES_XORDATA registers are written. This counter indicates which data word is written. If DATASTART/XORSTART in AES_CTRL is set, an encryption will start each time the counter overflows (DATA3 is written). Writing to the AES_CTRL register will reset the counter to 0.

25.3.3 Interrupt Request

The DONE interrupt flag is set when an encryption/ decryption has finished.

25.3.4 DMA Request

The AES module has 4 DMA requests which are all set on a finished encryption/decryption and cleared on the following conditions:

- DATAWR: Cleared on a AES_DATA write or AES_CTRL write
- XORDATAWR: Cleared on a AES_XORDATA write or AES_CTRL write
- DATARD: Cleared on a AES_DATA read or AES_CTRL write
- KEYWR: Cleared on a AES_KEYLn write or AES_CTRL write

25.3.5 Block Chaining Example

Example 25.1 (p. 547) below illustrates how the AES module could be configured to perform Cipher Block Chaining with 128-bit keys.

Example 25.1. AES Cipher Block Chaining

1. Configure module to encryption and XORSTART in AES_CTRL.
2. Write 128-bit initialization vector to AES_DATA, starting with least significant word.
3. Write PlainKey to AES_KEYLn, starting with least significant word.
4. Write PlainText to AES_XORDATA, starting with least significant word. Encryption will be started when the DATA3 is written.
5. When encryption is finished, read CipherText from AES_DATA, starting with least significant word.
6. Loop to step 3, if new PlainText is available.

25.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	AES_CTRL	RW	Control Register
0x004	AES_CMD	W1	Command Register
0x008	AES_STATUS	R	Status Register
0x00C	AES_IEN	RW	Interrupt Enable Register
0x010	AES_IF	R	Interrupt Flag Register
0x014	AES_IFS	W1	Interrupt Flag Set Register
0x018	AES_IFC	W1	Interrupt Flag Clear Register
0x01C	AES_DATA	RW	DATA Register
0x020	AES_XORDATA	RW	XORDATA Register
0x030	AES_KEYLA	RW	KEY Low Register
0x034	AES_KEYLB	RW	KEY Low Register
0x038	AES_KEYLC	RW	KEY Low Register
0x03C	AES_KEYLD	RW	KEY Low Register

25.5 Register Description

25.5.1 AES_CTRL - Control Register

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																										0	0	0				0
Access																										RW	RW	RW				RW
Name																										BYTEORDER	XORSTART	DATASTART				DECRYPT

Bit	Name	Reset	Access	Description
31:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6	BYTEORDER	0	RW	**Configure byte order in data and key registers**
	When set, the byte orders in the data and key registers are swapped before and after encryption/decryption.			
5	XORSTART	0	RW	**AES_XORDATA Write Start**
	Set this bit to start encryption/decryption when DATA3 is written through AES_XORDATA.			
4	DATASTART	0	RW	**AES_DATA Write Start**
	Set this bit to start encryption/decryption when DATA3 is written through AES_DATA.			
3:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	DECRYPT	0	RW	**Decryption/Encryption Mode**
	Select encryption or decryption.			

Value	Description
0	AES Encryption
1	AES Decryption

25.5.2 AES_CMD - Command Register

Offset	Bit Position																														
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0	0
Access																															W1	W1
Name																															STOP	START

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	STOP	0	W1	**Encryption/Decryption Stop**
	Set to stop encryption/decryption.			
0	START	0	W1	**Encryption/Decryption Start**
	Set to start encryption/decryption.			

25.5.3 AES_STATUS - Status Register

Offset	Bit Position																														
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																R
Name																																RUNNING

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	RUNNING	0	R	**AES Running**
	This bit indicates that the AES module is running an encryption/decryption.			

25.5.4 AES_IEN - Interrupt Enable Register

Offset	Bit Position																														
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																RW
Name																																DONE

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	DONE	0	RW	**Encryption/Decryption Done Interrupt Enable**
	Enable/disable interrupt on encryption/decryption done.			

25.5.5 AES_IF - Interrupt Flag Register

Offset	Bit Position																															
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																R
Name																																DONE

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	DONE	0	R	**Encryption/Decryption Done Interrupt Flag**
	Set when an encryption/decryption has finished.			

25.5.6 AES_IFS - Interrupt Flag Set Register

Offset	Bit Position																															
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																W1
Name																																DONE

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	DONE	0	W1	**Encryption/Decryption Done Interrupt Flag Set**
	Write to 1 to set encryption/decryption done interrupt flag			

25.5.7 AES_IFC - Interrupt Flag Clear Register

Offset	Bit Position																															
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																W1
Name																																DONE

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	DONE	0	W1	**Encryption/Decryption Done Interrupt Flag Clear**
	Write to 1 to clear encryption/decryption done interrupt flag			

25.5.8 AES_DATA - DATA Register

Offset	Bit Position																														
0x01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	RW																															
Name	DATA																															

Bit	Name	Reset	Access	Description
31:0	DATA	0x00000000	RW	**Data Access**
	Access data through this register.			

25.5.9 AES_XORDATA - XORDATA Register

Offset	Bit Position																														
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	RW																															
Name	XORDATA																															

Bit	Name	Reset	Access	Description
31:0	XORDATA	0x00000000	RW	**XOR Data Access**
	Access data with XOR function through this register.			

25.5.10 AES_KEYLA - KEY Low Register

Offset	Bit Position																														
0x030	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	RW																															
Name	KEYLA																															

Bit	Name	Reset	Access	Description
31:0	KEYLA	0x00000000	RW	**Key Low Access A**
	Access the low key words through this register.			

25.5.11 AES_KEYLB - KEY Low Register

Offset	Bit Position																													
0x034	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	RW																															
Name	KEYLB																															

Bit	Name	Reset	Access	Description
31:0	KEYLB	0x00000000	RW	**Key Low Access B**
	Access the low key words through this register.			

25.5.12 AES_KEYLC - KEY Low Register

Offset	Bit Position																													
0x038	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	RW																															
Name	KEYLC																															

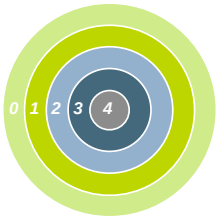
Bit	Name	Reset	Access	Description
31:0	KEYLC	0x00000000	RW	**Key Low Access C**
	Access the low key words through this register.			

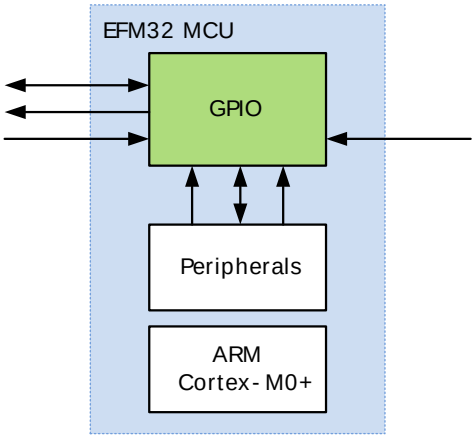
25.5.13 AES_KEYLD - KEY Low Register

Offset	Bit Position																													
0x03C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00000000																															
Access	RW																															
Name	KEYLD																															

Bit	Name	Reset	Access	Description
31:0	KEYLD	0x00000000	RW	**Key Low Access D**
	Access the low key words through this register.			

26 GPIO - General Purpose Input/Output





Quick Facts

What?

The GPIO (General Purpose Input/Output) is used for pin configuration and direct pin manipulation and sensing as well as routing for peripheral pin connections.

Why?

Easy to use and highly configurable input/output pins are important to fit many communication protocols as well as minimizing software control overhead. Flexible routing of peripheral functions helps to ease PCB layout.

How?

Each pin on the device can be individually configured as either an input or an output with several different drive modes. Also, individual bit manipulation registers minimizes control overhead. Peripheral connections to pins can be routed to several different locations, thus solving congestion issues that may arise with multiple functions on the same pin. Fully asynchronous interrupts can also be generated from any pin.

26.1 Introduction

In the EZR32HG devices the General Purpose Input/Output (GPIO) pins are organized into ports with up to 16 pins each. These pins can individually be configured as either an output or input. More advanced configurations like open-drain, filtering and drive strength can also be configured individually for the pins. The GPIO pins can also be overridden by peripheral pin connections, like Timer PWM outputs or USART communication, which can be routed to several locations on the device. The GPIO supports up to 16 asynchronous external pin interrupts, which enables interrupts from any pin on the device. Also, the input value of a pin can be routed through the Peripheral Reflex System to other peripherals.

26.2 Features

- Single-cycle I/O interface providing high speed access to GPIO
- Individual configuration for each pin
  - Tristate (reset state)
  - Push-pull
  - Open-drain
  - Pull-up resistor
  - Pull-down resistor
  - Four drive strength modes
    - HIGH
    - STANDARD
    - LOW

    - LOWEST
- EM4 IO pin retention. This includes
    - Output enable
    - Output value
    - Pull enable
    - Pull direction
- EM4 wake-up on selected GPIO pins
- Glitch suppression input filter.
- Analog connection to e.g. ADC.
- Alternate functions (e.g. peripheral outputs and inputs)
    - Routed to several locations on the device
    - Pin connections can be enabled individually
    - Output data can be overridden by peripheral
    - Output enable can be overridden by peripheral
- Toggle, set and clear registers for output data
- Dedicated data input register (read-only)
- Interrupts
    - 2 interrupt lines from up to 16 pending sources
        - All GPIO pins are selectable
    - Separate enable, status, set and clear registers
    - Asynchronous sensing
    - Rising, falling or both edges
    - Wake up from EM0-EM3
- Peripheral Reflex System producer
    - All GPIO pins are selectable
- Configuration lock functionality to avoid accidental changes

26.3 Functional Description

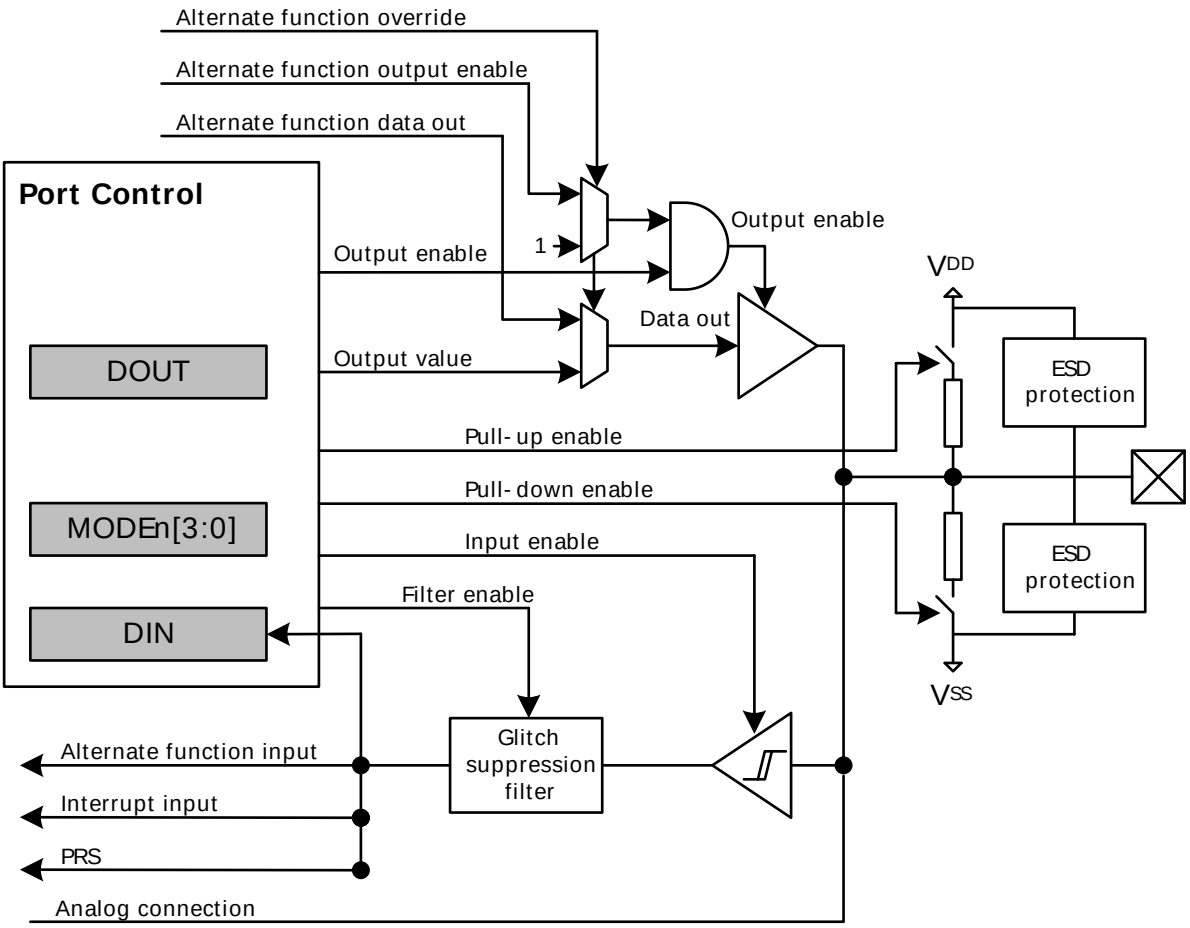
An overview of the GPIO module is shown in Figure 26.1 (p. 556) .The GPIO pins are grouped into 16-pin ports. Each individual GPIO pin is called Pxn where x indicates the port (A, B, C ...) and n indicates the pin number (0,1,....,15). Fewer than 16 bits may be available on some ports, depending on the total number of I/O pins on the package. After a reset both input and output is disabled for all pins on the device, except for debug pins. To use a pin, the port GPIO_Px_MODEL/GPIO_Px_MODEH registers must be configured for the pin to make it an input or output. These registers can also do more advanced configuration, which is covered in Section 26.3.1 (p. 556) . When the port is either configured as an input or an output, the Data In Register (GPIO_Px_DIN) can be used to read the level of each pin in the port (bit n in the register is connected to pin n on the port). When configured as an output, the value of the Data Out Register (GPIO_Px_DOUT) will be driven to the pin.

The DOUT value can be changed in 4 different ways

- Writing to the GPIO_Px_DOUT register.
- Writing a 1 to a bit in the GPIO_Px_DOUTSET register sets the corresponding DOUT bit
- Writing a 1 to a bit in the GPIO_Px_DOUTCLR register clears the corresponding DOUT bit
- Writing a 1 to a bit in the GPIO_Px_DOUTTGL register toggles the corresponding DOUT bit

Reading the GPIO_Px_DOUT register will return its contents. Reading the GPIO_Px_DOUTSET, GPIO_Px_CLR or GPIO_Px_TGL will return 0.

Figure 26.1. Pin Configuration



26.3.1 Pin Configuration

In addition to setting the pins as either outputs or inputs, the GPIO_Px_MODEL and GPIO_Px_MODEH registers can be used for more advanced configurations. GPIO_Px_MODEL contains 8 bit fields named MODEn (n=0,1,..7) which control pins 0-7, while GPIO_Px_MODEH contains 8 bit fields named MODEn (n=8,9,..15) which control pins 8-15. In some modes GPIO_Px_DOUT is also used for extra configurations like pull-up/down and glitch suppression filter enable. Table 26.1 (p. 556) shows the available configurations.

Table 26.1. Pin Configuration

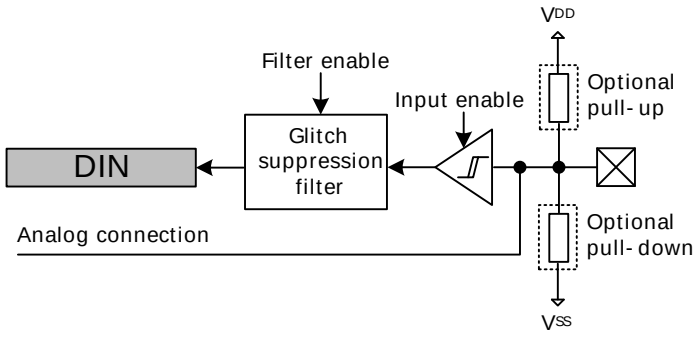
MODEn	Input	Output	DOUT	Pull-down	Pull-up	Alt. strength	Input Filter	Description
0b0000	Disabled	Disabled	0					Input disabled
			1		On			Input disabled with pull-up
0b0001	Enabled		0					Input enabled
			1				On	Input enabled with filter
0b0010			0	On				Input enabled with pull-down
			1		On			Input enabled with pull-up

MODEn	Input	Output	DOUT	Pull-down	Pull-up	Alt. strength	Input Filter	Description
0b0011			0	On			On	Input enabled with pull-down and filter
			1		On		On	Input enabled with pull-up and filter
0b0100		Push-pull	x					Push-pull
0b0101			x			On		Push-pull with alt. drive strength
0b0110		Open Source (Wired-OR)	x					Open-source
0b0111			x	On				Open-source with pull-down
0b1000		Open Drain (Wired-AND)	x					Open-drain
0b1001			x				On	Open-drain with filter
0b1010			x		On			Open-drain with pull-up
0b1011			x		On		On	Open-drain with pull-up and filter
0b1100			x			On		Open-drain with alt. drive strength
0b1101			x			On	On	Open-drain with alt. drive strength and filter
0b1110			x		On	On		Open-drain with alt. drive strength and pull-up
0b1111			x		On	On	On	Open-drain with alt. drive strength, pull-up and filter

MODEn determines which mode the pin is in at a given time. Setting MODEn to 0b0000 disables the pin, reducing power consumption to a minimum. When the output driver is disabled, the pin can be used as a connection for an analog module (e.g. ADC). Input is enabled by setting MODEn to any value other than 0b0000. The pull-up, pull-down and filter function can optionally be applied to the input, see Figure 26.2 (p. 557) .

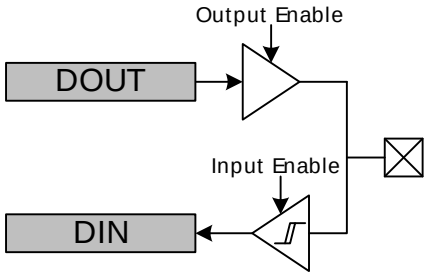
The internal pull-up resistance, R_{PU}, and pull-down resistance, R_{PD}, are defined in the device datasheet. When the filter is enabled it suppresses glitches with pulse widths as defined by the parameter $t_{IOGLITCH}$ in the device datasheet.

Figure 26.2. Tristated Output with Optional Pull-up or Pull-down



When MODEn=0b0100 or MODEn=0b0101, the pin operates in push-pull mode. In this mode, the pin is driven either high or low, dependent on the value of GPIO_Px_DOUT. The push-pull configuration is shown in Figure 26.3 (p. 558) .

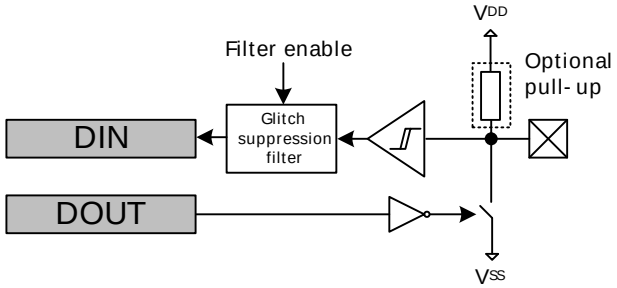
Figure 26.3. Push-Pull Configuration



When MODEn is 0110 or 0111, the pin operates in open-source mode, the latter with a pull-down resistor. When driving a high value in open-source mode, the pull-down is disconnected to save power.

For the remaining MODEn values, i.e. MODEn >= 1000, the pin operates in open-drain mode as shown in Figure 26.4 (p. 558) . In open-drain mode, the pin can have an input filter, a pull-up, different driver strengths or any combination of these. When driving a low value in open-drain mode, the pull-up is disconnected to save power.

Figure 26.4. Open-drain



When MODEn=0b0101 or 0b11xx, the output driver uses the drive strength specified in DRIVEMODE in GPIO_Px_CTRL. In all other output modes, the drive strength is set to STANDARD.

26.3.1.1 Configuration Lock

GPIO_Px_MODEL, GPIO_Px_MODEH, GPIO_Px_CTRL, GPIO_Px_PINLOCKN, GPIO_EXTIPSELL, GPIO_EXTIPSELH, GPIO_INSENSE and GPIO_ROUTE can be locked by writing any other value than 0xA534 to GPIO_LOCK. Writing the value 0xA534 to the GPIOx_LOCK register unlocks the configuration registers.

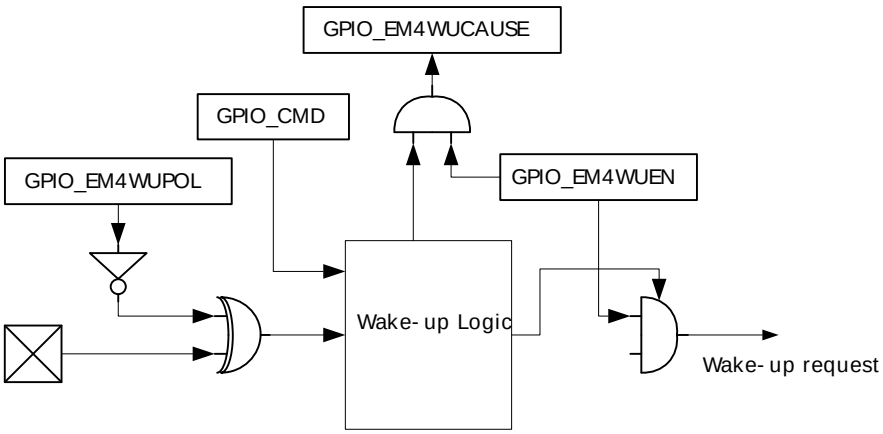
In addition to configuration lock, GPIO_Px_MODEL, GPIO_Px_MODEH, GPIO_Px_DOUT, GPIO_Px_DOUTSET, GPIO_Px_DOUTCLR, and GPIO_Px_DOUTTGL can be locked individually for each pin by clearing the corresponding bit in GPIO_Px_PINLOCKN. Bits in the GPIO_Px_PINLOCKN register can only be cleared, they are set high again after reset.

26.3.2 EM4 Wake-up

It is possible to wake-up from EM4 through reset triggered from any of up to 6 selectable GPIO pins. For the wake-up logic to work correctly, EM4 retention needs to be enabled before entering EM4, as described in Section 26.3.3 (p. 559) The wake-up request can be triggered through the pins by enabling the corresponding bit in the GPIO_EM4WUEN register. When EM4 wake-up is enabled for the

pin, the input filter is enabled during EM4. This is done to avoid false wake-up caused by glitches. In addition, the polarity of the EM4 wake-up request can be selected using the GPIO_EM4WUPOL register.

Figure 26.5. EM4 Wake-up Logic



The pins used for EM4 wake-up must be configured as inputs using the GPIO_Px_MODEL/ GPIO_Px_MODEH register. Before going down to EM4, it is important to clear the wake-up logic by setting the EM4WUCLR bitfield in the GPIO_CMD register, which clears the complete wake-up logic, including the GPIO_EM4WUCAUSE register. When the chip comes out of reset, it is possible to determine what caused the reset by reading the RMU_RSTCAUSE register. If an EM4 wake-up reset occurred, the EM4RST (indicating the chip was in EM4) and the EM4WU (indicating the EM4 wake-up reset) bits should be set. It is possible to determine which pin caused the reset by reading the GPIO_EM4WUCAUSE register. The mapping between pins and the bits in the GPIO_EM4WUEN, GPIO_EM4WUPOL, and GPIO_EM4WUCAUSE registers are described in Table 26.2 (p. 559)

Table 26.2. EM4 WU Register bits to pin mapping

Wake-up Registers Bits	Pin
bit 0	A0
bit 1	A6
bit 2	C9
bit 3	F1
bit 4	F2
bit 5	E13

26.3.3 EM4 Retention

It is possible to enable retention of output enable, output value, pull enable and pull direction when in EM4. EM4 retention also makes it possible to wake up from EM4 on pin reset as described in Section 26.3.2 (p. 558) EM4 retention can be enabled by setting the EM4RET field in GPIO_CTRL register before going down in EM4.

26.3.4 Alternate Functions

Alternate functions are connections to pins from Timers, USARTs etc. These modules contain route registers, where the pin connections are enabled. In addition, these registers contain a location bit field, which configures which pins the outputs of that module will be connected to if they are enabled. If an alternate signal output is enabled for a pin and output is enabled for the pin, the alternate

function's output data and output enable signals override the data output and output enable signals from the GPIO. However, the pin configuration stays as set in GPIO_Px_MODEL, GPIO_Px_MODEH and GPIO_Px_DOUT registers. I.e. the pin configuration must be set to output enable in GPIO for a peripheral to be able to use the pin as an output.

It is possible, but not recommended to select two or more peripherals as output on the same pin. These signals will then be OR'ed together. However, TIMER CCx and CDTIx outputs, which are routed as alternate functions, have priority, and will never be OR'ed with other alternate functions. The reader is referred to the pin map section of the device datasheet for more information on the possible locations of each alternate function and any priority settings.

26.3.4.1 Serial Wire Debug Port Connection

The SW Debug Port is routed as an alternate function and the SWDIO and SWCLK pin connections are enabled by default with internal pull-up and pull-down resistors, respectively. It is possible to disable these pin connections (and disable the pull resistors) by setting the SWDIOPEN and SWCLKPEN bits in GPIO_ROUTE to 0.

WARNING: When the debug pins are disabled, the device can no longer be accessed by a debugger. A reset will set the debug pins back to their default state as enabled. If you do disable the debug pins, make sure you have at least a 3 second timeout at the start of your program code before you disable the debug pins. This way the debugger will have time to halt the device after a reset before the pins are disabled.

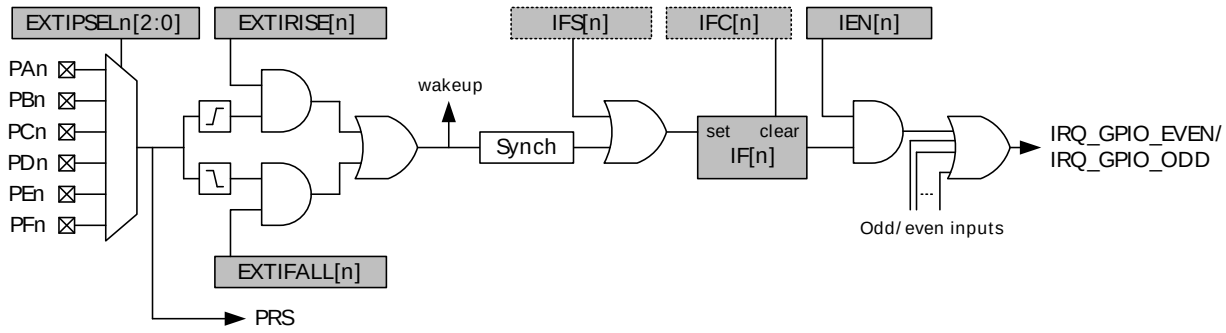
26.3.4.2 Analog Connections

When using the GPIO pin for analog functionality, it is recommended to disable the digital output and set the MODEn in GPIO_Px_MODEL/GPIO_Px_MODEH equal to 0b0000 to disable the input sense and pull resistors.

26.3.5 Interrupt Generation

The GPIO can generate an interrupt from the input of any GPIO pin on a device. The interrupts have asynchronous sense capability, enabling wake-up from energy modes as low as EM3, see Figure 26.6 (p. 560) .

Figure 26.6. Pin n Interrupt Generation



All pins with the same pin number (n) are grouped together to trigger one interrupt flag (EXT[n] in GPIO_IF). The EXTIPSELn[2:0] bits in GPIO_EXTIPSELL or GPIO_EXTIPSELH select which port will trigger the interrupt flag. The GPIO_EXTIRISE[n] and GPIO_EXTIFALL[n] registers enables sensing of rising and falling edges. By setting the EXT[n] bit in GPIO_IEN, a high interrupt flag n, will trigger one of two interrupt lines. The even interrupt line is triggered by any enabled even numbered interrupt flag, while the odd is triggered by odd flags. The interrupt flags can be set and cleared by software by writing the GPIO_IFS and GPIO_IFC registers, see Example 26.1 (p. 561) . Since the external interrupts are asynchronous, they are sensitive to noise. To increase noise tolerance, the MODEL and MODEH

fields in the GPIO_Px_MODEL and GPIO_Px_MODEH registers, respectively, should be set to include filtering for pins that have external interrupts enabled.

Example 26.1. GPIO Interrupt Example

Setting EXTIPSEL3 in GPIO_EXTIPSELL to 2 (Port C) and setting the GPIO_EXTIRISE[3] bit, the interrupt flag EXT[3] in GPIO_IF will be triggered by a rising edge on pin 3 on PORT C. If EXT[3] in GPIO_IEN is set as well, a interrupt request will be sent on IRQ_GPIO_ODD.

26.3.6 Output to PRS

All pins with the same pin number (n) are grouped together to form one PRS producer output, giving a total of 16 outputs to the PRS. The port on which the output n should be taken is selected by the EXTIPSELn[3:0] bits in the GPIO_EXTIPSELL or the GPIO_EXTIPSELH registers.

26.3.7 Synchronization

The EZR32HG devices are equipped with a single-cycle I/O interface providing high speed access to the GPIO. To avoid metastability in the synchronous logic connected to the pins, all inputs are synchronized with double flip-flops. The flip-flops for the input data run on the HFCORECLK. The first flip-flop is active whenever the pin is enabled and the second flip-flop is triggered on the negative edge of the HFCORECLK during a read operation. Consequently, when a pin changes state, the change is propagated to GPIO_Px_DIN in a single HFCORECLK cycle. To save power when a certain GPIO pin is not in use, the synchronizing flip-flop for this pin can be turned off by clearing the respective mode field in the GPIO_Px_MODEL/GPIO_Px_MODEH registers.

Synchronization (also running on the HFCORECLK) is also added for interrupt input. The input to the PRS generation is also synchronized, but these flip-flops run on the HFPERCLK. To save power when the external interrupts or PRS generation is not used, the synchronization flip-flops for these can be turned off by clearing the INTSENSE or PRSSENSE, respectively, in GPIO_INSENSE register.

Note

To use the GPIO, the GPIO clock must first be enabled in CMU_HFPERCLKEN0. Setting this bit enables the HFCORECLK and the HFPERCLK for the GPIO. HFCORECLK is used for updating registers, while HFPERCLK is only used to synchronize PRS and interrupts. The PRS and interrupt synchronization can also be disabled through GPIO_INSENSE, if these are not used.

Note

GPIO is not accessible from the DMA.

26.4 Register Map

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	GPIO_PA_CTRL	RW	Port Control Register
0x004	GPIO_PA_MODEL	RW	Port Pin Mode Low Register
0x008	GPIO_PA_MODEH	RW	Port Pin Mode High Register
0x00C	GPIO_PA_DOUT	RW	Port Data Out Register
0x010	GPIO_PA_DOUTSET	W1	Port Data Out Set Register
0x014	GPIO_PA_DOUTCLR	W1	Port Data Out Clear Register
0x018	GPIO_PA_DOUTTGL	W1	Port Data Out Toggle Register
0x01C	GPIO_PA_DIN	R	Port Data In Register
0x020	GPIO_PA_PINLOCKN	RW	Port Unlocked Pins Register
0x024	GPIO_PB_CTRL	RW	Port Control Register
0x028	GPIO_PB_MODEL	RW	Port Pin Mode Low Register
0x02C	GPIO_PB_MODEH	RW	Port Pin Mode High Register
0x030	GPIO_PB_DOUT	RW	Port Data Out Register
0x034	GPIO_PB_DOUTSET	W1	Port Data Out Set Register
0x038	GPIO_PB_DOUTCLR	W1	Port Data Out Clear Register
0x03C	GPIO_PB_DOUTTGL	W1	Port Data Out Toggle Register
0x040	GPIO_PB_DIN	R	Port Data In Register
0x044	GPIO_PB_PINLOCKN	RW	Port Unlocked Pins Register
0x048	GPIO_PC_CTRL	RW	Port Control Register
0x04C	GPIO_PC_MODEL	RW	Port Pin Mode Low Register
0x050	GPIO_PC_MODEH	RW	Port Pin Mode High Register
0x054	GPIO_PC_DOUT	RW	Port Data Out Register
0x058	GPIO_PC_DOUTSET	W1	Port Data Out Set Register
0x05C	GPIO_PC_DOUTCLR	W1	Port Data Out Clear Register
0x060	GPIO_PC_DOUTTGL	W1	Port Data Out Toggle Register
0x064	GPIO_PC_DIN	R	Port Data In Register
0x068	GPIO_PC_PINLOCKN	RW	Port Unlocked Pins Register
0x06C	GPIO_PD_CTRL	RW	Port Control Register
0x070	GPIO_PD_MODEL	RW	Port Pin Mode Low Register
0x074	GPIO_PD_MODEH	RW	Port Pin Mode High Register
0x078	GPIO_PD_DOUT	RW	Port Data Out Register
0x07C	GPIO_PD_DOUTSET	W1	Port Data Out Set Register
0x080	GPIO_PD_DOUTCLR	W1	Port Data Out Clear Register
0x084	GPIO_PD_DOUTTGL	W1	Port Data Out Toggle Register
0x088	GPIO_PD_DIN	R	Port Data In Register
0x08C	GPIO_PD_PINLOCKN	RW	Port Unlocked Pins Register
0x090	GPIO_PE_CTRL	RW	Port Control Register
0x094	GPIO_PE_MODEL	RW	Port Pin Mode Low Register
0x098	GPIO_PE_MODEH	RW	Port Pin Mode High Register
0x09C	GPIO_PE_DOUT	RW	Port Data Out Register

Offset	Name	Type	Description
0x0A0	GPIO_PE_DOUTSET	W1	Port Data Out Set Register
0x0A4	GPIO_PE_DOUTCLR	W1	Port Data Out Clear Register
0x0A8	GPIO_PE_DOUTTGL	W1	Port Data Out Toggle Register
0x0AC	GPIO_PE_DIN	R	Port Data In Register
0x0B0	GPIO_PE_PINLOCKN	RW	Port Unlocked Pins Register
0x0B4	GPIO_PF_CTRL	RW	Port Control Register
0x0B8	GPIO_PF_MODEL	RW	Port Pin Mode Low Register
0x0BC	GPIO_PF_MODEH	RW	Port Pin Mode High Register
0x0C0	GPIO_PF_DOUT	RW	Port Data Out Register
0x0C4	GPIO_PF_DOUTSET	W1	Port Data Out Set Register
0x0C8	GPIO_PF_DOUTCLR	W1	Port Data Out Clear Register
0x0CC	GPIO_PF_DOUTTGL	W1	Port Data Out Toggle Register
0x0D0	GPIO_PF_DIN	R	Port Data In Register
0x0D4	GPIO_PF_PINLOCKN	RW	Port Unlocked Pins Register
0x100	GPIO_EXTIPSELL	RW	External Interrupt Port Select Low Register
0x104	GPIO_EXTIPSELH	RW	External Interrupt Port Select High Register
0x108	GPIO_EXTIRISE	RW	External Interrupt Rising Edge Trigger Register
0x10C	GPIO_EXTIFALL	RW	External Interrupt Falling Edge Trigger Register
0x110	GPIO_IEN	RW	Interrupt Enable Register
0x114	GPIO_IF	R	Interrupt Flag Register
0x118	GPIO_IFS	W1	Interrupt Flag Set Register
0x11C	GPIO_IFC	W1	Interrupt Flag Clear Register
0x120	GPIO_ROUTE	RW	I/O Routing Register
0x124	GPIO_INSENSE	RW	Input Sense Register
0x128	GPIO_LOCK	RW	Configuration Lock Register
0x12C	GPIO_CTRL	RW	GPIO Control Register
0x130	GPIO_CMD	W1	GPIO Command Register
0x134	GPIO_EM4WUEN	RW	EM4 Wake-up Enable Register
0x138	GPIO_EM4WUPOL	RW	EM4 Wake-up Polarity Register
0x13C	GPIO_EM4WUCAUSE	R	EM4 Wake-up Cause Register

26.5 Register Description

26.5.1 GPIO_Px_CTRL - Port Control Register

Offset	Bit Position																													
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0x0	
Access																															RW	
Name																															DRIVEMODE	

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1:0	DRIVEMODE	0x0	RW	**Drive Mode Select**

Select drive mode for all pins on port configured with alternate drive strength.

Value	Mode	Description
0	STANDARD	6 mA drive current
1	LOWEST	0.1 mA drive current
2	HIGH	20 mA drive current
3	LOW	1 mA drive current

26.5.2 GPIO_Px_MODEL - Port Pin Mode Low Register

Offset	Bit Position																													
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x0				0x0				0x0				0x0				0x0				0x0				0x0				0x0			
Access	RW				RW				RW				RW				RW				RW				RW				RW			
Name	MODE7				MODE6				MODE5				MODE4				MODE3				MODE2				MODE1				MODE0			

Bit	Name	Reset	Access	Description
31:28	MODE7	0x0	RW	**Pin 7 Mode**
	Configure mode for pin 7. Enumeration is equal to MODE0.			
27:24	MODE6	0x0	RW	**Pin 6 Mode**
	Configure mode for pin 6. Enumeration is equal to MODE0.			
23:20	MODE5	0x0	RW	**Pin 5 Mode**
	Configure mode for pin 5. Enumeration is equal to MODE0.			
19:16	MODE4	0x0	RW	**Pin 4 Mode**
	Configure mode for pin 4. Enumeration is equal to MODE0.			
15:12	MODE3	0x0	RW	**Pin 3 Mode**
	Configure mode for pin 3. Enumeration is equal to MODE0.			
11:8	MODE2	0x0	RW	**Pin 2 Mode**
	Configure mode for pin 2. Enumeration is equal to MODE0.			
7:4	MODE1	0x0	RW	**Pin 1 Mode**
	Configure mode for pin 1. Enumeration is equal to MODE0.			
3:0	MODE0	0x0	RW	**Pin 0 Mode**
	Configure mode for pin 0.			

Value	Mode	Description
0	DISABLED	Input disabled. Pullup if DOUT is set.
1	INPUT	Input enabled. Filter if DOUT is set
2	INPUTPULL	Input enabled. DOUT determines pull direction
3	INPUTPULLFILTER	Input enabled with filter. DOUT determines pull direction
4	PUSHPULL	Push-pull output
5	PUSHPULLDRIVE	Push-pull output with drive-strength set by DRIVEMODE
6	WIREDOR	Wired-or output
7	WIREDORPULLDOWN	Wired-or output with pull-down
8	WIREDAND	Open-drain output
9	WIREDANDFILTER	Open-drain output with filter
10	WIREDANDPULLUP	Open-drain output with pullup
11	WIREDANDPULLUPFILTER	Open-drain output with filter and pullup
12	WIREDANDDRIVE	Open-drain output with drive-strength set by DRIVEMODE

Bit	Name	Reset	Access	Description

Value	Mode	Description
13	WIREDANDDRIVEFILTER	Open-drain output with filter and drive-strength set by DRIVEMODE
14	WIREDANDDRIVEPULLUP	Open-drain output with pullup and drive-strength set by DRIVEMODE
15	WIREDANDDRIVEPULLUPFILTER	Open-drain output with filter, pullup and drive-strength set by DRIVEMODE

26.5.3 GPIO_Px_MODEH - Port Pin Mode High Register

Offset	Bit Position																													
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x0				0x0				0x0				0x0				0x0				0x0				0x0				0x0			
Access	RW				RW				RW				RW				RW				RW				RW				RW			
Name	MODE15				MODE14				MODE13				MODE12				MODE11				MODE10				MODE9				MODE8			

Bit	Name	Reset	Access	Description
31:28	MODE15	0x0	RW	**Pin 15 Mode**
	Configure mode for pin 15. Enumeration is equal to MODE8.			
27:24	MODE14	0x0	RW	**Pin 14 Mode**
	Configure mode for pin 14. Enumeration is equal to MODE8.			
23:20	MODE13	0x0	RW	**Pin 13 Mode**
	Configure mode for pin 13. Enumeration is equal to MODE8.			
19:16	MODE12	0x0	RW	**Pin 12 Mode**
	Configure mode for pin 12. Enumeration is equal to MODE8.			
15:12	MODE11	0x0	RW	**Pin 11 Mode**
	Configure mode for pin 11. Enumeration is equal to MODE8.			
11:8	MODE10	0x0	RW	**Pin 10 Mode**
	Configure mode for pin 10. Enumeration is equal to MODE8.			
7:4	MODE9	0x0	RW	**Pin 9 Mode**
	Configure mode for pin 9. Enumeration is equal to MODE8.			
3:0	MODE8	0x0	RW	**Pin 8 Mode**
	Configure mode for pin 8.			

Value	Mode	Description
0	DISABLED	Input disabled. Pullup if DOUT is set.
1	INPUT	Input enabled. Filter if DOUT is set
2	INPUTPULL	Input enabled. DOUT determines pull direction
3	INPUTPULLFILTER	Input enabled with filter. DOUT determines pull direction
4	PUSHPULL	Push-pull output
5	PUSHPULLDRIVE	Push-pull output with drive-strength set by DRIVEMODE
6	WIREDOR	Wired-or output
7	WIREDORPULLDOWN	Wired-or output with pull-down
8	WIREDAND	Open-drain output
9	WIREDANDFILTER	Open-drain output with filter
10	WIREDANDPULLUP	Open-drain output with pullup
11	WIREDANDPULLUPFILTER	Open-drain output with filter and pullup
12	WIREDANDDRIVE	Open-drain output with drive-strength set by DRIVEMODE
13	WIREDANDDRIVEFILTER	Open-drain output with filter and drive-strength set by DRIVEMODE
14	WIREDANDDRIVEPULLUP	Open-drain output with pullup and drive-strength set by DRIVEMODE
15	WIREDANDDRIVEPULLUPFILTER	Open-drain output with filter, pullup and drive-strength set by DRIVEMODE

26.5.4 GPIO_Px_DOUT - Port Data Out Register

Offset	Bit Position																													
0x00C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	DOUT															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	DOUT	0x0000	RW	**Data Out**
	Data output on port.			

26.5.5 GPIO_Px_DOUTSET - Port Data Out Set Register

Offset	Bit Position																													
0x010	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	W1															
Name																	DOUTSET															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	DOUTSET	0x0000	W1	**Data Out Set**
	Write bits to 1 to set corresponding bits in GPIO_Px_DOUT. Bits written to 0 will have no effect.			

26.5.6 GPIO_Px_DOUTCLR - Port Data Out Clear Register

Offset	Bit Position																													
0x014	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	W1															
Name																	DOUTCLR															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	DOUTCLR	0x0000	W1	**Data Out Clear**
	Write bits to 1 to clear corresponding bits in GPIO_Px_DOUT. Bits written to 0 will have no effect.			

26.5.7 GPIO_Px_DOUTTGL - Port Data Out Toggle Register

Offset	Bit Position																													
0x018	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	W1															
Name																	DOUTTGL															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	DOUTTGL	0x0000	W1	**Data Out Toggle**
	Write bits to 1 to toggle corresponding bits in GPIO_Px_DOUT. Bits written to 0 will have no effect.			

26.5.8 GPIO_Px_DIN - Port Data In Register

Offset	Bit Position																													
0x01C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	R															
Name																	DIN															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	DIN	0x0000	R	**Data In**
	Port data input.			

26.5.9 GPIO_Px_PINLOCKN - Port Unlocked Pins Register

Offset	Bit Position																													
0x020	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0xFFFF														
Access																	RW														
Name																	PINLOCKN														

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	PINLOCKN	0xFFFF	RW	**Unlocked Pins**
	Shows unlocked pins in the port. To lock pin n, clear bit n. The pin is then locked until reset.			

26.5.10 GPIO_EXTIPSELL - External Interrupt Port Select Low Register

Offset	Bit Position																													
0x100	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset		0x0				0x0				0x0				0x0				0x0				0x0				0x0				0x0		
Access		RW				RW				RW				RW				RW				RW				RW				RW		
Name		EXTIPSEL7				EXTIPSEL6				EXTIPSEL5				EXTIPSEL4				EXTIPSEL3				EXTIPSEL2				EXTIPSEL1				EXTIPSEL0		

Bit	Name	Reset	Access	Description
31	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
30:28	EXTIPSEL7	0x0	RW	**External Interrupt 7 Port Select**
	Select input port for external interrupt 7.			

Value	Mode	Description
0	PORTA	Port A pin 7 selected for external interrupt 7
1	PORTB	Port B pin 7 selected for external interrupt 7
2	PORTC	Port C pin 7 selected for external interrupt 7
3	PORTD	Port D pin 7 selected for external interrupt 7
4	PORTE	Port E pin 7 selected for external interrupt 7
5	PORTF	Port F pin 7 selected for external interrupt 7

27	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
26:24	EXTIPSEL6	0x0	RW	**External Interrupt 6 Port Select**
	Select input port for external interrupt 6.			

Value	Mode	Description
0	PORTA	Port A pin 6 selected for external interrupt 6
1	PORTB	Port B pin 6 selected for external interrupt 6
2	PORTC	Port C pin 6 selected for external interrupt 6
3	PORTD	Port D pin 6 selected for external interrupt 6
4	PORTE	Port E pin 6 selected for external interrupt 6
5	PORTF	Port F pin 6 selected for external interrupt 6

Bit	Name	Reset	Access	Description
23	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
22:20	EXTIPSEL5	0x0	RW	**External Interrupt 5 Port Select**

Select input port for external interrupt 5.

Value	Mode	Description
0	PORTA	Port A pin 5 selected for external interrupt 5
1	PORTB	Port B pin 5 selected for external interrupt 5
2	PORTC	Port C pin 5 selected for external interrupt 5
3	PORTD	Port D pin 5 selected for external interrupt 5
4	PORTE	Port E pin 5 selected for external interrupt 5
5	PORTF	Port F pin 5 selected for external interrupt 5

19	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
18:16	EXTIPSEL4	0x0	RW	**External Interrupt 4 Port Select**

Select input port for external interrupt 4.

Value	Mode	Description
0	PORTA	Port A pin 4 selected for external interrupt 4
1	PORTB	Port B pin 4 selected for external interrupt 4
2	PORTC	Port C pin 4 selected for external interrupt 4
3	PORTD	Port D pin 4 selected for external interrupt 4
4	PORTE	Port E pin 4 selected for external interrupt 4
5	PORTF	Port F pin 4 selected for external interrupt 4

15	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
14:12	EXTIPSEL3	0x0	RW	**External Interrupt 3 Port Select**

Select input port for external interrupt 3.

Value	Mode	Description
0	PORTA	Port A pin 3 selected for external interrupt 3
1	PORTB	Port B pin 3 selected for external interrupt 3
2	PORTC	Port C pin 3 selected for external interrupt 3
3	PORTD	Port D pin 3 selected for external interrupt 3
4	PORTE	Port E pin 3 selected for external interrupt 3
5	PORTF	Port F pin 3 selected for external interrupt 3

11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	EXTIPSEL2	0x0	RW	**External Interrupt 2 Port Select**

Select input port for external interrupt 2.

Value	Mode	Description
0	PORTA	Port A pin 2 selected for external interrupt 2
1	PORTB	Port B pin 2 selected for external interrupt 2
2	PORTC	Port C pin 2 selected for external interrupt 2
3	PORTD	Port D pin 2 selected for external interrupt 2
4	PORTE	Port E pin 2 selected for external interrupt 2
5	PORTF	Port F pin 2 selected for external interrupt 2

7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:4	EXTIPSEL1	0x0	RW	**External Interrupt 1 Port Select**

Select input port for external interrupt 1.

Value	Mode	Description
0	PORTA	Port A pin 1 selected for external interrupt 1
1	PORTB	Port B pin 1 selected for external interrupt 1
2	PORTC	Port C pin 1 selected for external interrupt 1
3	PORTD	Port D pin 1 selected for external interrupt 1
4	PORTE	Port E pin 1 selected for external interrupt 1

Bit	Name	Reset	Access	Description

Value	Mode	Description
5	PORTF	Port F pin 1 selected for external interrupt 1

3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2:0	EXTIPSEL0	0x0	RW	**External Interrupt 0 Port Select**

Select input port for external interrupt 0.

Value	Mode	Description
0	PORTA	Port A pin 0 selected for external interrupt 0
1	PORTB	Port B pin 0 selected for external interrupt 0
2	PORTC	Port C pin 0 selected for external interrupt 0
3	PORTD	Port D pin 0 selected for external interrupt 0
4	PORTE	Port E pin 0 selected for external interrupt 0
5	PORTF	Port F pin 0 selected for external interrupt 0

26.5.11 GPIO_EXTIPSELH - External Interrupt Port Select High Register

Offset	Bit Position																														
0x104	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset		0x0				0x0				0x0				0x0				0x0				0x0				0x0				0x0		
Access		RW				RW				RW				RW				RW				RW				RW				RW		
Name		EXTIPSEL15				EXTIPSEL14				EXTIPSEL13				EXTIPSEL12				EXTIPSEL11				EXTIPSEL10				EXTIPSEL9				EXTIPSEL8		

Bit	Name	Reset	Access	Description
31	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
30:28	EXTIPSEL15	0x0	RW	**External Interrupt 15 Port Select**

Select input port for external interrupt 15.

Value	Mode	Description
0	PORTA	Port A pin 15 selected for external interrupt 15
1	PORTB	Port B pin 15 selected for external interrupt 15
2	PORTC	Port C pin 15 selected for external interrupt 15
3	PORTD	Port D pin 15 selected for external interrupt 15
4	PORTE	Port E pin 15 selected for external interrupt 15
5	PORTF	Port F pin 15 selected for external interrupt 15

27	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
26:24	EXTIPSEL14	0x0	RW	**External Interrupt 14 Port Select**

Select input port for external interrupt 14.

Value	Mode	Description
0	PORTA	Port A pin 14 selected for external interrupt 14
1	PORTB	Port B pin 14 selected for external interrupt 14
2	PORTC	Port C pin 14 selected for external interrupt 14
3	PORTD	Port D pin 14 selected for external interrupt 14
4	PORTE	Port E pin 14 selected for external interrupt 14
5	PORTF	Port F pin 14 selected for external interrupt 14

23	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
22:20	EXTIPSEL13	0x0	RW	**External Interrupt 13 Port Select**

Select input port for external interrupt 13.

Bit	Name	Reset	Access	Description

Value	Mode	Description
0	PORTA	Port A pin 13 selected for external interrupt 13
1	PORTB	Port B pin 13 selected for external interrupt 13
2	PORTC	Port C pin 13 selected for external interrupt 13
3	PORTD	Port D pin 13 selected for external interrupt 13
4	PORTE	Port E pin 13 selected for external interrupt 13
5	PORTF	Port F pin 13 selected for external interrupt 13

19	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
18:16	EXTIPSEL12	0x0	RW	**External Interrupt 12 Port Select**

Select input port for external interrupt 12.

Value	Mode	Description
0	PORTA	Port A pin 12 selected for external interrupt 12
1	PORTB	Port B pin 12 selected for external interrupt 12
2	PORTC	Port C pin 12 selected for external interrupt 12
3	PORTD	Port D pin 12 selected for external interrupt 12
4	PORTE	Port E pin 12 selected for external interrupt 12
5	PORTF	Port F pin 12 selected for external interrupt 12

15	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
14:12	EXTIPSEL11	0x0	RW	**External Interrupt 11 Port Select**

Select input port for external interrupt 11.

Value	Mode	Description
0	PORTA	Port A pin 11 selected for external interrupt 11
1	PORTB	Port B pin 11 selected for external interrupt 11
2	PORTC	Port C pin 11 selected for external interrupt 11
3	PORTD	Port D pin 11 selected for external interrupt 11
4	PORTE	Port E pin 11 selected for external interrupt 11
5	PORTF	Port F pin 11 selected for external interrupt 11

11	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
10:8	EXTIPSEL10	0x0	RW	**External Interrupt 10 Port Select**

Select input port for external interrupt 10.

Value	Mode	Description
0	PORTA	Port A pin 10 selected for external interrupt 10
1	PORTB	Port B pin 10 selected for external interrupt 10
2	PORTC	Port C pin 10 selected for external interrupt 10
3	PORTD	Port D pin 10 selected for external interrupt 10
4	PORTE	Port E pin 10 selected for external interrupt 10
5	PORTF	Port F pin 10 selected for external interrupt 10

7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:4	EXTIPSEL9	0x0	RW	**External Interrupt 9 Port Select**

Select input port for external interrupt 9.

Value	Mode	Description
0	PORTA	Port A pin 9 selected for external interrupt 9
1	PORTB	Port B pin 9 selected for external interrupt 9
2	PORTC	Port C pin 9 selected for external interrupt 9
3	PORTD	Port D pin 9 selected for external interrupt 9
4	PORTE	Port E pin 9 selected for external interrupt 9
5	PORTF	Port F pin 9 selected for external interrupt 9

3	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
2:0	EXTIPSEL8	0x0	RW	**External Interrupt 8 Port Select**

Bit	Name	Reset	Access	Description

Select input port for external interrupt 8.

Value	Mode	Description
0	PORTA	Port A pin 8 selected for external interrupt 8
1	PORTB	Port B pin 8 selected for external interrupt 8
2	PORTC	Port C pin 8 selected for external interrupt 8
3	PORTD	Port D pin 8 selected for external interrupt 8
4	PORTE	Port E pin 8 selected for external interrupt 8
5	PORTF	Port F pin 8 selected for external interrupt 8

26.5.12 GPIO_EXTIRISE - External Interrupt Rising Edge Trigger Register

Offset	Bit Position																															
0x108	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	EXTIRISE															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	EXTIRISE	0x0000	RW	**External Interrupt n Rising Edge Trigger Enable**

Set bit n to enable triggering of external interrupt n on rising edge.

Value	Description
EXTIRISE[n] = 0	Rising edge trigger disabled
EXTIRISE[n] = 1	Rising edge trigger enabled

26.5.13 GPIO_EXTIFALL - External Interrupt Falling Edge Trigger Register

Offset	Bit Position																															
0x10C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	EXTIFALL															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	EXTIFALL	0x0000	RW	**External Interrupt n Falling Edge Trigger Enable**

Bit	Name	Reset	Access	Description

Set bit n to enable triggering of external interrupt n on falling edge.

Value	Description
EXTIFALL[n] = 0	Falling edge trigger disabled
EXTIFALL[n] = 1	Falling edge trigger enabled

26.5.14 GPIO_IEN - Interrupt Enable Register

Offset	Bit Position																															
0x110	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	EXT															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	EXT	0x0000	RW	**External Interrupt n Enable**

Set bit n to enable external interrupt from pin n.

Value	Description
EXT[n] = 0	Pin n external interrupt disabled
EXT[n] = 1	Pin n external interrupt enabled

26.5.15 GPIO_IF - Interrupt Flag Register

Offset	Bit Position																															
0x114	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	R															
Name																	EXT															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	EXT	0x0000	R	**External Interrupt Flag n**

Pin n external interrupt flag.

Value	Description
EXT[n] = 0	Pin n external interrupt flag cleared

Bit	Name	Reset	Access	Description

Value	Description
EXT[n] = 1	Pin n external interrupt flag set

26.5.16 GPIO_IFS - Interrupt Flag Set Register

Offset	Bit Position																															
0x118	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	W1															
Name																	EXT															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	EXT	0x0000	W1	**External Interrupt Flag n Set**

Write bit n to 1 to set interrupt flag n.

Value	Description
EXT[n] = 0	Pin n external interrupt flag unchanged
EXT[n] = 1	Pin n external interrupt flag set

26.5.17 GPIO_IFC - Interrupt Flag Clear Register

Offset	Bit Position																															
0x11C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	W1															
Name																	EXT															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	EXT	0x0000	W1	**External Interrupt Flag Clear**

Write bit n to 1 to clear external interrupt flag n.

Value	Description
EXT[n] = 0	Pin n external interrupt flag unchanged
EXT[n] = 1	Pin n external interrupt flag cleared

26.5.18 GPIO_ROUTE - I/O Routing Register

Offset	Bit Position																													
0x120	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															1	1
Access																															RW	RW
Name																															SWDIOPEN	SWCLKPEN

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	SWDIOPEN	1	RW	**Serial Wire Data Pin Enable**
	Enable Serial Wire Data connection to pin. WARNING: When this pin is disabled, the device can no longer be accessed by a debugger. A reset will set the pin back to a default state as enabled. If you disable this pin, make sure you have at least a 3 second timeout at the start of you program code before you disable the pin. This way, the debugger will have time to halt the device after a reset before the pin is disabled.			
0	SWCLKPEN	1	RW	**Serial Wire Clock Pin Enable**
	Enable Serial Wire Clock connection to pin. WARNING: When this pin is disabled, the device can no longer be accessed by a debugger. A reset will set the pin back to a default state as enabled. If you disable this pin, make sure you have at least a 3 second timeout at the start of you program code before you disable the pin. This way, the debugger will have time to halt the device after a reset before the pin is disabled.			

26.5.19 GPIO_INSENSE - Input Sense Register

Offset	Bit Position																													
0x124	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															1	1
Access																															RW	RW
Name																															PRS	INT

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
1	PRS	1	RW	**PRS Sense Enable**
	Set this bit to enable input sensing for PRS.			
0	INT	1	RW	**Interrupt Sense Enable**
	Set this bit to enable input sensing for interrupts.			

26.5.20 GPIO_LOCK - Configuration Lock Register

Offset	Bit Position																															
0x128	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																	0x0000															
Access																	RW															
Name																	LOCKKEY															

Bit	Name	Reset	Access	Description
31:16	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
15:0	LOCKKEY	0x0000	RW	**Configuration Lock Key**

Write any other value than the unlock code to lock MODEL, MODEH, CTRL, PINLOCKN, EPISELL, EIPSELH, INSENSE and SWDPROUTE from editing. Write the unlock code to unlock. When reading the register, bit 0 is set when the lock is enabled.

Mode	Value	Description
Read Operation		
UNLOCKED	0	GPIO registers are unlocked
LOCKED	1	GPIO registers are locked
Write Operation		
LOCK	0	Lock GPIO registers
UNLOCK	0xA534	Unlock GPIO registers

26.5.21 GPIO_CTRL - GPIO Control Register

Offset	Bit Position																															
0x12C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																RW
Name																																EM4RET

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	EM4RET	0	RW	**Enable EM4 retention**

Set to enable EM4 retention of output enable, output value and pull enable.

26.5.22 GPIO_CMD - GPIO Command Register

Offset	Bit Position																													
0x130	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																																0
Access																																W1
Name																																EM4WUCLR

Bit	Name	Reset	Access	Description
31:1	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
0	EM4WUCLR	0	W1	**EM4 Wake-up clear**
	Write 1 to clear all wake-up requests.			

26.5.23 GPIO_EM4WUEN - EM4 Wake-up Enable Register

Offset	Bit Position																													
0x134	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																										0x00						
Access																										RW						
Name																										EM4WUEN						

Bit	Name	Reset	Access	Description
31:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:0	EM4WUEN	0x00	RW	**EM4 Wake-up enable**
	Write 1 to enable wake-up request, write 0 to disable wake-up request.			

Value	Mode	Description
0x01	A0	Enable em4 wakeup on pin A0
0x04	C9	Enable em4 wakeup on pin C9
0x08	F1	Enable em4 wakeup on pin F1
0x10	F2	Enable em4 wakeup on pin F2
0x20	E13	Enable em4 wakeup on pin E13
0x40	C4	Enable em4 wakeup on pin C4

26.5.24 GPIO_EM4WUPOL - EM4 Wake-up Polarity Register

Offset	Bit Position																													
0x138	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																										0x00						
Access																										RW						
Name																										EM4WUPOL						

Bit	Name	Reset	Access	Description
31:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:0	EM4WUPOL	0x00	RW	**EM4 Wake-up Polarity**

Write bit n to 1 for high wake-up request. Write bit n to 0 for low wake-up request

Value	Mode	Description
0x01	A0	Determines polarity on pin A0
0x04	C9	Determines polarity on pin C9
0x08	F1	Determines polarity on pin F1
0x10	F2	Determines polarity on pin F2
0x20	E13	Determines polarity on pin E13
0x40	C4	Determines polarity on pin C4

26.5.25 GPIO_EM4WUCAUSE - EM4 Wake-up Cause Register

Offset	Bit Position																													
0x13C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																													0x00			
Access																													R			
Name																													EM4WUCAUSE			

Bit	Name	Reset	Access	Description
31:7	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
6:0	EM4WUCAUSE	0x00	R	**EM4 wake-up cause**

Bit n indicates which pin the wake-up request occurred.

Value	Mode	Description
0x01	A0	This bit indicates an em4 wake-up request occurred on pin A0
0x04	C9	This bit indicates an em4 wake-up request occurred on pin C9
0x08	F1	This bit indicates an em4 wake-up request occurred on pin F1
0x10	F2	This bit indicates an em4 wake-up request occurred on pin F2
0x20	E13	This bit indicates an em4 wake-up request occurred on pin E13
0x40	C4	This bit indicates an em4 wake-up request occurred on pin C4

27 Revision History

27.1 Revision 0.30

March 22nd, 2016

Corrected DI Page PART_FAMILY, RADIO_ID, and DI_CRC values.

Updated package description in the System Overview chapter to QFN48 for all parts.

27.2 Revision 0.20

February 3rd, 2015

Preliminary Release.

A Abbreviations

A.1 Abbreviations

This section lists abbreviations used in this document.

Table A.1. Abbreviations

Abbreviation	Description
ACMP	Analog Comparator
ADC	Analog to Digital Converter
AHB	AMBA Advanced High-performance Bus. AMBA is short for "Advanced Microcontroller Bus Architecture".
APB	AMBA Advanced Peripheral Bus. AMBA is short for "Advanced Microcontroller Bus Architecture".
ALE	Address Latch Enable
AUXHFRCO	Auxiliary High Frequency RC Oscillator.
CC	Compare / Capture
CLK	Clock
CMD	Command
CMU	Clock Management Unit
CTRL	Control
DAC	Digital to Analog Converter
DBG	Debug
DMA	Direct Memory Access
DRD	Dual Role Device
DTI	Dead Time Insertion
EFM	Energy Friendly Microcontroller
EM	Energy Mode
EM0	Energy Mode 0 (also called active mode)
EM1 to EM4	Energy Mode 1 to Energy Mode 4 (also called low energy modes)
EMU	Energy Management Unit
ENOB	Effective Number of Bits
FS	Full-speed
GPIO	General Purpose Input / Output
HFRCO	High Frequency RC Oscillator
HFXO	High Frequency Crystal Oscillator
HW	Hardware
I^2C	Inter-Integrated Circuit interface
LETIMER	Low Energy Timer
LEUART	Low Energy Universal Asynchronous Receiver Transmitter
LFRCO	Low Frequency RC Oscillator

Abbreviation	Description
LFXO	Low Frequency Crystal Oscillator
LS	Low-speed
MAC	Media Access Controller
NVIC	Nested Vector Interrupt Controller
OSR	Oversampling Ratio
OTG	On-the-go
PCNT	Pulse Counter
PHY	Physical Layer
PRS	Peripheral Reflex System
PWM	Pulse Width Modulation
RC	Resistance and Capacitance
RMU	Reset Management Unit
RTC	Real Time Clock
SAR	Successive Approximation Register
SOF	Start of Frame
SPI	Serial Peripheral Interface
SW	Software
USART	Universal Synchronous Asynchronous Receiver Transmitter
USB	Universal Serial Bus
VCMP	Voltage supply Comparator
WDOG	Watchdog timer
XTAL	Crystal

B Disclaimer and Trademarks

B.1 Disclaimer

Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products must not be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are generally not intended for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

B.2 Trademark Information

Silicon Laboratories Inc., Silicon Laboratories, Silicon Labs, SiLabs and the Silicon Labs logo, CMEMS®, EFM, EFM32, EFR, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZMac®, EZRadio®, EZRadioPRO®, DSPLL®, ISOmodem®, Precision32®, ProSLIC®, SiPHY®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

C Contact Information

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

1. Energy Friendly Wireless Microcontrollers 2
1.1. Typical Applications 2
1.2. EZR32HG Development 2
2. About This Document 3
2.1. Conventions 3
2.2. Related Documentation 4
3. System Overview 5
3.1. Introduction 5
3.2. Block Diagram 5
3.3. Features 6
3.4. Energy Modes 8
3.5. Product Overview 8
3.6. Device Revision 9
4. Radio Overview 10
4.1. EZRadioPRO Overview 10
4.2. EZRadio Overview 38
5. System Processor 54
5.1. Introduction 54
5.2. Features 54
5.3. Functional Description 55
6. Memory and Bus System 57
6.1. Introduction 57
6.2. Functional Description 58
6.3. Access to Low Energy Peripherals (Asynchronous Registers) 61
6.4. Flash 63
6.5. SRAM 64
6.6. Device Information (DI) Page 64
7. DBG - Debug Interface 66
7.1. Introduction 66
7.2. Features 66
7.3. Functional Description 66
7.4. Debug Lock and Device Erase 67
7.5. Register Map 69
7.6. Register Description 69
8. MSC - Memory System Controller 71
8.1. Introduction 71
8.2. Features 72
8.3. Functional Description 72
8.4. Register Map 78
8.5. Register Description 78
9. DMA - DMA Controller 88
9.1. Introduction 88
9.2. Features 88
9.3. Block Diagram 89
9.4. Functional Description 90
9.5. Examples 107
9.6. Register Map 109
9.7. Register Description 110
10. RMU - Reset Management Unit 125
10.1. Introduction 125
10.2. Features 125
10.3. Functional Description 125
10.4. Register Map 129
10.5. Register Description 129
11. EMU - Energy Management Unit 131
11.1. Introduction 131
11.2. Features 131
11.3. Functional Description 132
11.4. Register Map 137
11.5. Register Description 137
12. CMU - Clock Management Unit 139
12.1. Introduction 139
12.2. Features 139
12.3. Functional Description 140
12.4. Register Map 148
12.5. Register Description 149
13. WDOG - Watchdog Timer 171
13.1. Introduction 171
13.2. Features 171
13.3. Functional Description 171
13.4. Register Map 173

13.5. Register Description 173
14. PRS - Peripheral Reflex System 176
14.1. Introduction 176
14.2. Features 176
14.3. Functional Description 176
14.4. Register Map 181
14.5. Register Description 181
15. USB - Universal Serial Bus Controller 187
15.1. Introduction 187
15.2. Features 187
15.3. USB System Description 188
15.4. USB Core Description 192
15.5. Register Map 292
15.6. Register Description 294
16. I2C - Inter-Integrated Circuit Interface 334
16.1. Introduction 334
16.2. Features 334
16.3. Functional Description 335
16.4. Register Map 356
16.5. Register Description 356
17. USART - Universal Synchronous Asynchronous Receiver/Transmitter 368
17.1. Introduction 368
17.2. Features 368
17.3. Functional Description 369
17.4. Register Map 395
17.5. Register Description 395
18. LEUART - Low Energy Universal Asynchronous Receiver/Transmitter 415
18.1. Introduction 415
18.2. Features 415
18.3. Functional Description 416
18.4. Register Map 427
18.5. Register Description 427
19. TIMER - Timer/Counter 441
19.1. Introduction 441
19.2. Features 441
19.3. Functional Description 442
19.4. Register Map 460
19.5. Register Description 461
20. RTC - Real Time Counter 479
20.1. Introduction 479
20.2. Features 479
20.3. Functional Description 480
20.4. Register Map 483
20.5. Register Description 483
21. PCNT - Pulse Counter 488
21.1. Introduction 488
21.2. Features 488
21.3. Functional Description 488
21.4. Register Map 495
21.5. Register Description 495
22. VCMP - Voltage Comparator 505
22.1. Introduction 505
22.2. Features 505
22.3. Functional Description 506
22.4. Register Map 509
22.5. Register Description 509
23. ADC - Analog to Digital Converter 513
23.1. Introduction 513
23.2. Features 513
23.3. Functional Description 514
23.4. Register Map 524
23.5. Register Description 524
24. IDAC - Current Digital to Analog Converter 537
24.1. Introduction 537
24.2. Features 537
24.3. Functional Description 537
24.4. Register Map 541
24.5. Register Description 541
25. AES - Advanced Encryption Standard Accelerator 544
25.1. Introduction 544
25.2. Features 544
25.3. Functional Description 544
25.4. Register Map 548
25.5. Register Description 548

26. GPIO - General Purpose Input/Output 554
26.1. Introduction 554
26.2. Features 554
26.3. Functional Description 555
26.4. Register Map 562
26.5. Register Description 563
27. Revision History 579
27.1. Revision 0.30 579
27.2. Revision 0.20 579
A. Abbreviations 580
A.1. Abbreviations 580
B. Disclaimer and Trademarks 582
B.1. Disclaimer 582
B.2. Trademark Information 582
C. Contact Information 583
C.1. 583

List of Figures

3.1. Block Diagram of EZR32HG 5
3.2. Energy Mode Indicator 5
3.3. Revision Number Extraction 9
4.1. EZRadioPRO Block Diagram 11
4.2. Serial Interface Timing 13
4.3. SPI Write Command 13
4.4. SPI Read Command - Check CTS Value 14
4.5. SPI Read Command - Clock Out Read Data 14
4.6. State Machine Diagram 15
4.7. POR Timing Diagram 16
4.8. Start_TX Commands and Timing 18
4.9. RX Architecture vs. Data Rate 24
4.10. +20 dBm TX Power vs. PA_PWR_LVL 31
4.11. +20 dBm TX Power vs. VDD 31
4.12. +20 dBm TX Power vs. Temp 31
4.13. Capacitor Bank Frequency Offset Characteristics 32
4.14. TX and RX FIFOs 33
4.15. Packet Handler Structure 34
4.16. RX and TX LDC Sequences 35
4.17. Low Duty Cycle Mode for RX 35
4.18. Preamble Sense Mode 37
4.19. EZRadio Block Diagram 38
4.20. Tx Power vs PA_PWR_LVL and VDD 41
4.21. Capacitor Bank Frequency Offset Characteristics 42
4.22. Manchester Code Example 44
4.23. Preamble Sense Mode 45
4.24. Serial Interface Timing 46
4.25. SPI Write Command 46
4.26. SPI Read Command-Check CTS Value 47
4.27. SPI Read Command-Clock Out Read Data 47
4.28. State Machine Diagram 48
4.29. Start-Up Timing and Current Consumption using Shutdown State 49
4.30. Start-Up Timing and Current Consumption using Standby State 49
4.31. POR Timing Diagram 50
4.32. START_TX Commands and Timing 51
5.1. Interrupt Operation 55
6.1. EZR32HG Bus System 58
6.2. System Address Space 59
6.3. Write operation to Low Energy Peripherals 62
6.4. Read operation from Low Energy Peripherals 63
7.1. AAP - Authentication Access Port 67
7.2. Device Unlock 68
7.3. AAP Expansion 68
8.1. Instruction Cache 75
9.1. DMA Block Diagram 89
9.2. Polling flowchart 92
9.3. Ping-pong example 94
9.4. Memory scatter-gather example 97
9.5. Peripheral scatter-gather example 99
9.6. Memory map for 4 channels, including the alternate data structure 101
9.7. Detailed memory map for the 4 channels, including the alternate data structure 102
9.8. channel_cfg bit assignments 103
10.1. RMU Reset Input Sources and Connections. 126
10.2. RMU Power-on Reset Operation 127
10.3. RMU Brown-out Detector Operation 127
11.1. EMU Overview 132
11.2. EMU Energy Mode Transitions 133
12.1. CMU Overview 140
12.2. CMU Switching from HFRCO to HFXO before HFXO is ready 143
12.3. CMU Switching from HFRCO to HFXO after HFXO is ready 143
12.4. HFXO Pin Connection 144
12.5. LFXO Pin Connection 144
12.6. HW-support for RC Oscillator Calibration 146
12.7. Single Calibration (CONT=0) 146
12.8. Continuous Calibration (CONT=1) 146
14.1. PRS Overview 177
14.2. TIMER0 overflow starting ADC0 single conversions through PRS channel 5. 180
15.1. USB Block Diagram 188
15.2. Bus-powered Device 189
15.3. Self-powered Device 190
15.4. Self-powered Device (with bus-power switch) 191

15.5. Transmit Transaction-Level Operation in Slave Mode 198
15.6. Receive Transaction-Level Operation in Slave Mode 198
15.7. Transmit FIFO Write Task in Slave Mode 203
15.8. Receive FIFO Read Task in Slave Mode 203
15.9. Normal Bulk/Control OUT/SETUP and Bulk/Control IN Transactions in Slave Mode 205
15.10. Normal Bulk/Control OUT/SETUP and Bulk/Control IN Transactions in DMA Mode 210
15.11. Interrupt Service Routine for Bulk/Control OUT Transaction in DMA Mode 211
15.12. Normal Interrupt OUT/IN Transactions in Slave Mode 215
15.13. Normal Interrupt OUT/IN Transactions in DMA Mode 219
15.14. Normal Isochronous OUT/IN Transactions in Slave Mode 223
15.15. Normal Isochronous OUT/IN Transactions in DMA Mode 226
15.16. Processing a SETUP Packet 234
15.17. Two-Stage Control Transfer 237
15.18. Receive FIFO Packet Read in Slave Mode 238
15.19. Slave Mode Bulk OUT Transaction 242
15.20. ISOC OUT Application Flow for Periodic Transfer Interrupt Feature 247
15.21. Isochronous OUT Core Internal Flow for Periodic Transfer Interrupt Feature 248
15.22. Bulk IN Stall 252
15.23. USBTRDTIM Max Timing Case ERROR wrong image 255
15.24. Slave Mode Bulk IN Transaction 257
15.25. Slave Mode Bulk IN Transfer (Pipelined Transaction) 259
15.26. Slave Mode Bulk IN Two-Endpoint Transfer 260
15.27. Periodic IN Application Flow for Periodic Transfer Interrupt Feature 264
15.28. Periodic IN Core Internal Flow for Periodic Transfer Interrupt Feature 266
15.29. SRP Detection by Core When Operating as A-device 270
15.30. SRP Initiation by the Core When Acting as a B-Device 271
15.31. HNP When the Core is an A-Device 272
15.32. HNP When the Core is a B-Device 273
15.33. Core Interrupt Handler 281
16.1. I^2C Overview 335
16.2. I^2C-Bus Example 335
16.3. I^2C START and STOP Conditions 336
16.4. I^2C Bit Transfer on I^2C-Bus 336
16.5. I^2C Single Byte Write to Slave 337
16.6. I^2C Double Byte Read from Slave 337
16.7. I^2C Single Byte Write, then Repeated Start and Single Byte Read 337
16.8. I^2C Master Transmitter/Slave Receiver with 10-bit Address 338
16.9. I^2C Master Receiver/Slave Transmitter with 10-bit Address 338
16.10. I^2C Master State Machine 342
16.11. I^2C Slave State Machine 349
17.1. USART Overview 369
17.2. USART Asynchronous Frame Format 370
17.3. USART Transmit Buffer Operation 374
17.4. USART Receive Buffer Operation 376
17.5. USART Sampling of Start and Data Bits 377
17.6. USART Sampling of Stop Bits when Number of Stop Bits are 1 or More 378
17.7. USART Local Loopback 379
17.8. USART Half Duplex Communication with External Driver 380
17.9. USART Transmission of Large Frames 381
17.10. USART Transmission of Large Frames, MSBF 381
17.11. USART Reception of Large Frames 382
17.12. USART ISO 7816 Data Frame Without Error 383
17.13. USART ISO 7816 Data Frame With Error 384
17.14. USART SmartCard Stop Bit Sampling 384
17.15. USART SPI Timing 386
17.16. USART SPI timing with SMSDELAY 387
17.17. USART SPI Slave Timing with SSSEARLY 388
17.18. USART Standard I2S waveform 389
17.19. USART Standard I2S waveform (reduced accuracy) 390
17.20. USART Left-justified I2S waveform 390
17.21. USART Right-justified I2S waveform 390
17.22. USART Mono I2S waveform 391
17.23. USART Example RZI Signal for a given Asynchronous USART Frame 393
18.1. LEUART Overview 416
18.2. LEUART Asynchronous Frame Format 417
18.3. LEUART Transmitter Overview 419
18.4. LEUART Receiver Overview 421
18.5. LEUART Local Loopback 424
18.6. LEUART Half Duplex Communication with External Driver 424
18.7. LEUART - NRZ vs. RZI 426
19.1. TIMER Block Overview 443
19.2. TIMER Hardware Timer/Counter Control 444
19.3. TIMER Clock Selection 444
19.4. TIMER Connections 445

19.5. TIMER TOP Value Update Functionality 445
19.6. TIMER Quadrature Encoded Inputs 446
19.7. TIMER Quadrature Decoder Configuration 446
19.8. TIMER X2 Decoding Mode 447
19.9. TIMER X4 Decoding Mode 447
19.10. TIMER Input Pin Logic 448
19.11. TIMER Input Capture Buffer Functionality 449
19.12. TIMER Output Compare/PWM Buffer Functionality 449
19.13. TIMER Input Capture 450
19.14. TIMER Period and/or Pulse width Capture 450
19.15. TIMER Block Diagram Showing Comparison Functionality 451
19.16. TIMER Output Logic 451
19.17. TIMER Up-count Frequency Generation 452
19.18. TIMER Up-count PWM Generation 452
19.19. TIMER CC out in 2x mode 453
19.20. TIMER Up/Down-count PWM Generation 454
19.21. TIMER CC out in 2x mode 454
19.22. TIMER Dead-Time Insertion Unit Overview 455
19.23. TIMER Triple Half-Bridge 455
19.24. TIMER Overview of Dead-Time Insertion Block for a Single PWM channel 456
19.25. TIMER Polarity of Both Signals are Set as Active-High 456
19.26. TIMER Output Polarities 457
20.1. RTC Overview 480
21.1. PCNT Overview 489
21.2. PCNT Quadrature Coding 490
21.3. PCNT Triggered compare and clear 492
21.4. PCNT Direction Change Interrupt (DIRCNG) Generation 494
22.1. VCMP Overview 506
22.2. VCMP 20 mV Hysteresis Enabled 507
23.1. ADC Overview 515
23.2. ADC Conversion Timing 516
23.3. ADC Analog Power Consumption With Different WARMUPMODE Settings 517
23.4. ADC RC Input Filter Configuration 518
23.5. ADC Bias Programming 519
23.6. ADC Conversion Tailgating 520
24.1. IDAC Overview 538
24.2. IDAC Charge Injection Example 539
25.1. AES Key and Data Definitions 545
25.2. AES Data and Key Orientation as Defined in the Advanced Encryption Standard 545
25.3. AES Data and Key Register Operation 546
26.1. Pin Configuration 556
26.2. Tristated Output with Optional Pull-up or Pull-down 557
26.3. Push-Pull Configuration 558
26.4. Open-drain 558
26.5. EM4 Wake-up Logic 559
26.6. Pin n Interrupt Generation 560

List of Tables

2.1. Register Access Types 3
3.1. Energy Mode Description 8
3.2. EZR32HG Wireless Microcontroller Series 9
3.3. Minor Revision Number Interpretation 9
4.1. Serial Interface Timing Parameters 12
4.2. Operating State Response Time and Current Consumption 16
4.3. POR Timing 17
4.4. Energy Mode Description 20
4.5. Recommended Preamble Length 23
4.6. Output Divider (Outdiv) Values for the EZRadioPRO 27
4.7. Ramp Times as a Function of TC[3:0] Value 30
4.8. Command 30
4.9. Data Rates 37
4.10. Energy Mode Description 38
4.11. Output Divider (Outdiv) Values 41
4.12. Data Rates 45
4.13. Serial Interface Timing Parameters 46
4.14. Operating State Response Time and Current Consumption 48
4.15. POR Timing 50
4.16. GPIOs 52
4.17. Packet Structure for Fixed Packet Length 53
5.1. Interrupt Request Lines (IRQ) 55
6.1. Memory System Core Peripherals 59
6.2. Memory System Low Energy Peripherals 60
6.3. Memory System Peripherals 60
6.4. Device Information Page Contents 64
8.1. MSC Flash Memory Mapping 73
8.2. Lock Bits Page Structure 73
9.1. AHB bus transfer arbitration interval 91
9.2. DMA channel priority 91
9.3. DMA cycle types 92
9.4. channel_cfg for a primary data structure, in memory scatter-gather mode 96
9.5. channel_cfg for a primary data structure, in peripheral scatter-gather mode 98
9.6. Address bit settings for the channel control data structure 101
9.7. src_data_end_ptr bit assignments 102
9.8. dst_data_end_ptr bit assignments 103
9.9. channel_cfg bit assignments 103
9.10. DMA cycle of six words using a word increment 106
9.11. DMA cycle of 12 bytes using a halfword increment 107
10.1. RMU Reset Cause Register Interpretation 126
11.1. EMU Energy Mode Overview 134
11.2. EMU Entering a Low Energy Mode 135
11.3. EMU Wakeup Triggers from Low Energy Modes 136
14.1. Reflex Producers 178
14.2. Reflex Consumers 179
15.1. Host Programming Operations 202
15.2. 231
15.3. 275
15.4. 276
16.1. I^2C Reserved I^2C Addresses 337
16.2. I^2C High and Low Periods for Low CLKDIV 339
16.3. I^2C Clock Mode 340
16.4. I^2C Interactions in Prioritized Order 343
16.5. I^2C Master Transmitter 345
16.6. I^2C Master Receiver 347
16.7. I^2C STATE Values 348
16.8. I^2C Transmission Status 348
16.9. I^2C Slave Transmitter 351
16.10. I^2C - Slave Receiver 352
16.11. I^2C Bus Error Response 353
17.1. USART Asynchronous vs. Synchronous Mode 370
17.2. USART Pin Usage 370
17.3. USART Data Bits 371
17.4. USART Stop Bits 371
17.5. USART Parity Bits 372
17.6. USART Oversampling 372
17.7. USART Baud Rates @ 4MHz Peripheral Clock 373
17.8. USART SPI Modes 385
17.9. USART I2S Modes 389
17.10. USART IrDA Pulse Widths 394
18.1. LEUART Parity Bit 417

18.2. LEUART Baud Rates 418
19.1. TIMER Counter Response in X2 Decoding Mode 447
19.2. TIMER Counter Response in X4 Decoding Mode 447
19.3. TIMER Events 459
20.1. RTC Resolution Vs Overflow 481
21.1. PCNT QUAD Mode Counter Control Function 491
22.1. Bias Configuration 506
23.1. ADC Single Ended Conversion 520
23.2. ADC Differential Conversion 521
23.3. Oversampling Result Shifting and Resolution 521
23.4. ADC Results Representation 522
23.5. Calibration Register Effect 523
24.1. Range Selection 538
26.1. Pin Configuration 556
26.2. EM4 WU Register bits to pin mapping 559
A.1. Abbreviations 580

List of Examples

9.1. DMA Transfer 108
17.1. USART Multi-processor Mode Example 382
19.1. TIMER DTI Example 1 457
19.2. TIMER DTI Example 2 457
25.1. AES Cipher Block Chaining 547
26.1. GPIO Interrupt Example 561

List of Equations

4.1. RF Channel Equation 27
4.2. RF Frequency Equation 28
4.3. Wake-up Time Equation 34
4.4. LDC Wake-up Time Equation 35
4.5. RSSI Equation 40
4.6. RF Frequency Equation 40
4.7. RF Frequency Equation 41
6.1. Memory Wait Cycles with Clock Equal or Faster than HFCORECLK 61
6.2. Memory Wait Cycles with Clock Slower than CPU 61
13.1. WDOG Timeout Equation 172
16.1. I^2C Pull-up Resistor Equation 335
16.2. I^2C Maximum Transmission Rate 339
16.3. I^2C High and Low Cycles Equations 339
16.4. Maximum Data Hold Time 339
17.1. USART Baud Rate 372
17.2. USART Desired Baud Rate 372
17.3. USART Synchronous Mode Bit Rate 385
17.4. USART Synchronous Mode Clock Division Factor 385
18.1. LEUART Baud Rate Equation 418
18.2. LEUART CLKDIV Equation 418
18.3. LEUART Optimal Sampling Point 422
18.4. LEUART Actual Sampling Point 422
19.1. TIMER Rotational Position Equation 447
19.2. TIMER Up-count Frequency Generation Equation 452
19.3. TIMER Up-count PWM Resolution Equation 452
19.4. TIMER Up-count PWM Frequency Equation 452
19.5. TIMER Up-count Duty Cycle Equation 453
19.6. TIMER 2x PWM Resolution Equation 453
19.7. TIMER 2x Mode PWM Frequency Equation(Up-count) 453
19.8. TIMER 2x Mode Duty Cycle Equation 453
19.9. TIMER Up/Down-count PWM Resolution Equation 454
19.10. TIMER Up/Down-count PWM Frequency Equation 454
19.11. TIMER Up/Down-count Duty Cycle Equation 454
19.12. TIMER 2x PWM Resolution Equation 454
19.13. TIMER 2x Mode PWM Frequency Equation(Up/Down-count) 455
19.14. TIMER 2x Mode Duty Cycle Equation 455
20.1. RTC Frequency Equation 480
21.1. Absolute position with hysteresis and even TOP value 491
21.2. Absolute position with hysteresis and odd TOP value 491
22.1. VCMP V_{DD} Trigger Level 507
23.1. ADC Total Conversion Time (in ADC_CLK cycles) Per Output 515
23.2. ADC Temperature Measurement 518

silabs.com