EZ-USB® FX3™ Technical Reference Manual

Spec No.: 001-76074 Rev. *E

May 31, 2017

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
www.cypress.com

Copyrights

Contents

1. Introduction to EZ-USB FX3

EZ-USB® FX3™ is Cypress's high-bandwidth USB 3.0 peripheral controller that provides integrated and flexible features. FX3 integrates the USB 3.0 and USB 2.0 physical layers (PHYs) along with a 32-bit ARM926EJ-S microprocessor for powerful data processing and for building custom USB SuperSpeed applications.

To provide high-bandwidth access to USB 3.0 data, FX3 contains a hardware unit called General Programmable Interface, Generation 2 (GPIF II). GPIF II is an enhanced version of the GPIF in FX2LP™, Cypress's USB 2.0 product. GPIF II provides easy and glueless connectivity to popular interfaces such as asynchronous SRAM and asynchronous and synchronous address and data multiplexed interfaces. FX3 implements a DMA-centric architecture that enables direct 375-MBps data transfer from GPIF II to the USB interface without CPU intervention.

An integrated USB 2.0 USB On-The-Go (OTG) controller enables applications in which FX3 may serve dual high-speed roles; for example, EZ-USB FX3 may function as a High-Speed On-The-Go (HS-OTG) host to USB Mass Storage Class (MSC) devices and HID-class devices. FX3 contains 512 KB or 256 KB of on-chip SRAM for code and data. EZ-USB FX3 also provides interfaces to connect to serial peripherals such as UART, SPI, I2C, and I2S. FX3 comes with application development tools. The software development kit (SDK) provides application examples for accelerating the time to market. FX3 complies with the USB 3.0 v1.0 specification and is also backward compatible with USB 2.0. It also complies with the Battery Charging Specification v1.1 and USB 2.0 OTG Specification v2.0.

In addition to these features, FX3S has an integrated storage controller and can support up to two independent mass storage devices. It can support SD 3.0 and eMMC 4.41 memory cards. It can also support SDIO on these ports.

This TRM describes the following functional blocks of the FX3/FX3S device: CPU subsystem, memory, global control, DMA, USB, GPIF II, low-bandwidth (serial and GPIO) peripherals, and storage (storage block is specific to FX3S only). Registers associated with these functional blocks are documented in the Registers chapter on page 235.

The following sections describe the details of USB 3.0 and briefly outline the EZ-USB FX3/FX3S architecture.

1.1 Overview of USB 3.0

This section gives an overview of USB 3.0 and describes the significant changes in each layer from the USB 2.0 specification, including the new power management features provided in USB 3.0. Refer to AN57294 - USB 101: An Introduction to Universal Serial Bus 2.0 for more details on USB 2.0. The USB 2.0 and 3.0 specifications can be downloaded from http://www.usb.org/developers/docs.

The USB 3.0 specification, released in 2008, allows a maximum signaling rate of 5 Gbps (SuperSpeed), which is 10 times the signaling rate of High Speed. The USB 3.0 architecture contains three layers: the physical layer, the link layer on top of the physical layer, and the protocol layer on top of the link layer.

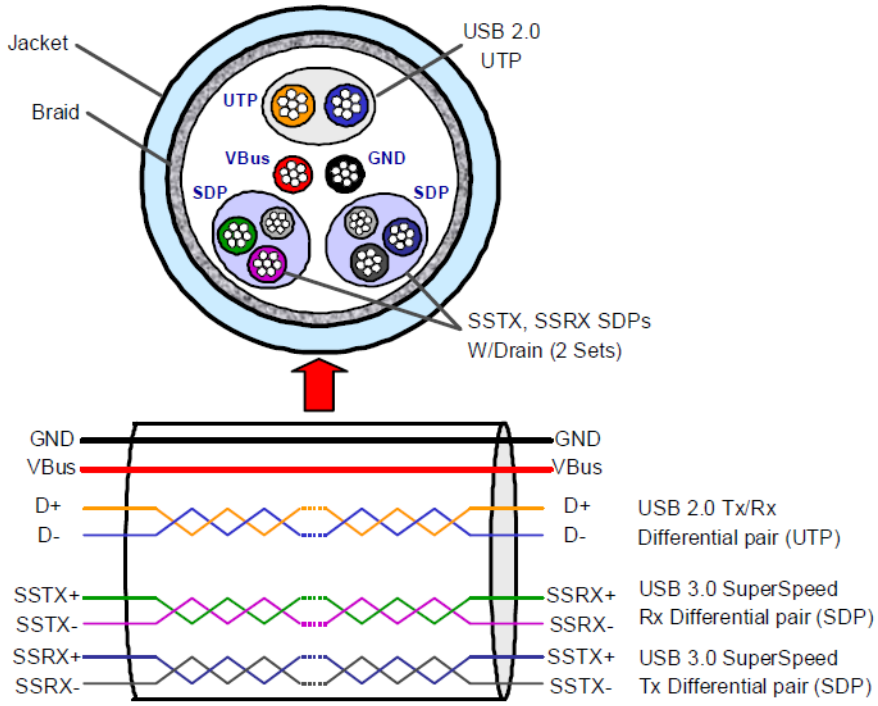
The additional bandwidth provided by USB SuperSpeed transactions can benefit applications like real-time audio and video streaming that require a higher bus bandwidth at regular intervals. Mass storage applications can also benefit from the SuperSpeed bandwidth. FX3 has also been used to implement a high-performance PC-based logic analyzer.

1.1.1 Physical Layer

The physical layer refers to the PHY part of the port and the cable connecting the upstream and downstream ports. USB 3.0 cables have separate shielded differential pairs of lines for transmitting and receiving data. These lines exist along with the USB 2.0 signals. So a USB 3.0 cable contains a total of nine wires including the four wires that are part of the USB 2.0 cable

(see Figure 1-1). The SuperSpeed bus employs a dual-simplex approach that allows simultaneous transmission and reception of packets. In many cases, a SuperSpeed device may be both transmitting and receiving data at the same time. For example, during burst transactions, a device may be receiving data from the host and returning acknowledgements associated with the data it already received.

Figure 1-1. Cross-Section of a USB 3.0 Cable



Coming to the power distribution via the USB 3.0 host, 150 mA is considered as the unit load. USB 3.0 host supplies one unit load of current for unconfigured devices and six unit loads of current for configured devices. The USB 3.0 host detects the device connection based on the receiver end termination, and the transmitter is responsible for detecting the device connection. USB 3.0 uses spread-spectrum clocking on its signaling. If spread-spectrum clocking is enabled, then the energy of the signal is spread over a larger frequency band rather concentrated over a small frequency band at a high level, helping to reduce the EMI emissions. The USB 3.0 physical layer supports low-frequency periodic signaling (LFPS), which is used to manage signal initiation and low-power management on the bus on an idle link to consume less power. Table 1-1 lists the differences between the USB 3.0 and USB 2.0 physical layers.

Table 1-1. Differences Between Physical Layers of USB 3.0 and USB 2.0

Feature	USB 3.0	USB 2.0
Signaling rate	5 Gbps	480 Mbps
Data transfers	Dual simplex	Half duplex
Number of pins in the USB cable	9 (VBUS, Ground, D+, D-, SSRX+, SSRX-, SSTX+, SSTX-, Ground_Drain)	4 (VBUS, Ground, D+, D-)
Data lines in the cable	Shielded differential pair (SDP, twisted, or twinax)	Unshielded twisted pair (UTP)
Current supplied by the host	150 mA for unconfigured devices, 900 mA for configured devices	100 mA for unconfigured devices, 500 mA for configured devices
Device detection by the host	Receiver end termination	Pull-up resistor on D+ or D- lines

1.1.2 Link Layer

The link layer is responsible for maintaining a reliable and robust communication channel between the host and the device. The Link Training and Status State Machine (LTSSM) at the core of the USB 3.0 link layer establishes the link connectivity

and link power management states and transitions. LTSSM consists of 12 states, including four operational link states (U0, U1, U2, U3). The link layer offers these four link power states for better power management:

- U0-Fully powered; link partners are fully powered and ready to send packets
- U1-Standby with fast recovery; link is in a low-power state and is not ready to send packets but can transition back to U0 within microseconds
- U2-Standby with slow recovery; link power saving is greater than U1 and transitioning back to U0 within microseconds to milliseconds
- U3-Suspend; greatest power savings and longest recovery back to U0 (milliseconds)
- U1, U2, and U3 have increasingly longer wakeup times into U0 and thus allow transmitters to go into increasingly deeper sleep.
- Four link initialization and training states (Rx.Detect, Polling, Recovery, Hot Reset)
- Two link test states (Loopback and Compliance mode)
- SS.Inactive (link error state where USB 3.0 is nonoperable)
- SS.Disabled (SuperSpeed bus is disabled and operates as USB 2.0 only)

Link commands are used to maintain the link flow control and to initiate a change in the link power state.

1.1.3 Protocol Layer

The protocol layer manages the communication rules between a host and device. The SuperSpeed protocol layer includes the following improvements to enable better performance, efficiency, and power conservation. The information in this section is provided as background material; the FX3 logic manages protocol details so the application program can deal directly with USB data.

1.1.3.1 Unicast Transactions

SuperSpeed transactions are routed directly from a root port to the target device with the help of a route string in the packet header. Therefore, only links in the direct path between the root port and target device see the traffic, which lets other links in the topology enter or remain in a low-power state.

1.1.3.2 Token/ Data/Handshake Sequences

A USB 2.0 transaction consists of three packets: token, data, and handshake. A transaction is initiated with the token packet and it is always from the host. Data packets deliver the payload data, which can be sourced by the host or device. The handshake packet acknowledges the error-free receipt of data and is sent by the receiver of the data. But with SuperSpeed, to save bandwidth, the token is incorporated into the data packet for OUT transactions; it is replaced by the handshake for IN transactions. So an ACK packet acknowledges the previous data packet sent and requests the next data packet. The following examples clarify the differences between USB 2.0 and USB 3.0 IN and OUT transactions.

1.1.3.2.1 IN Transaction Example

Figure 1-2 on page 22 illustrates the differences between USB 2.0 and SuperSpeed OUT transactions. The example on the left in Figure 1-2 on page 22 shows the sequence of packets required to perform two USB 2.0 IN transactions that require six packets:

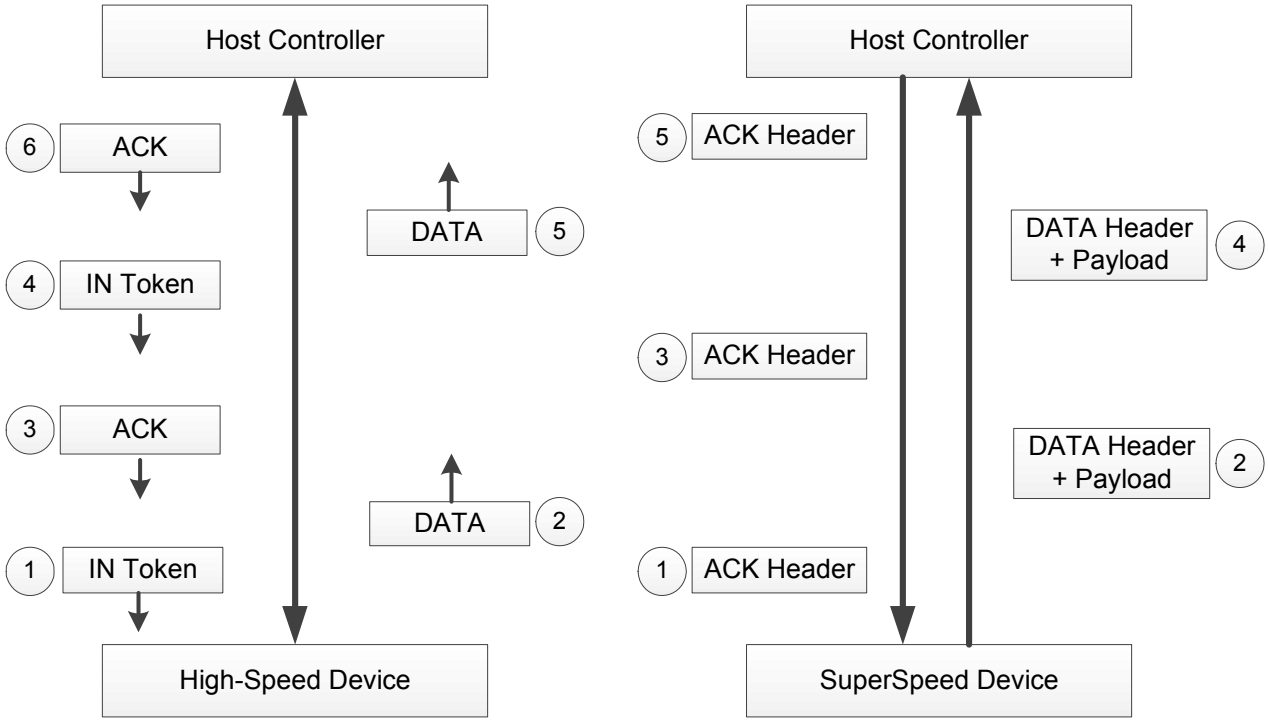
1. Host broadcasts an IN token packet (1) to initiate the transaction.
2. Device returns the requested data packet (2).
3. Host acknowledges receipt of data with an ACK handshake packet (3).
4. Steps 1-3 are repeated.

The example on the right indicates the packet sequence necessary to perform two back-to-back SuperSpeed IN transactions, which require only five packets to be exchanged:

1. SuperSpeed uses an ACK header (1) to initiate an IN transaction.
2. The SuperSpeed device returns the data packet (2).

3. The second ACK header (3) both acknowledges receipt of the data and requests a second transaction.
4. The second data packet (4) is delivered by the device.
5. The final ACK header (5) acknowledges receipt of the data, but does not request additional data.

Figure 1-2. USB 2.0 IN Transaction Versus USB 3.0 IN Transaction



1.1.3.2.2 OUT Transaction Example

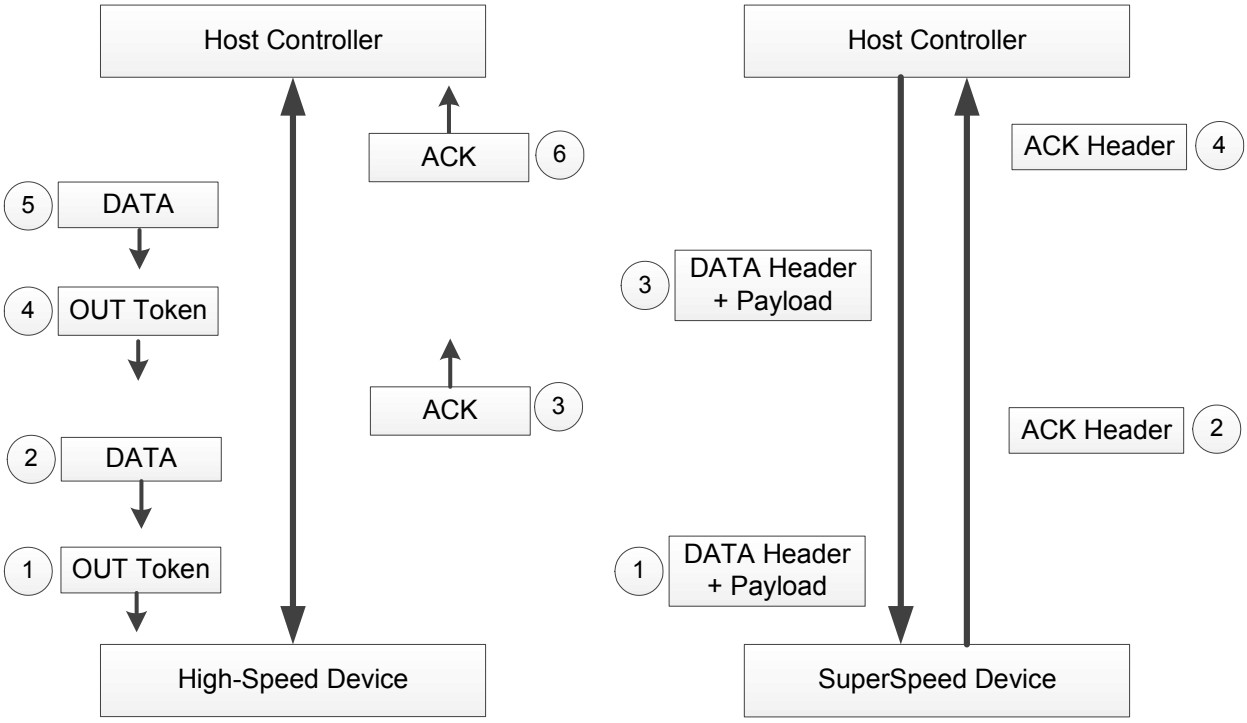
Figure 1-3 illustrates the differences between USB 2.0 and SuperSpeed OUT transactions. The example on the left shows two back-to-back OUT transactions that require six packets:

1. Host broadcasts an OUT token packet (1) to initiate the transaction.
2. Host sends a data packet (2) to the device.
3. Device acknowledges receipt of data with an ACK handshake packet (3).
4. Steps 1-3 are repeated

The right side of Figure 1-3 shows the packet sequence required to perform two back-to-back SuperSpeed OUT transactions, requiring only 4 packets to be exchanged:

1. SuperSpeed USB uses a data header (1) to initiate an OUT transaction and to deliver data to the device.
2. Device acknowledges receipt of data via an ACK packet (2).
3. The second data packet (3) initiates the second transaction and delivers data to the device.
4. Device acknowledges receipt of data via an ACK packet (4), completing the sequence.

Figure 1-3. USB 2.0 OUT Transaction Versus USB 3.0 OUT Transaction

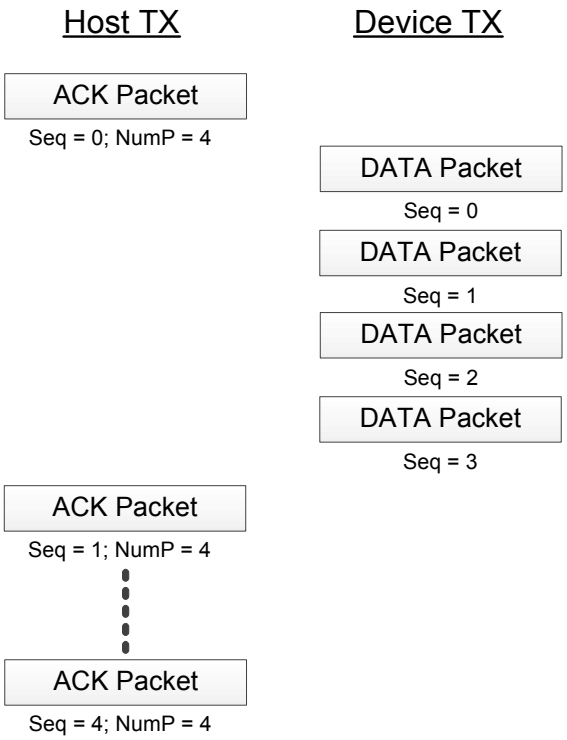


1.1.3.3 Data Bursting

The SuperSpeed end-to-end protocol supports transmitting the data in bursts (multiple data packets) without receiving an acknowledgement to improve latency and performance. The protocol allows efficient bus utilization by concurrently transmitting and receiving over the bus. A transmitter (host or device) can burst multiple packets of data back to back, and the receiver can transmit data acknowledgements without interrupting the burst of data packets. Also, the host may simultaneously schedule multiple OUT bursts to be active at the same time as an IN burst. Devices report their ability to support bursting in their device descriptors. The maximum burst size is 16, and the actual number to be used represents the number of data packets that can be sent without receiving an acknowledgement.

This bursting approach is explained in Figure 1-3 with an IN endpoint that supports a burst size of four. The host initiates the burst transfer and indicates the expected sequence number of the first data packet returned (Seq=0) and the number of packets it wishes to receive (NumP=4). The target device responds with a burst sequence of four data packets without receiving any handshakes. A fifth data packet cannot be returned until data packet zero is acknowledged and the host has indicated a request for another data packet (that is, a second ACK packet with NumP=4). In this burst example, the host continues to request additional data by keeping the NumP value at 4.

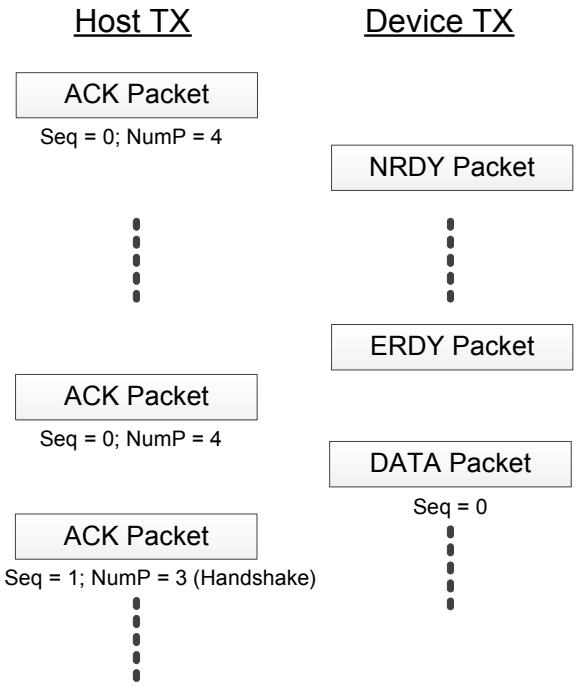
Figure 1-4. Data Bursting in SuperSpeed



1.1.3.4 End-to-End Flow Control

USB 2.0 uses polling and the NAK handshake packet for flow control. For example, USB keyboards must be constantly polled by the host software to check for activity. When an IN token packet is delivered and no keyboard activity has occurred, the keyboard returns a NAK packet. Subsequently, the host software will poll the device again and receive another NAK. This process continues until there is renewed activity. SuperSpeed flow control uses a poll-once approach coupled with an asynchronous ready notification. Consider the IN transaction shown in Figure 1-5. An ACK packet initiates the IN transaction and if a device responds with "NRDY" (Not Ready), then the host stops talking to that device until the device sends the "ERDY" (ready) packet saying that now it is ready to transmit the data. So the host does not need to continue polling. This can significantly reduce SuperSpeed traffic and improve link power management.

Figure 1-5. End-to-End Flow Control in SuperSpeed



1.1.3.5 Streams

USB 3.0 enhanced the bulk transfer capabilities by adding a concept called "streams." This allows a device to accept multiple commands on a pipe from the host and to complete them out of order using the stream IDs.

Table 1-2 lists the differences between the USB 3.0 and USB 2.0 protocol layers:

Table 1-2. Differences Between USB 3.0 and USB 2.0 Protocol Layers

Feature	USB 3.0	USB 2.0
Bus transaction protocol	Host directly routes the packet to the targeted device with the help of route string; exception-isochronous timestamp packet	Host broadcasts the packets to all devices-no route string
	Asynchronous traffic flow	Polled traffic flow
	Simultaneous IN/OUTs	IN or Out
Data transfer types	USB 2.0 types with SuperSpeed constraints; bulk has streams capability	Four types: Control, interrupt, bulk, isochronous
Maximum packet size		
Control	512	64
Interrupt	1024	1024
Bulk	1024	
	Supports streaming functionality	Does not support
Isochronous	1024	1024

1.2 SuperSpeed Power Management

SuperSpeed USB provides a much improved mechanism for entering and exiting low-power states. USB 2.0 implements a feature known as "Suspend" that forces devices to limit current consumption to 2.5 mA. Entry into the low-power state requires a minimum of 3 ms, and exit requires more than 20 ms. SuperSpeed power management provides finer granularity when entering low-power states and reduces entry and exit times. The device can also initiate the low-power link states when it is idle.

Power management features are implemented at all layers:

The physical layer supports the remote wakeup signaling.

The link layer supports low-power link state entry and exit with the help of LTSSM and link commands. It offers four link power states (U0, U1, U2, and U3) for better power management. The following architectural features aid in SuperSpeed link power management:

- The much higher transmission rates mean that SuperSpeed transactions complete very quickly, leaving links in the idle state for a longer period of time.
- The Unicast approach involves only the links in the direct path between the originating root port and target device, leaving other links idle.
- The "poll once and notify" mechanism used in SuperSpeed end-to-end flow control reduces overall link traffic.

Protocol layer supports endpoint busy/ready notifications.

1.2.1 Function Power Management

SuperSpeed power management includes the ability to place a specific function (an interface or a set of interfaces) into a suspended state. This means that a multifunction device can have some functions suspended while others remain fully operational. Functions are placed into suspend under software control. The asynchronous Function Wake notification tells software that a suspended function or device is requesting a remote wakeup.

1.3 FX3/FX3S Features

EZ-USB FX3/FX3S supports the following features.

- Universal Serial Bus (USB) integration
  - USB 3.0 and USB 2.0 peripherals compliant with USB 3.0 specification revision 1.0
  - 5-Gbps USB 3.0 PHY
  - HS-OTG host and peripheral compliant with OTG Supplement version 2.0
  - 32 physical endpoints
  - Support for Battery Charging Spec 1.1 and accessory charger adapter (ACA) detection
- General Programmable Interface (GPIF II)*
  - Programmable GPIF II, enabling connectivity to a wide range of external devices
  - Interface frequency of up to 100 MHz
  - 8-, 16-, and 32-bit data bus
  - As many as 16 configurable control signals
- 32-bit CPU
  - ARM926EJ core with 200-MHz operation
  - 512 KB or 256 KB embedded SRAM
- Additional connectivity to the following peripherals:
  - I2C master controller up to 1 MHz
  - I2S master (transmitter only) at sampling frequencies of 32 kHz, 44.1 kHz, and 48 kHz
  - UART support of up to 4 Mbps
  - SPI master at 33 MHz
- Selectable clock input frequencies
  - 19.2, 26, 38.4, and 52 MHz
  - 19.2-MHz crystal input support
- Ultra low-power in core power-down mode
  - Less than 60 µA with VBATT on and 20 µA with VBATT off

- Independent power domains for core and interfaces
- Core and USB operation at 1.2 V
- GPIF II, I2S, UART, and SPI operation at 1.8 to 3.3 V
- I2C and JTAG operation at 1.2 to 3.3 V

- Storage interfaces*
  - Mass storage support
  - SD 3.0 (SDXC) UHS-1
  - eMMC 4.41
  - Two ports that can support memory card sizes up to 2 TB
  - Built-in RAID (implemented in firmware) with support for RAID0 and RAID1
- System I/O expansion with two secure digital I/O (SDIO 3.0) ports
- Support for USB-attached storage (UAS), mass-storage class (MSC), human interface device (HID), full, and Turbo-MTP™

*Storage interfaces are available only with FX3S and its GPIF data bus width is limited to 16 bits.

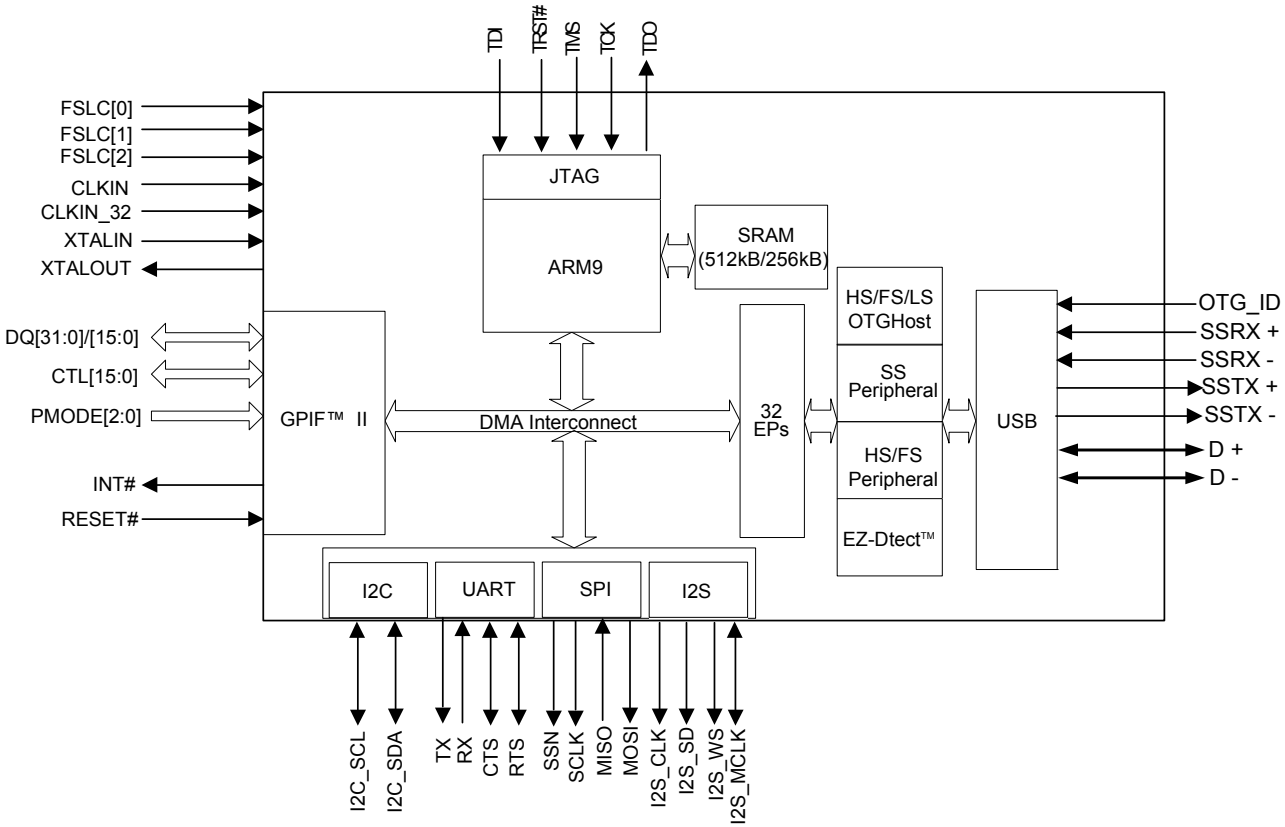
Table 1-3. Feature Differences Between FX3 and FX3S

Feature	EZ-USB FX3	EZ-USB FX3S
GPIF	8/16/32-bit	8/16-bit
Storage ports	No	1 or 2 ports (SD3.0, eMMC4.41, SDIO3.0)
USB 3.0, USB 2.0 Device	Yes	Yes
HS-OTG	Yes	Yes
CPU	ARM9, 200 MHz	ARM9, 200 MHz
Embedded SRAM	256 KB/512 KB	256 KB/512 KB
Serial Interfaces*	I2C, SPI, I2S, UART	I2C, SPI, I2S, UART
Boot Options	I2C, SPI, USB, GPIF based	All FX3 boot options + eMMC based boot options
Package	121-ball BGA, 10 x 10 mm, 0.8 mm pitch. 131-ball, WLCSP, 4.7 x 5.1 mm, 0.4 mm pitch	121-ball BGA, 10 x 10 mm, 0.8 mm pitch

*All serial interfaces might not be available under all configuration options. Refer to the pin description section in the datasheet for details.

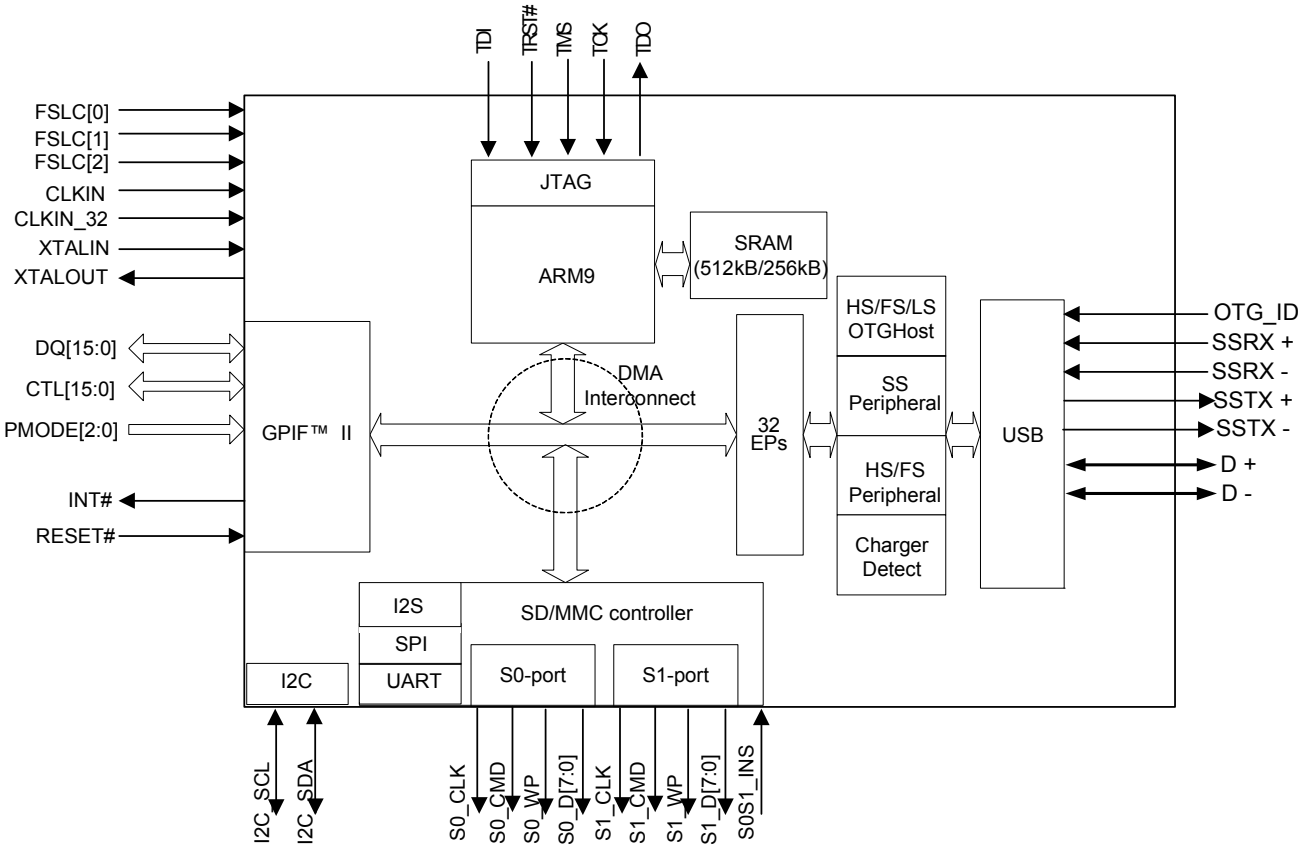
1.3.1 FX3 Block Diagram

Figure 1-6. FX3 Block Diagram



1.3.2 FX3S Block Diagram

Figure 1-7. FX3S Block Diagram



Note: For more details on pin mapping and their descriptions, refer to FX3 and FX3S datasheets.

1.4 Functional Overview

1.4.1 CPU

FX3 has an on-chip 32-bit, 200-MHz ARM926EJ-S core CPU. The core has direct access to 16 KB of instruction tightly coupled memory (TCM) and 8 KB of data TCM. The ARM926EJ-S processor also has associated instruction cache (I-cache) and data cache (D-cache) memories. Both the instruction and data caches are 8 KB.

FX3 integrates 512 KB or 256 KB (depending on the part number) of embedded SRAM for storing code and data. The ARM926EJ-S core provides a JTAG interface for firmware debugging.

FX3 interrupts are managed through the standard ARM PrimeCell Vectored Interrupt Controller (PL192) block. This interrupt controller provides vectored interrupt support with configurable priorities for all interrupt sources.

Examples of the FX3 firmware are available with the Cypress EZ-USB FX3 Development Kit.

For more information about the CPU subsystem, refer to the FX3 CPU Subsystem chapter on page 35.

For more information about the memory map, refer to the Memory and System Interconnect chapter on page 45.

1.4.2 DMA

FX3 enables efficient and flexible DMA transfers between the various peripherals (such as USB, GPIF II, I2S, SPI, and UART), requiring firmware only to configure data accesses between peripherals. The data transfers are managed by distributed DMA controllers within each peripheral. For more information about the FX3 DMA interconnect, refer to the FX3 DMA Subsystem chapter on page 61.

1.4.3 USB Interface

The FX3 USB interface supports the following:

- USB SuperSpeed and High-Speed peripheral functionality is compliant with USB 3.0 specification revision 1.0 and is backward compatible with the USB 2.0 specification.
- As a USB peripheral, FX3 supports SuperSpeed, High-Speed, and Full-Speed transfers. As a host, FX3 supports High-Speed, Full-Speed, and Low-Speed transfers.
- Complies with OTG Supplement revision 2.0, supporting dual-role operation. As an OTG host, FX3 supports
- USB classes such as Mass Storage (MSC) and Human Interface Device (HID).
- Carkit Pass-through UART functionality on USB D+/D- lines based on the CEA-936A specification
- 16 IN and 16 OUT endpoints
- CONTROL, BULK, INTERRUPT, and ISOCHRONOUS endpoints
- USB 3.0 BULK streams feature.
- USB charger and accessory detection (EZ-DTECT).

The charger detection mechanism complies with the USB Battery Charging Specification revision 1.1. In addition to supporting this version of the specification, FX3 provides hardware support to detect resistance values on the ID pin.

For more information about the USB block, refer to the Universal Serial Bus (USB) chapter on page 81.

1.4.4 GPIF II

The GPIF II is a programmable state machine that enables a flexible interface that may function either as a master or slave to industry-standard or proprietary interfaces. The high-performance GPIF II interface provides functionality similar to, but more advanced than, the FX2LP™ GPIF and Slave FIFO interfaces. Both parallel and serial interfaces may be implemented with GPIF II.

The GPIF II implements an interface by creating a GPIF II state machine. GPIF II state transitions are based on input signals, and the control output signals are driven as a result of the GPIF II state transitions. Some popular interfaces that can be implemented with GPIF II are the Slave FIFO interface, SRAM, Address/Data bus interfaces, and Address Multiplexed (ADMux) interfaces.

For more information about the synchronous Slave FIFO interface, refer to the application note AN65974 - Designing with the EZ-USB FX3 Slave FIFO Interface.

The key features of GPIF II are:

- Functions as a master or slave
- Provides 256 firmware programmable states
- Supports 8-bit, 16-bit, 24-bit, and 32-bit parallel data buses
- Enables interface frequencies up to 100 MHz
- Supports 14 configurable control pins when a 32-bit data bus is used. All control pins can be either input or outputor bidi-rectional.
- Supports 16 configurable control pins when a 16- or 8-bit data bus is used. All control pins can be either input or output or bidirectional.

Cypress's GPIF II Designer tool enables GPIF II designs to be developed quickly and includes examples of common interfaces. For more information about the GPIF II block, refer to the General Programmable Interface II (GPIF II) chapter on page 125.

1.4.5 UART Interface

FX3 UART supports full-duplex communication and consists of the TX, RX, CTS, and RTS signals. The UART is capable of generating a range of baud rates, from 300 bps to 4608 Kbps, selectable by the firmware. If flow control is enabled, then the FX3 UART transmits data when the CTS input is asserted. In addition, the FX3 UART asserts the RTS output signal when it is ready to receive data.

1.4.6 I2C Interface

The FX3 I2C interface is compatible with the I2C Bus Specification revision 3. This I2C interface is capable of operating only as an I2C master; therefore, it may be used to communicate with other I2C slave devices. For example, FX3 may boot from an EEPROM connected to the I2C interface, as a selectable boot option.

The FX3 I2C master controller also supports multimaster functionality. The FX3 I2C controller is powered by a dedicated power pin, VIO5, which also powers the JTAG interface. This gives the I2C interface the flexibility to operate at a different voltage than other serial interfaces.

The I2C controller supports bus frequencies of 100 kHz, 400 kHz, and 1 MHz. When VIO5 is 1.2 V, the maximum operating frequency is 100 kHz. When VIO5 is 1.8 V, 2.5 V, or 3.3 V, the operating frequencies supported are 400 kHz and 1 MHz. The I2C controller supports the clock-stretching feature to enable slower devices to exercise flow control. The I2C interface's SCL and SDA signals require external pull-up resistors, which must be connected to VIO5.

1.4.7 I2S Interface

FX3 has an I2S port to support external audio codec devices. FX3 functions as an I2S master as a transmitter only.

The I2S interface consists of four signals: clock (I2S_CLK), serial data (I2S_SD), word select (I2S_WS), and master system clock (I2S_MCLK). FX3 can generate the system clock as an output on I2S_MCLK or accept an external system clock input on I2S_MCLK.

The sampling frequencies supported by the I2S interface are 32 kHz, 44.1 kHz, and 48 kHz.

1.4.8 SPI Interface

FX3 provides an SPI master interface. The maximum operation frequency is 33 MHz. The SPI controller supports four modes of SPI communication. This controller is a single master controller with a single automated Slave Select, Negative-true (SSN) control. It supports transaction sizes ranging from 4 bits to 32 bits.

For more information about the UART, I2C, I2S, and SPI interfaces, refer to the Low Performance Peripherals (LPP) chapter on page 173.

1.4.9 JTAG Interface

The FX3 JTAG interface is a standard five-pin interface to connect to a JTAG debugger to debug the firmware through the CPU core's on-chip-debug circuitry. Industry-standard debugging tools for the ARM926EJ-S core can be used for FX3 application development.

1.4.10 Storage Interface

FX3S has two independent storage ports (S0-Port and S1-Port). Both storage ports support the following specifications:

- MMC system specification, MMCA Technical Committee, version 4.41
- SD specification, version 3.0
- SDIO host controller compliant with SDIO Specification version 3.00

Both storage ports support the following features:

1.4.10.1 SD/MMC Clock Stop

FX3S supports the stop clock feature, which can save power if the internal buffer is full when receiving data from the SD/MMC/SDIO.

1.4.10.2 SD_CLK Output Clock Stop

During the data transfer, the SD_CLK clock can be enabled (on) or disabled (stopped) at any time by the internal flow control mechanism.

SD_CLK output frequency is dynamically configurable using a clock divider from a system clock. The clock choice for the divisor is user-configurable through a register. For example, the following frequencies may be configured:

- 400 kHz - For the SD/MMC card initialization
- 20 MHz - For a card with 0- to 20-MHz frequency
- 24 MHz - For a card with 0- to 26-MHz frequency
- 48 MHz - For a card with 0- to 52-MHz frequency (48-MHz frequency on SD_CLK is supported when the clock input to FX3S is 19.2 MHz or 38.4 MHz)
- 52 MHz - For a card with 0- to 52-MHz frequency (52-MHz frequency on SD_CLK is supported when the clock input to FX3S is 26 MHz or 52 MHz)
- 100 MHz - For a card with 0- to 100-MHz frequency If the DDR mode is selected, data is clocked on both the rising and falling edge of the SD clock. DDR clocks run up to 52 MHz.

1.4.10.3 Card Insertion and Removal Detection

FX3S supports the following two card insertion and removal detection mechanisms:

- Use of SD_D[3] data
- Use of the S0S1_INS pin

1.4.10.4 Write Protection (WP)

The S0_WP/S1_WP (SD Write Protection) on S-Port is used to connect to the WP microswitch of the SD/MMC card connector. This pin internally connects to a CPU-accessible GPIO for firmware to detect the SD card write protection.

1.4.10.5 SDIO Interrupt

The SDIO interrupt functionality is supported as specified in the SDIO specification version 2.00 (January 30, 2007).

1.4.10.6 SDIO Read-Wait Feature

FX3S supports the optional read-wait and suspend-resume features as defined in the SDIO specification version 2.00 (January 30, 2007).

1.4.10.7 Boot Options

FX3 integrates 32 KB of ROM, which contains a bootloader, allowing FX3 to load boot images from various sources. The boot mode is selected by the configuration of the PMODE pins as shown in Table 1-3 on page 27

The FX3 boot options are:

- Boot from USB
- Boot from I2C
- Boot from SPI
- Boot from GPIF II Async ADMux mode
- Boot from GPIF II Sync ADMux mode

- Boot from GPIF II Async SRAM mode

Please refer to the application note AN76405 - EZ-USB FX3 Boot Options.

In addition to these FX3 boot options, FX3S supports the following:

- Boot from eMMC (Storage port)
- Boot from PMMC (Processor port)

Table 1-4. Boot Mode Selection Based on PMODE Pins

PMODE[2:0] Pins			Boot Option
PMODE[2]	**PMODE[1]**	**PMODE[0]**	
Z	0	0	Sync ADMUX (16-bit)
Z	0	1	Async ADMUX (16-bit)
Z	0	Z	Async SRAM (16-bit)
Z	1	1	USB Boot
1	Z	Z	I2C
Z	1	Z	I2C; on failure, USB Boot is enabled
0	Z	1	SPI; on failure, USB Boot is enabled
FX3S Specific Boot Options			
Z	1	0	PMMC Legacy
0	0	0	S0-port (eMMC); On Failure, USB Boot is enabled
1	0	0	S0-port (eMMC)

Z = Pin is floating; left unconnected.

1.4.11 Clocking

FX3 allows either connecting a crystal between the XTALIN and XTALOUT pins or connecting an external clock to the CLKIN pin. The XTALIN, XTALOUT, CLKIN, and CLKIN_32 pins can be left unconnected if they are not used.

The crystal frequency supported is 19.2 MHz, while the external clock frequencies supported are 19.2, 26, 38.4, and 52 MHz.

FX3 has an on-chip oscillator circuit that uses an external 19.2-MHz (±100 ppm) crystal (when the crystal option is used). Please refer to the application note AN70707 - EZ-USB FX3/FX3S Hardware Design Guidelines and Schematic Checklist for guidelines on crystal selection. An appropriate load capacitance is required with a crystal. The FSLC[2:0] pins must be configured appropriately to select the crystal- or clock-frequency option. The configuration options are listed in Table 1-4.

Table 1-5. Crystal and Clock Frequency Selection

FSLC[2]	FSLC[1]	FSLC[0]	Crystal/Clock Frequency
0	0	0	19.2-MHz Crystal
1	0	0	19.2-MHz Input CLK
1	0	1	26-MHz Input CLK
1	1	0	38.4-MHz Input CLK
1	1	1	52-MHz input CLK

Clock inputs to FX3 must meet the phase noise and jitter requirements specified in the EZ-USB FX3 datasheet.

2. FX3 CPU Subsystem

The EZ-USB FX3 device has an embedded 32-bit ARM926EJ-S core that delivers a processing capability up to 220 MIPS. This ARM core is coupled with instruction and data caches, Tightly Coupled Memories (TCM), and a PL192 vectored interrupt controller (VIC). FX3 also implements the standard ARM JTAG Test Access Point (TAP), which allows you to use standard JTAG debuggers to debug firmware applications.

The ARM926EJ-S processor is targeted at multitasking applications and can support high-performance and low-power requirements.

Interrupts in the FX3 device are managed through the standard ARM PL192 VIC block.

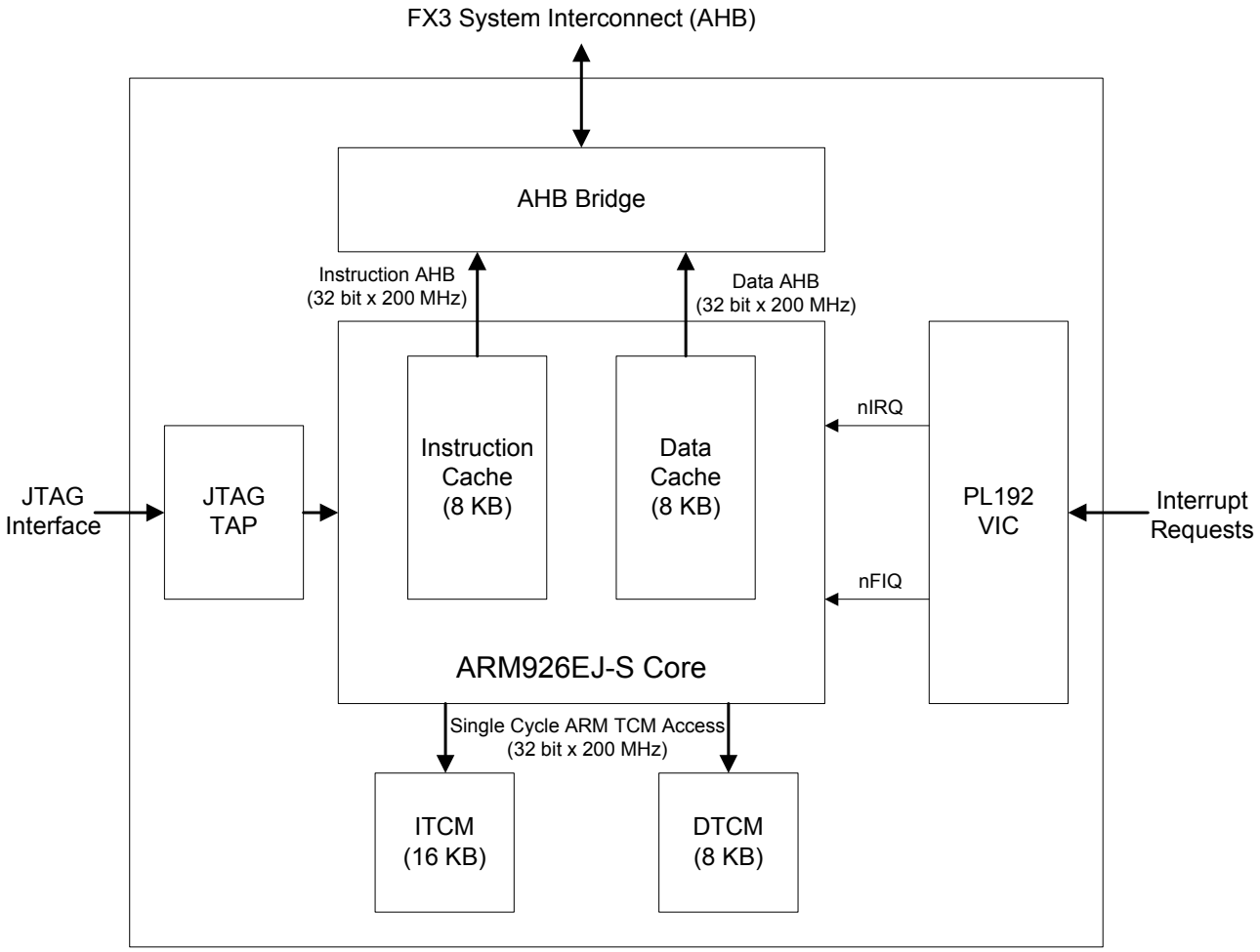
2.1 Features

The ARM9 core in the FX3 device supports the following features:

- Operation at frequencies up to 200 MHz
- Support for both 32-bit ARM and 16-bit thumb instructions
- Integrated data and instruction caches of 8 KB each
- Dedicated instruction and data TCMs for guaranteed low-latency memory access
- VIC capable of managing 32 internal interrupt sources with programmable interrupt priorities
- Standard ARM JTAG interface for debugging
- Clock frequency control for power saving
- System RAM on FX3 device serves as main storage for code and data

2.2 Block Diagram

Figure 2-1. CPU Subsystem in FX3 Device



2.3 Functional Overview

Figure 2-1 shows the ARM9 core and the associated blocks in the FX3 device. The CPU is associated with TCM blocks that enable zero-latency accesses to performance-critical instructions and data, and it provides separate instruction and data caches for other memory accesses. The PL192 VIC manages interrupts raised by the FX3 hardware blocks.

2.3.1 ARM926EJ-S CPU

The FX3 device has an embedded 32-bit ARM926EJ-S CPU core. This makes the device capable of implementing multitasking applications where high performance and low power consumption are important.

This ARM9 core supports both 32-bit ARM instructions and 16-bit thumb instructions. The processor has separate advanced high-performance bus (AHB) interfaces for internal instruction and data accesses. It also has separate instruction and data TCM interfaces.

Note: The 32-bit ARM instruction set is commonly used in the FX3 SDK from Cypress, as this makes it more convenient to address all of the available device memory.

The following subsections provide information for programming the FX3 device. The FX3 SDK takes care of these aspects and initializes the ARM core and memory blocks on the FX3 device. The following detail is only required if you are making any modifications to the base SDK source code. For more details on the ARM CPU architecture, instruction set, and so on, refer to the *ARM926EJ-S Technical Reference Manual*.

2.3.1.1 Processor Modes

The ARM architecture supports seven operating modes, as shown in Table 2-1.

Table 2-1. ARM9 CPU Operating Modes

Processor Mode	Abbreviation	Description
User	Usr	Normal program execution mode
FIQ	Fiq	Fast interrupt mode; used for a performance-critical interrupt
IRQ	Irq	General-purpose interrupt handling mode
Supervisor	Svc	Protected mode used by operating systems
Abort	Abt	Instruction/data abort handling mode; used for virtual memory implementation
Undefined	Und	Undefined instruction mode; used to support software emulation of hardware coprocessors; generally not useful on FX3 device.
System	Sys	Runs privileged operating system tasks

All of the modes except the user mode are privileged modes that have full access to all system resources and can freely change modes.

The FIQ, IRQ, supervisor, abort, and undefined modes are entered when specific exceptions occur. These modes have a separate register set, so that the user mode registers are not corrupted when the exception occurs.

The system mode uses the same set of registers as the user mode and is used to execute OS tasks that do not need a separate register set.

Run-time stack regions need to be set up for all these modes at system startup. The following code snippet shows how this can be done using ARM assembly language code. This code is provided by the Cypress FX3 firmware library, and rarely needs to be modified.

```
#define FX3_STACK_BASE          0x10000000  /* D-TCM area ( 8KB ) */
#define FX3_SVC_STACK_SIZE      0x1000      /* SVC stack size */
#define FX3_FIQ_STACK_SIZE      0x0200      /* FIQ stack size */
#define FX3_IRQ_STACK_SIZE      0x0400      /* IRQ stack size */
#define FX3_SYS_STACK_SIZE      0x0800      /* SYS stack size */
#define FX3_ABT_STACK_SIZE      0x0100      /* ABT stack size */
#define FX3_UND_STACK_SIZE      0x0100      /* UND stack size */

#define ARM_FIQ_MODE            0xD1        /* Disable Interrupts + FIQ mode */
#define ARM_IRQ_MODE            0xD2        /* Disable Interrupts + IRQ mode */
#define ARM_SVC_MODE            0xD3        /* Disable Interrupts + SVC mode */
#define ARM_ABT_MODE            0xD7        /* Disable Interrupts + ABT mode */
#define ARM_UND_MODE            0xDB        /* Disable Interrupts + UND mode */
#define ARM_SYS_MODE            0xDF        /* Disable Interrupts + SYS mode */

/* Stack pointers are placed at the top address as the stack grows downwards */
SetupStackPtrs:
    ldr r1, =FX3_STACK_BASE         /* Load the stack base address */
    sub r1, r1, #8                  /* Prevent overflow */

    ldr r2, =FX3_SYS_STACK_SIZE     /* Pickup stack size */
    mov r3, #ARM_SYS_MODE           /* Build SYS mode CPSR */
    msr CPSR_cxsf, r3               /* Enter SYS mode */
    add r1, r1, r2                  /* Calculate start of SYS stack */
```

```
bic r1, r1, #7                  /* Ensure 8-byte alignment */
mov sp, r1                      /* Setup SYS stack pointer */
mov r10, #0                     /* Clear SYS mode sl */
mov r11, #0                     /* Clear SYS fp */

ldr r2, =FX3_ABT_STACK_SIZE     /* Pickup stack size */
mov r3, #ARM_ABT_MODE           /* Build ABT mode CPSR */
msr CPSR_cxsf, r3               /* Enter ABT mode */
add r1, r1, r2                  /* Calculate start of ABT stack */
bic r1, r1, #7                  /* Ensure 8-byte alignment */
mov sp, r1                      /* Setup ABT stack pointer */
mov r10, #0                     /* Clear ABT mode sl */
mov r11, #0                     /* Clear ABT fp */

ldr r2, =FX3_UND_STACK_SIZE     /* Pickup stack size */
mov r3, #ARM_UND_MODE           /* Build UND mode CPSR */
msr CPSR_cxsf, r3               /* Enter UND mode */
add r1, r1, r2                  /* Calculate start of UND stack */
bic r1, r1, #7                  /* Ensure 8-byte alignment */
mov sp, r1                      /* Setup UND stack pointer */
mov r10, #0                     /* Clear UND mode sl */
mov r11, #0                     /* Clear UND fp */

ldr r2, =FX3_FIQ_STACK_SIZE     /* Pickup stack size */
mov r0, #ARM_FIQ_MODE           /* Build FIQ mode CPSR */
msr CPSR_c, r0                  /* Enter FIQ mode */
add r1, r1, r2                  /* Calculate start of FIQ stack */
bic r1, r1, #7                  /* Ensure 8-byte alignment */
mov sp, r1                      /* Setup FIQ stack pointer */
mov sl, #0                      /* Clear sl */
mov fp, #0                      /* Clear fp */

ldr r2, =FX3_IRQ_STACK_SIZE     /* Pickup IRQ stack size */
mov r0, #ARM_IRQ_MODE           /* Build IRQ mode CPSR */
msr CPSR_c, r0                  /* Enter IRQ mode */
add r1, r1, r2                  /* Calculate start of IRQ stack */
bic r1, r1, #7                  /* Ensure 8-byte alignment */
mov sp, r1                      /* Setup IRQ stack pointer */

ldr r2, =FX3_SVC_STACK_SIZE     /* Pickup System stack size */
mov r0, #ARM_SVC_MODE           /* Build SVC mode CPSR */
msr CPSR_c, r0                  /* Enter SVC mode */
add r1, r1, r2                  /* Calculate start of SVC stack */
bic r1, r1, #7                  /* Ensure 8-byte alignment */
mov sp, r1                      /* Setup SVC stack pointer */
```

Hint: The 8-KB D-TCM region can be used for the run-time stack regions if there is no other performance-critical data that needs to be placed there.

2.3.1.2 Processor Registers

Table 2-2 shows the registers provided by the ARM9 processor core. The CPSR register is used to switch between various processor modes.

Table 2-2. ARM9 Processor Registers

Register	Description	Attributes
R0-R7	General-purpose registers	These are not banked registers, which means that the same physical register is used in all processor modes.
R8-R12	General-purpose registers	Two copies of these registers exist. One copy is used only in FIQ mode, and the other copy is used in all other processor modes.
R13	Stack pointer (SP)	Six copies of this register exist. The user and system modes share one register, and all other modes have their own SP register.
R14	Link register (LR), used to hold return address when executing branch instructions	Six copies of this register exist. The user and system modes share one register, and all other modes have their own LR register.
R15	Used to read/write the program counter	Single register that is used across all processor modes.
CPSR	Current program status register, provides status flags, global interrupt enable control, and so on	The CPSR register reflects the current program status in each of the processor modes.
SPSR	Saved program status register, provides saved program status for each exception mode	Separate copies of this register exist for each of the FIQ, IRQ, supervisor, abort, and undefined modes.

2.3.1.3 Exception Vectors

The exception vectors that serve as the entry point for each of the exception modes are stored in a table in the main system memory. These vectors can be placed at one of two addresses: 0x00000000 or 0xFFFF0000. The selection of the exception vector table location is based on the ARM standard system control coprocessor (CP15) configuration.

As shown in Table 2-3, the normal exception vectors are located in the address range from 0x0 onward, which falls in the instruction TCM (ITCM) region. The high exception vectors are located in the address range from 0xFFFF0000, which is part of the BootROM on the FX3 device. As the high-exception vectors are hard-wired to vector to the bootloader on the FX3 device, the normal (user-definable) exception vectors are used when running user applications.

Table 2-3. ARM Exception Vector Locations

Exception Type	Processor Mode	Exception Vector 1 (normal)	Exception Vector 2 (high)
Reset	Supervisor	0x00000000	0xFFFF0000
Undefined instruction	Undefined	0x00000004	0xFFFF0004
Software interrupt (SWI)	Supervisor	0x00000008	0xFFFF0008
Prefetch abort	Abort	0x0000000C	0xFFFF000C
Data abort	Abort	0x00000010	0xFFFF0010
IRQ (interrupt)	IRQ	0x00000018	0xFFFF0018
FIQ (interrupt)	FIQ	0x0000001C	0xFFFF001C

The following code snippet shows the procedure to move the exception vectors to address 0x0. This step is done by the FX3 firmware library as part of device initialization.

```
MRC   p15, 0, r1, c1, c0, 0/* Read the CP15 register value */
MOV   r2, #0xFFFFDFFF
AND   r1, r1, r2/* Mask off the vector location bit. */
MCR   p15, 0, r1, c1, c0, 0/* Write back to CP15 register. */
```

2.3.1.4 MMU

The MMU in the ARM926EJ-S processor is an ARM architecture v5 implementation, which supports the virtual memory features required by standard embedded operating systems. The MMU uses a set of two-level page tables located in the main memory to control the address translation, permission checks, and so on.

The data cache on the ARM core can be enabled only if the MMU is enabled. However, most FX3 designs do not use any secondary storage and do not need a virtual memory system. FX3 provides a fixed set of page tables that maps each physical address to the equivalent virtual address.

2.3.1.5 Cache Memories

The ARM926EJ-S processor has associated instruction and data cache memories. The RAM on the FX3 device holds DMA data buffers in addition to code and data. The DMA driver and APIs in the FX3 library ensure cache coherency by using the cache clean and invalidate operations.

The instruction and data caches on FX3 are 8 KB. The caches are four-way set associative with eight word (32-byte) cache lines. The data cache implements two dirty bits per cache line and is typically configured for write-back operations.

The ARM926EJ-S processor supports the following operations using the CP15 coprocessor interface:

- Invalidating the entire D-cache or I-cache
- Invalidating (flushing) regions of the D-cache or I-cache
- Cleaning the entire D-cache
- Cleaning regions of the D-cache
- Locking specified memory regions into the D-cache or I-cache

The CPU also provides a write buffer that is used for writes to regions that are not cacheable or bufferable, and for write-through operations. It is also used for write misses during write-back operations. A separate write buffer is provided as part of the D-cache for holding write-back data during cache line eviction or clean operations. Instructions are provided to drain both write buffers, and the CPU ensures data coherency when read operations are addressed to data that is sitting in the write buffer.

The following code snippet shows the procedure to enable the caches and the MMU on the FX3 device. Please refer to the FX3 Memories chapter for more information on cache operations.

```
MRC p15, 0, r1, c1, c0, 0/* Read CP15 register value */
ORR r1, r1, #0x1000/* Update I-Cache enable bit. */
BIC r1, r1, #0x4000/* Select random replacement. */
ORR r1, r1, #0x05/* Enable MMU and D-Cache. */
MCR p15, 0, r1, c1, c0, 0/* Write modified value back */
```

2.3.1.6 Tightly Coupled Memories

Some operations, such as interrupt handlers, may not be able to tolerate the added latency created by a cache miss. The ARM9 CPU provides a zero wait state TCM interface to facilitate quick access to such instructions and data. Firmware applications can locate performance-critical code and data sections in the TCM regions using the appropriate linker settings. The FX3 SDK provides a linker script that sets up the recommended memory map for FX3 applications.

As in memory and cache access, separate paths are used for instruction and data access from the TCMs. FX3 implements 16 KB of instruction TCM (ITCM) and 8 KB of data TCM (DTCM). The ITCM area can also be accessed by the data side of the ARM core. This is required to facilitate loading the code into the ITCM region.

The ITCM region on FX3 is located in the address range 0x0000-0x3FFF, and the DTCM region is located in the address range 0x10000000-0x10001FFF. The ITCM region is typically used to store the ARM exception vectors and the interrupt service routine (ISR) code. The DTCM region is typically used to store performance-critical data and the run-time stacks for various processor modes.

The TCMs must be configured as non-cacheable memories, and any instruction or data movement between the TCM and the main memory must be performed by the CPU. The TCMs are disabled when the device is reset, and they need to be enabled by the firmware. This is done by the FX3 library as part of device initialization.

```
MOV r1, #0x15  /* ITCM address is 0x0 and size is 16 KB. */
MCR   p15, 0, r1, c9, c1, 1  /* Initialize the ITCM */
MOV r1, 0x10000011  /* DTCM address is 0x10000000 and size is 8 KB. */
MCR   p15, 0, r1, c9, c1, 0  /* Initialize the DTCM */
```

2.3.1.7 JTAG Interface

FX3 makes the standard ARM JTAG TAP available for users to connect a JTAG debugger. Only the 5-pin JTAG mode of debugging is supported, and 2-pin serial wire debug (SWD) mode is not supported by the FX3 device. The JTAG pins are directly connected to the ARM CPU, and they do not support boundary scan.

The JTAG interface allows the use of industry-standard ARM debug probes to debug the firmware running on FX3. No Cypress custom software tools are required to enable the debugging.

The JTAG instruction register (IR) length for the ARM926EJ-S device is 4 bits. If multiple devices are connected on the JTAG chain, the offset and length for the FX3 device have to be set correctly to achieve JTAG connection. Refer to Section 5.10.3 of the Segger J-Link User Guide for instructions on how to do this when using the J-Link debugger.

2.3.1.8 Vectored Interrupt Controller

The FX3 device provides a number of interrupt notifications for the ARM CPU to handle. Interrupts in the FX3 system are managed through the standard ARM PrimeCell Vectored Interrupt Controller (PL192) block. This interrupt controller provides vectored interrupt support with configurable priorities for all interrupt sources.

The PL192 controller supports up to 32 interrupt sources and generates the nIRQ and nFIQ signals to the ARM CPU based on the configuration. The controller is connected to the CPU on the AHB bus and allows you to perform the interrupt configuration through a set of memory mapped registers.

Table 2-4 shows the various interrupt sources on the FX3 device, along with their vector numbers.

Table 2-4. FX3 Interrupt Sources

Vector Number	Interrupt Source	Description	Comments
0	GCTL_CORE	Interrupt raised on FX3 waking up from suspend or standby low-power modes.	
1	SW_INTR	Software interrupt	This is a custom implementation of the software interrupt scheme.
2	UNUSED		Do not enable.
3	UNUSED		Do not enable.
4	WATCHDOG_TIMER	Watchdog timer interrupt; the watchdog timer functions on the basis of a 32-kHz clock signal with a user-configured period	This is commonly used for OS scheduling on the FX3 device.
5	UNUSED		Do not enable.
6	GPIF_DMA	DMA socket interrupt from the GPIF block	
7	GPIF_CORE	General-purpose GPIF interrupts; indicates conditions such as state machine interrupt, GPIF errors, mailbox register access, and so on	
8	USB_DMA	DMA socket interrupt from the USB block	Applies to both USB device and host mode operation.
9	USB_CORE	General-purpose USB interrupts; indicates various conditions triggered during device or host operation of the USB block	
10	UNUSED		Do not enable.
11	STORAGE_DMA	DMA socket interrupt from the storage (SD/MMC) interface	Applies only to the FX3S™ devices.
12	STORAGE0_CORE	General-purpose interrupt from storage interface 0	Applies only to the FX3S™ devices.
13	STORAGE1_CORE	General purpose interrupt from storage interface 1.	Applies only to the FX3S™ devices.
14	UNUSED		Do not enable.
15	I2C_CORE	General-purpose I2C interrupt; indicates conditions such as transfer completion, error detect, and so on	
16	I2S_CORE	General-purpose I2S interrupt	
17	SPI_CORE	General-purpose SPI interrupt	
18	UART_CORE	General-purpose UART interrupt	
19	GPIO_CORE	General-purpose GPIO interrupt; common for both simple and complex GPIOs	
20	PERIPH_DMA	DMA socket interrupt from any serial peripheral block (I2C, I2S, SPI, or UART).	
21	GCTL_POWER	Power detect interrupt; indicates voltage changes on any power inputs that can be dynamically changed during device operation.	This is commonly used for VBus voltage detection.
22-31	UNUSED		Do not enable.

You can configure any of the 32 interrupt sources as fast interrupt request (FIQ), which takes the highest priority among the interrupts. The rest of the interrupts are prioritized by user code through the VIC_VECT_PRIORITY registers. If two or more interrupts are programmed with the same priority value, the source with the lower vector number assumes the higher priority.

The PL192 controller allows each of the interrupt sources to be independently masked (disabled) or unmasked (enabled) and provides registers that report the raw interrupt status and the interrupt status after masking. The vector addresses for various interrupt sources are programmed through the VIC_VEC_ADDRESS registers. When one or more interrupt sources are active, the controller identifies the highest priority interrupt, stores the corresponding vector address in the VIC_ADDRESS register, and then asserts the nIRQ signal to interrupt the ARM CPU.

Refer to **section TBD** on for information about the various configuration and status registers associated with the VIC.

Some interrupts in the FX3 system need to be prioritized over others to ensure that the application can meet all the USB spec requirements and transfer rates. In particular, the USB core interrupt must be handled at the highest priority (can be selected as FIQ), and all the DMA interrupts should be allotted the next high priority level.

To enable a specific interrupt source, the firmware has to do the following:

- Point the VIC_VEC_ADDRESS register to the ISR.
- Set the VIC_VECT_PRIORITY register value as desired.

- Enable the interrupt by setting the corresponding bit in the VIC_INT_ENABLE register.

The following code snippet shows the procedure for setting up Fx3IntHandler as the ISR for interrupt vector i:

```
CY_U3P_VIC_VEC_ADDRESS[i]   = Fx3IntHandler;/* ISR address. */
CY_U3P_VIC_VECT_PRIORITY[i] = 2;/* Set the priority to 2. */
CY_U3P_VIC_INT_ENABLE      |= (1 << i);/* Enable the interrupt. */
```

The VIC sets the VIC_ADDRESS register to point to the vector for the active interrupt with the highest priority before making the nIRQ signal active. The IRQ exception vector is designed to jump to the address pointed by the VIC_ADDRESS register.

The ISR is responsible for saving all the necessary context information, including the non-banked registers, while executing. Nesting of interrupts is not allowed by the VIC. The ISR needs to clear the interrupt by writing any value to the VIC_ADDRESS register before the next interrupt can be raised.

As direct interrupt nesting is not supported by the VIC, it is recommended that the handling of low-priority interrupts be deferred to allow higher priority interrupts to run with bounded latency. The firmware needs to mask out the deferred interrupts until the source of the interrupt has been cleared to avoid repeated interrupt calls. This can be done by setting the corresponding bit in the VIC_INT_CLEAR register.

```
CY_U3P_VIC_INT_CLEAR = (1 << i);/* Mask out interrupt vector i. */
```

Note: The ARM CPU uses the ARM instruction set when starting to execute any interrupt handlers. If the ISR uses the thumb instruction set, the firmware needs to ensure that the switch to thumb mode is done before entering the ISR.

Refer to the PL192 Technical Reference Manual for more details on the interrupt controller and its use.

2.3.1.9 CPU Operating Frequency

The operating clock for the FX3 CPU is derived from the input clock or crystal frequency.

First, the input clock is multiplied to generate the FX3 system clock. The system clock frequency is 384 MHz when the FX3 device is clocked using a 19.2-MHz crystal or a 38.4-MHz clock input. The system clock frequency is 416 MHz when the FX3 device is clocked using a 26-MHz or 52-MHz input clock.

The CPU clock is then derived from the system clock using a programmable divider. The minimum divisor supported is 2, which means that the maximum CPU clock frequency is 192 MHz or 208 MHz, depending on the clock source.

Note: While the multipliers used to derive the system clock are programmable, Cypress strongly recommends the use of the previously mentioned default frequencies. The device has not been tested for proper functioning at other frequencies.

The CPU clock frequency can be reduced by specifying divisor values greater than 2. As the clocks used by the DMA engine and the memory mapped I/O (MMIO) register interface are derived from the CPU clock, reducing the CPU clock frequency will also reduce them and result in reduced data transfer throughput. Reducing the clock frequency will also increase the effective interrupt latency and can cause USB specification compliance errors. Therefore, Cypress recommends that you use a reduced clock rate only when USB 3.0 connections are not active.

Refer to the chapter on FX3 Global Control (GCTL) for details on how to configure the device clocks.

2.3.1.10 CPU Power Modes

The FX3 CPU supports low-power modes, which can be used when the device is not active. The CPU supports the following power modes:

Normal mode: The CPU operates at a clock frequency determined by the programmed dividers.

Suspend mode: This applies to both the L1 and L2 modes of the FX3 device. The clock to the CPU is gated in this mode, and the CPU is placed in the wait for interrupt state. Firmware execution resumes from the next instruction, once the device wakes from the L1/L2 mode.

Standby mode: This applies to the L3 mode of the FX3 device. The CPU is powered down, while the program RAM content is retained. Firmware execution starts from the reset vector once the device wakes from the L3 mode.

2.3.1.11 Timers

The ARM CPU does not have any associated timer blocks. The FX3 device provides a pair of general-purpose timers that can also provide the watchdog functionality. These timers are provided as part of the Global Control block on the FX3 device. These timers operate on a 32-kHz input clock and can be configured in one of the free running counter, timer with interrupt, or watchdog reset modes.

Note: If the system provides a clock input through the CLKIN_32 pin of the FX3 device, the timer uses this clock. If not, the 32-kHz clock is derived from the system clock, which runs at about 200 MHz.

The WATCHDOG_CS register in the GCTL block controls the operation of these timers. The relevant bits in this register are shown in Table 2-5.

Table 2-5. Watchdog Timer Control Register

Field Name	Bit Range	Description
MODE0	1:0	0-Free running mode; counter wraps around after reaching 0xFFFFFFFF. 1-Interrupt mode; raises WATCHDOG_TIMER interrupt when lowest significant BITS0 bits of the counter are cleared 2-Reset mode; resets the FX3 when lowest significant BITS0 bits of the counter are cleared 3-Disabled
INTR0	2	Interrupt status for timer 0
BITS0	7:3	Number of bits to be considered when checking for counter limit
MODE1	9:8	Timer mode for timer 1
INTR1	10	Interrupt status for timer 1
BITS1	15:11	Number of bits to be considered when checking for counter limit

The actual counter values for the timers are stored in the WATCHDOG_TIMER0 and WATCHDOG_TIMER1 registers. When the timer is configured in reset mode, the firmware is expected to restore this register to its initial value before the counter limit (lowest BITS bits getting set) is reached.

Hint: As a single interrupt vector is used for both timers, the ISR needs to check the INTR bits in the WATCHDOG_CS register to identify the timer that triggered the interrupt.

3. Memory and System Interconnect

The memory subsystem on the FX3 device comprises the system RAM that forms the main memory, SRAM controller, and AHB-based interconnect that allows the ARM CPU and the hardware blocks to access these memories. The MMIO interconnect provides access to registers in various peripheral blocks.

Because USB data moves through the system RAM (where USB endpoint buffers are implemented), FX3 implements a specialized memory controller to arbitrate between the various types of traffic with high throughput and predictable latency. Details on the arbitration mechanism and priorities are provided in System Interconnect on page 48.

3.1 Features

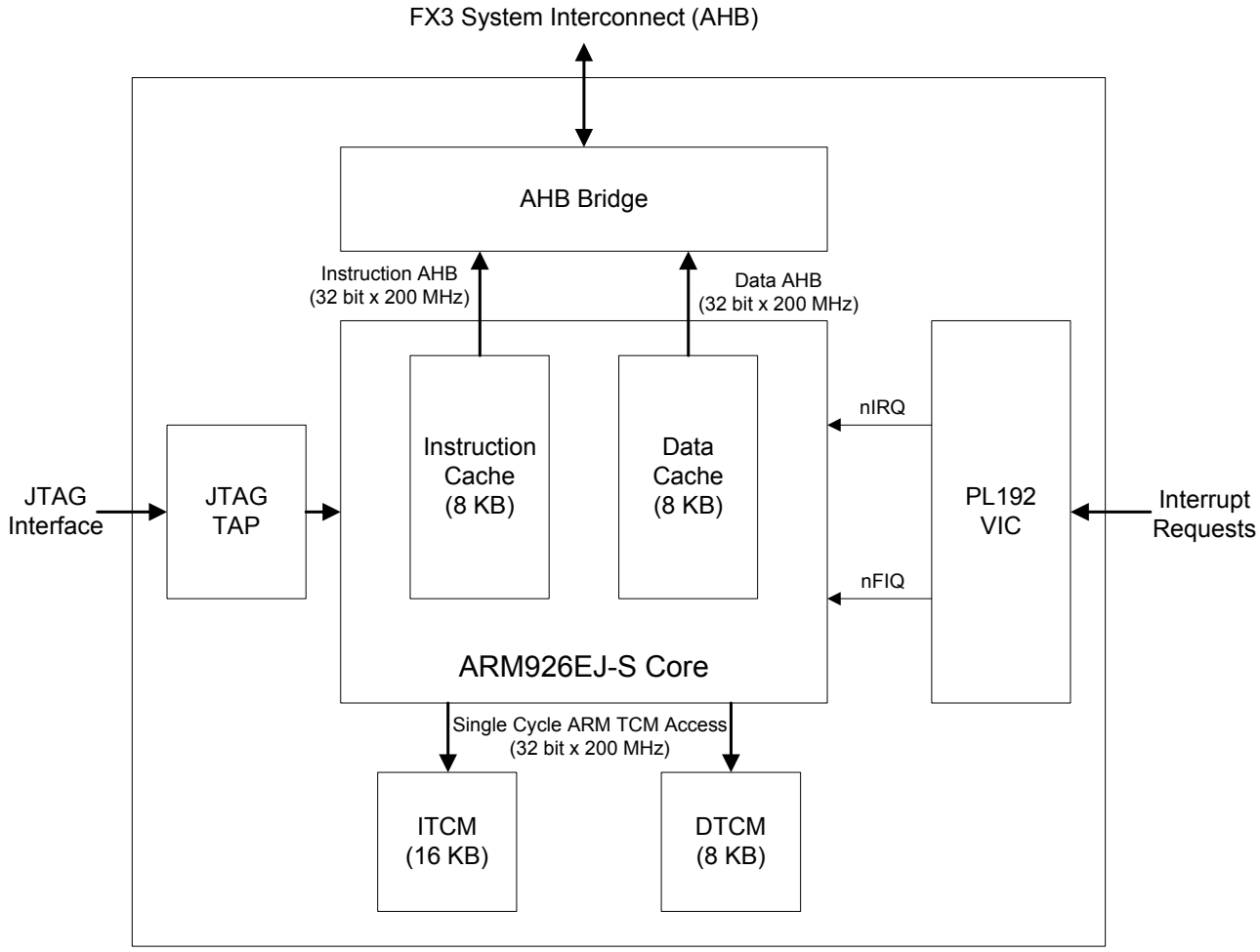
The FX3 memory and system interconnect supports the following:

- 512 KB or 256 KB of system memory, depending on the FX3 part number selected
- 16 KB of Instruction Tightly Couple Memory (I-TCM) and 8 KB of Data Tightly Couple Memory (D-TCM).
- DMA architecture that can deliver 800 MBps bandwidth to memory
- MMIO register access from CPU at up to 50 MBps (12.5 million 32-bit register accesses per second)
- Guaranteed and bounded memory access latency for both CPU and DMA accesses

3.2 Block Diagram

Figure 3-1 shows a block diagram of the memory and system interconnect on the FX3 device.

Figure 3-1. CPU Subsystem in the FX3 Device

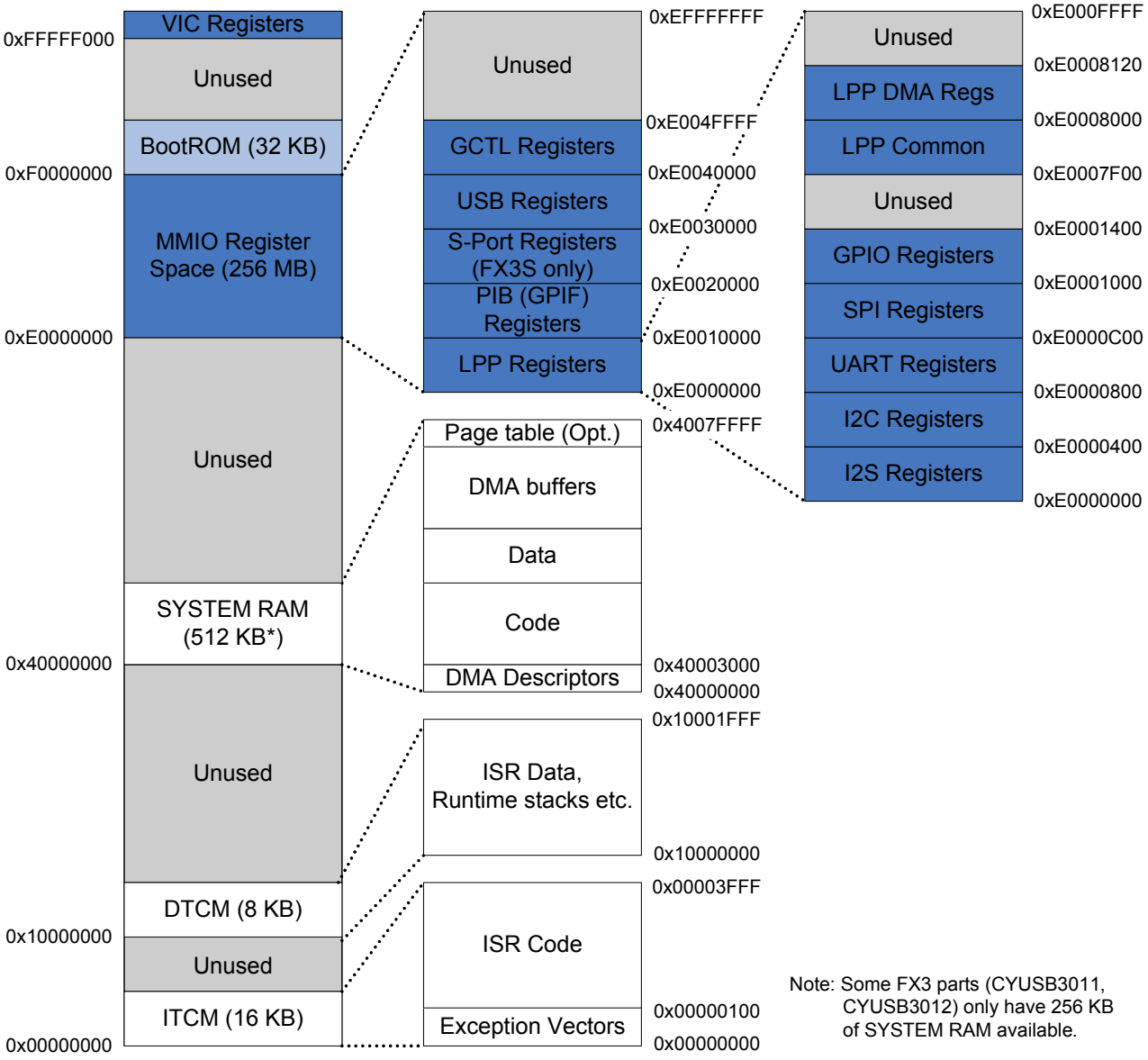


3.3 Functional Overview

3.3.1 Memory Regions

Figure 3-2 shows the memory map of the FX3 device.

Figure 3-2. FX3 Device Memory Map



Note: The memory map shown is for the CYUSB3014 part, which implements 512 KB of system RAM. For other parts such as CYUSB3011, the system RAM region is limited to 256 KB (from 0x40000000 to 0x4003FFFF).

The major memory regions on the FX3 device are the following:

- ITCM-16 KB dedicated space for holding exception vectors and ISR code. While it is possible to read and write to the ITCM memory region from the ARM CPU, it is not possible to use this memory region as a target for DMA transfers.
- DTCM-8 KB memory region that can be used for holding frequently accessed data structures, ISR data, run-time stacks, and more. It is not possible to use this region as a target for DMA transfers.
- System RAM-The main SRAM region that is used for code and data storage as well as for buffering any data that is flowing through the FX3 device. The System RAM region can be 256 KB or 512 KB, depending on the FX3 part being used. The first 12 KB of this region is reserved for storing DMA-related data structures (descriptors) that are used by the FX3 hardware. The remainder of the System RAM can be used as required by the application. Figure 3-2 shows the commonly used subdivisions for the RAM region.

- MMIO-The register space that holds all the configuration and status registers implemented by all the blocks on the FX3 device. While a total memory region of 256 MB has been allocated for the registers, most of this memory is unused and unimplemented. Refer to the chapters on each of the FX3 hardware blocks for information on the registers corresponding to those blocks.
- BootROM-A 32 KB ROM region that is preprogrammed with the FX3 device bootloader. The bootloader implements multiple boot modes such as USB boot, I2C boot, SPI boot, and GPIF boot. The desired boot mode is selected through a set of pin straps. This memory region is not accessible to FX3 user applications.
- VIC-Control and status registers for the VIC block. They are separate from the other MMIO registers and are located at the address 0xFFFFF000.

3.3.2 System Interconnect

The FX3 device implements a hierarchical AHB-based interconnect system that allows the device interfaces and the ARM CPU to access the system memory with high throughput and bounded latency. Different parts of the FX3 device function at different clock rates. The ARM CPU typically runs on a 200 MHz clock and has separate 32-bit wide buses for instruction and data access. The DMA interconnect runs at 100 MHz and has separate 64-bit buses for read and write accesses. The MMIO interconnect runs at 100 MHz and has a single 32-bit bus for all register accesses. The system RAM is organized as 128-bit wide memories and is clocked at 200 MHz.

As the DMA interconnect provides separate buses for read and write transfers, it can issue one read access and one write access during each DMA clock cycle. This means that the DMA interconnect can simultaneously support DMA read and DMA write traffic at 800 MBps each (100 MHz times 8 bytes in the 64-bit bus equals 800 MBps).

The system interconnect supports the following kinds of transfers:

- CPU traffic to system RAM
- DMA traffic to system RAM
- CPU accesses to MMIO registers
- DMA accesses to MMIO registers

Note: DMA access to MMIO registers is used to synchronize data transfers between a pair of communicating hardware blocks.

A specialized FX3 memory controller arbitrates between all these accesses. The memory controller connects to the high-speed system interconnect, which has two 128-bit wide buses running at 200 MHz. The memory controller guarantees equal SRAM bandwidth for CPU and DMA accesses. It also allows the DMA controller to use any unused CPU cycles so that typical FX3 applications can sustain a higher bandwidth.

3.3.3 Low-Power Operations

FX3 supports low-power operating modes in which the device clocks can be turned off and most of the blocks can be powered off. Table 3-1 shows the state of various FX3 blocks in the different power modes.

Table 3-1. State of FX3 Blocks in Low-Power Modes

Device Block	Normal Mode	Suspend (L1) Mode	Suspend (L2) Mode	Standby (L3) Mode
ARM9 CPU	ON	Waiting for interrupt	Waiting for interrupt	OFF
System RAM	ON	Clock gated	Clock gated	Clock gated
USB block	ON	Clock gated	OFF	OFF
GPIF	ON	Clock gated	Clock gated	OFF
Serial peripherals (UART, I2C, I2S and SPI)	ON	Clock gated	Clock gated	OFF
GPIO	ON	Holds previous state	Holds previous state	Holds previous state

The device supports two variants of suspend modes: L1 and L2. All the device blocks will be in the clock gated state in the L1 mode. This mode can be entered when the USB connection to the FX3 device has been suspended by the host, and the device should be configured to wake up on any USB bus activity.

The USB block is powered off in the L2 suspend mode. This mode can be entered only if the VBus input to the device is turned off. The device can be configured to wake up when the VBus input is detected.

In the standby mode, most of the blocks on the device are powered off. Only the system RAM and the logic required to detect wakeup requests are left powered on. Since the CPU as well as all the blocks including USB are powered off in this mode, firmware operation after wakeup is similar to that on a warm reset of the device.

3.3.4 Cache Operations

The FX3 device has dedicated instruction and data caches that improve access latencies to the SYSTEM RAM region. The caches are not used for TCM, MMIO, and VIC access. TCMs guarantee single-cycle access for both instructions and data, and they do not require a cache. The MMIO and VIC registers need to be updated atomically, and they are configured as not cacheable.

The ARM9 architecture supports the following cache operations:

- Flushing the entire I-cache or D-cache
- Cleaning (writing back to memory) the entire D-cache
- Flushing (evicting) a memory region from the I-cache or D-cache
- Cleaning (writing back to memory) a memory region from the D-cache
- Loading a specific memory region into the I-cache or D-cache

Please refer to the ARM926EJ-S Technical Reference Manual or the ARM System Developer's Guide for details on how to perform these operations.

Enabling the instruction cache is recommended for all FX3 applications. Enabling the data cache helps improve performance in applications where the FX3 CPU performs data manipulations.

3.3.4.1 Cache Coherency

Almost all FX3 applications involve transferring data in or out through one or more of the device interfaces. All these data transfers are achieved through the distributed DMA fabric on the device and make use of a part of the system RAM for buffering.

As the system RAM is used for DMA buffers as well as for code and data storage, there is a possibility of cache/memory corruption when the D-cache is enabled. The firmware application needs to avoid this possibility using the following guidelines. These steps are handled by the FX3 firmware library when the DMA APIs in the SDK are used.

1. As data is loaded or evicted from the cache one line at a time, it is likely that any data that shares a cache line with a DMA data buffer will be corrupted. Ensure that no code/data shares a cache line with DMA buffers to avoid this possibility.

   Hint: It is recommended that all DMA buffers be located in a separate memory region within the system RAM. It is also recommended that each DMA buffer occupy an integral number of cache lines (32 bytes) to ensure that an adjacent DMA buffer is not corrupted by a data transfer.

2. Ensure that the memory region corresponding to a DMA buffer is cleaned from the D-cache before initiating any egress (data output from FX3) data transfers.
3. Ensure that the memory region corresponding to a DMA buffer is flushed (evicted) from the D-cache before initiating any ingress (data input into FX3) data transfers.

3.3.5 Memory Usage

The TCM and system RAM regions on the FX3 device are general purpose and can be used by the firmware application as desired. The only constraints are as follows:

- The initial part of the I-TCM (approximately 256 bytes starting at address 0) is reserved for setting up the ARM exception vectors.
- The initial part of the system RAM (12 KB starting from address 0x40000000) is reserved for setting up DMA transfer-related data structures.
- The TCM regions cannot be used for direct data transfers, as the DMA engine does not support data transfers from/to these regions.

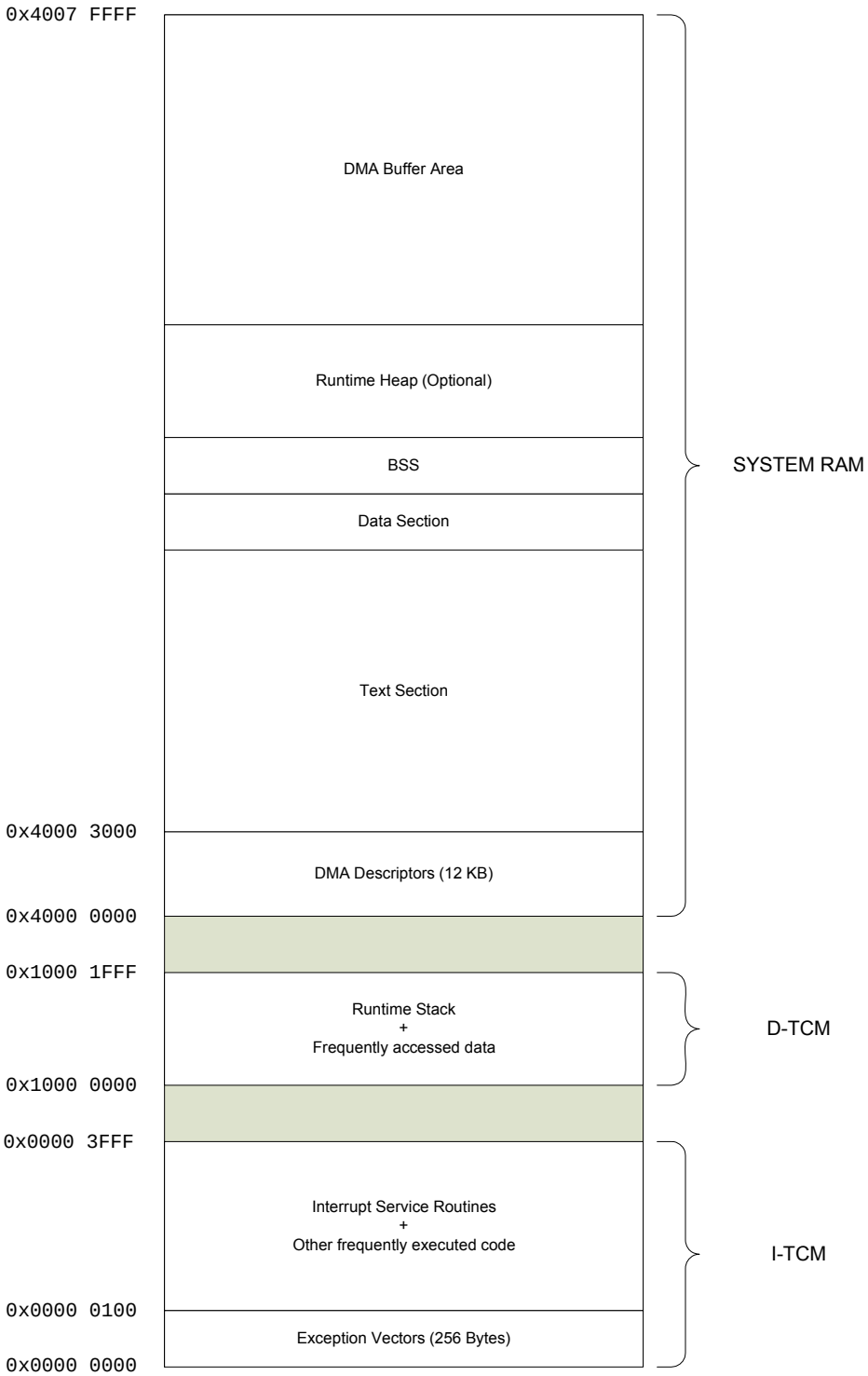
This section provides guidelines on mapping out the firmware code, data, and DMA buffers within the available memory on the device.

Table 3-2 shows the memory sections required by an FX3 firmware application. Figure 3-3 shows the mapping of these sections to FX3 memory addresses in a typical application. The size of the sections other than exception vectors and DMA descriptors can be set according to the needs of the application.

Table 3-2. Memory Regions Used by an FX3 Application

Section Name	Description
Vectors	ARM exception vectors; needs to be located at the beginning of the I-TCM region
Text	All executable code for the application
Data	Explicitly initiated global data used by the application
BSS	Uninitialized global data used by the application
Stacks	Stack regions for the ARM processor operating modes (supervisor, user, IRQ, FIQ, abort, and undefined)
Heap	Run-time heap for dynamic allocation (malloc or new) of variables used by the application; this section is optional and is only required if dynamic memory allocation (malloc, free, new, delete) is used
DMA descriptors	Memory region reserved for DMA transfer-related data structures; these structures are used by the FX3 hardware and have to be located at the beginning of the system RAM region
DMA buffer	Memory region reserved for DMA data buffers used by the application; as larger DMA buffers can improve data transfer throughput, it is recommended that as much memory as is possible be allocated to this section

Figure 3-3. Memory Map for Typical FX3 Firmware Application



The memory map for the firmware application is specified through a linker script file. The format of the linker script file used by the standard GNU C compiler for ARM processors is documented here.

4. Global Controller (GCTL)

The FX3 Global Controller (GCTL) supports I/O configuration, clock management, power management, and watchdog timer configuration. The GCTL features include the following:

- Can generate up to 500-MHz master clock
- Serves as a clock source for various peripherals (like UIB, PIB, and LPP) in FX3
- Supports simple and complex GPIO configuration
- Supports special function (like SPI and GPIF II) I/O configuration
- Supports watchdog timer configuration
- Supports power mode control and various wakeup source configuration

4.1 GPIO Pins

All 60 GPIO pins in FX3 can function as GPIOs. Each is multiplexed to support other functions/peripheral blocks (like UART, SPI, and so on). By default, the pins are allocated in groups to either one function block or the other, depending on the interface mode, in their respective power domains. In a typical application, all FX3 peripheral blocks are not used. Also, not all pins of the blocks being used are utilized. Unused pins in each block may be overridden as simple or complex GPIO pins on a pin-by-pin basis.

Simple GPIOs provide software-controlled and observable input and output capability only. In addition, they can also raise interrupts. Complex GPIOs add three timer/counter registers for each group and support a variety of time-based functions. They work off a slow or fast clock. Complex GPIOs can also be used as general-purpose timers by the firmware. There are eight complex I/O pin groups, the elements of which are chosen in a modulo 8 fashion (complex I/O group 0: GPIO 0, 8, 16; complex I/O group 1: GPIO 1, 9, 17, and so on). Each group can have different complex I/O functions (like PWM, one shot, and so on). However, only one pin from a group can use the complex I/O functions. The rest of the pins in the group are used as block I/O or simple GPIO. Refer to Table 7 in the *EZ-USB FX3 datasheet* for the GPIO configuration options.

4.1.1 I/O Matrix Configuration

I/O matrix configuration is used to configure the interface mode for I/O pins. The GCTL_IOMATRIX register must be configured before accessing any alternate function pins. Table 4-1 lists the I/O pin alternate functions.

Table 4-1. I/O Pin Alternate Function(s)

Pin Names	Alternate Function	Comments
GPIO[0] to GPIO[15]	DQ[0] to DQ[15]	GPIF II data pins
GPIO[16]	PCLK or CLK	GPIF II clock pin
GPIO[17] to GPIO[29]	CTL[x]	GPIF II control pins
GPIO[30] to GPIO[32]	PMOD[x]	Boot mode
GPIO[33] to GPIO[44]*	DQ[16] to DQ[27]	GPIF II data pins
GPIO[45]	-	No alternate function
GPIO[46] to GPIO[49]**	DQ[28] to DQ[31] or UART	GPIF II data pins or UART pins
GPIO[50] to GPIO[52] and GPIO[57]	I2S	
GPIO[53] to GPIO[56]**	SPI or UART	
GPIO[58] to GPIO[59]	I2C	

* 24- or 32-bit GPIF II bus width is not supported by all FX3 chips. If the FX3 chip does not support more than a 16-bit bus width, then alternate functions are not applicable. Refer to the EZ-USB FX3 datasheet for more details.

** If the GPIF II bus width is configured to 8 or 16 bits, then UART lines are available on the GPIO[46] to GPIO[49] pins, and SPI lines are available on the GPIO[53] to GPIO[56] pins. If the GPIF II bus width is configured to 24 or 32 bits, then UART lines are available on the GPIO[53] to GPIO[56] pins, and SPI is not supported.

Table 4-2 lists the FX3 registers associated with GPIO pin control. These registers are described in detail in following tables.

Table 4-2. Registers Associated with GPIO Pins

GCTL_IOMATRIX	GCTL_DS
GCTL_WPU_CFG0	GCTL_WPU_CFG1
GCTL_WPD_CFG0	GCTL_WPD_CFG1
GCTL_GPIO_SIMPLE0	GCTL_GPIO_SIMPLE1
GCTL_GPIO_COMPLEX0	GCTL_GPIO_COMPLEX1
LPP_GPIO_ID	LPP_GPIO_POWER
LPP_GPIO_PIN_STATUS(n)	LPP_GPIO_PIN_TIMER(n)
LPP_GPIO_PIN_THRESHOLD(n)	LPP_GPIO_SIMPLE(n)
LPP_GPIO_DRIVE_LO_EN	LPP_GPIO_DRIVE_HI_EN
LPP_GPIO_INPUT_EN	LPP_GPIO_PIN_INTR
LPP_GPIO_INVALUE0	LPP_GPIO_INVALUE1
LPP_GPIO_INTR0_REG	LPP_GPIO_INTR1_REG

See I/O Matrix Configuration Register on page 247

The FX3 SDK API CyU3PDeviceConfigureIOMatrix configures the GCTL_IOMATRIX register. A code snippet to configure the I/O matrix follows. Refer to the SDK firmware example for the complete details on the IOMATRIX configuration.

```
/* Configure the IO matrix for the device.
     * 32 bit bus width is disabled.
     * S0 port is disabled.
     * S1 port is disabled.
     * UART is enabled on S1 port.
     * IOs 43, 45, 52 and 57 are chosen as GPIO. */
     */
    io_cfg.isDQ32Bit        = CyFalse;
    io_cfg.s0Mode           = CY_U3P_SPORT_INACTIVE;
    io_cfg.s1Mode           = CY_U3P_SPORT_INACTIVE;
    io_cfg.gpioSimpleEn[0]  = 0;
    io_cfg.gpioSimpleEn[1]  = 0x02102800;
    io_cfg.gpioComplexEn[0] = 0;
    io_cfg.gpioComplexEn[1] = 0;
```

```
io_cfg.useUart          = CyTrue;
io_cfg.useI2C           = CyFalse;
io_cfg.useI2S           = CyFalse;
io_cfg.useSpi           = CyFalse;
io_cfg.lppMode          = CY_U3P_IO_MATRIX_LPP_UART_ONLY;
status = CyU3PDeviceConfigureIOMatrix (&io_cfg);
if (status != CY_U3P_SUCCESS)
{
    goto handle_fatal_error;
}
```

4.1.2 I/O Drive Strength

The drive strength for I/O pins is programmable even if the pin is configured for an alternate function. The I/O pin drive strength can be set to quarter strength, half strength, three-quarter strength, or full strength by configuring the appropriate bits in the GCTL_DS register.

See I/O Drive Strength Configuration Register on page 252.

In the FX3 SDK, I/Os on the FX3 device are grouped based on function into multiple interfaces (GPIF, I2C, I2S, SPI, UART). The I/O drive strength for each group can be separately configured using APIs.

- CyU3PSetPportDriveStrength
- CyU3PSetI2cDriveStrength
- CyU3PSetGpioDriveStrength
- CyU3PSetSerialIoDriveStrength

4.1.3 GPIO Pull-up and Pull-down

FX3 supports internal weak (50 K ohm) pull-up or pull-down I/O pins. A weak pull-up on I/O can be enabled by setting GCTL_WPU_CFG registers. A weak pull-down on I/O can be enabled by setting the GCTL_WPD_CFGx registers. The default state of the IOs at power-on and after reset is tristate.

See the following:

- GCTL_WPU_CFG on page 254
- GCTL_WPD_CFG on page 256

The FX3 SDK API CyU3PGpioSetIoMode is used to set pull-up or pull-down on an I/O pin.

4.1.4 Simple GPIO Override

FX3 supports the simple GPIO override option to configure any I/O as a simple GPIO pin. Register CY_U3P_GCTL_GPIO_SIMPLE is used to set the simple GPIO override. Data pins DQ[0[to DQ[31] can be configured as Simple GPIOs using IO matrix configuration and the override is not needed whereas all other GPIO pins needs the override to function as simple GPIO. Refer to GPIO on page 198 for more details on GPIO configuration.

See GCTL_GPIO_SIMPLE on page 248.

The FX3 SDK API CyU3PDeviceGpioOverride is used to configure the CY_U3P_GCTL_GPIO_SIMPLE register.

4.1.5 Complex GPIO Override

FX3 supports the complex GPIO override option to configure any I/O as a complex GPIO pin. Register CY_U3P_GCTL_GPIO_COMPLEX is used to set the complex GPIO override. Refer to GPIO on page 198 for more details on GPIO configuration.

See GCTL_GPIO_COMPLEX on page 250.

The FX3 SDK API CyU3PDeviceGpioOverride is used to configure the CY_U3P_GCTL_GPIO_COMPLEX register.

4.1.6 I/O Power Observability

Three GCTL registers-GCTL_IOPOWER, GCTL_IOPWR_INTR, and GCTL_IOPWR_INTR_MASK-allow the firmware to monitor the power status of various I/O blocks.

4.1.6.1 GCTL_IOPOWER

See GCTL_IOPOWER on page 258.

4.1.6.2 GCTL_IOPWR_INTR

See GCTL_IOPOWER_INTR on page 260.

4.1.6.3 GCTL_IOPWR_INTR_MASK

See GCTL_IOPOWER_INTR_MASK on page 262.

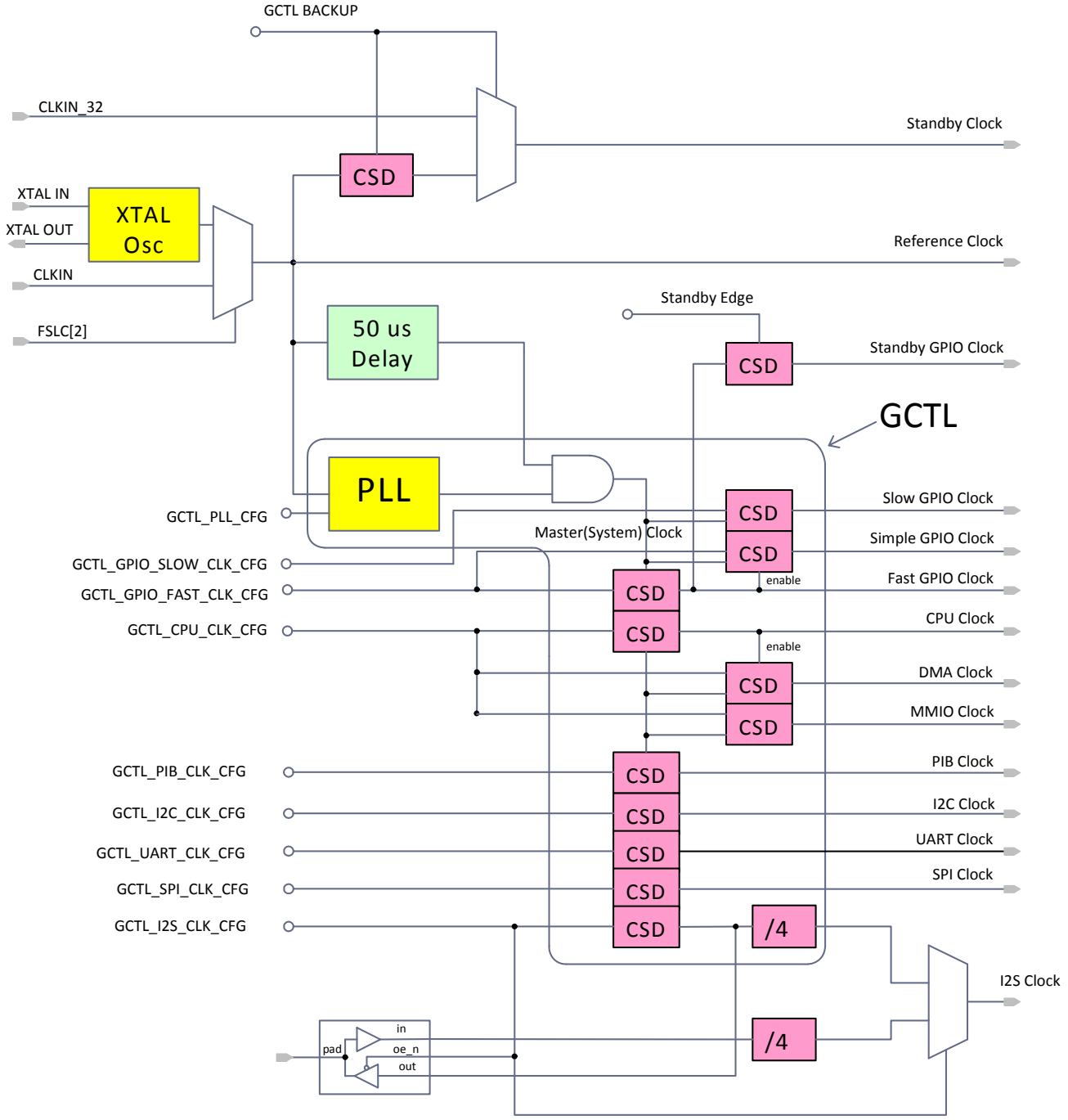
4.2 Clock Management

Clocks for FX3 peripherals can be configured using the GCTL registers. To enable an FX3 functional block (UART, SPI, and so on), configure the corresponding clock. The clock can be disabled for the unused functional blocks to save power.

As shown in Figure 4-1, CLKIN and PLL clock are the two input clock sources to the GCTL block. CLKIN is the chip reference clock provided by a 19.2-MHz, 26-MHz, 38.4-MHz, or 52-MHz external clock source or a 19.2-MHz crystal. It is used as the source clock for the PLL, which generates a master clock at frequencies up to 500 MHz. This PLL output clock is used to generate all the core clocks in the system.

Programmable dividers generate clocks in GCTL (except the blocks that contain their own PLL, for example, USB block). All generated clocks have a configurable divide capability and on/off programmability.

Figure 4-1. Clock System Block Diagram



Four system clocks are obtained by dividing the master clock by 1, 2, 4, and 16. The system clocks are then used to generate clocks for most peripherals in the device through the respective Clock Select and Divide (CSD) block. A CSD block is used to select one of the four system clocks and then divide it using the specified divider value. The depth of the divider is different for different peripherals.

The CPU clock is derived by selecting and dividing one of the four system clocks by an integer factor between 1 and 16. The bus clocks are derived from the CPU clock. Independent 4-bit dividers are provided for both the DMA and MMIO bus clocks.

The frequency of the MMIO clock, however, must be an integer divide of the DMA clock frequency. It is not recommended to drop the frequency of the CPU clock and DMA clock while the device is handling any data traffic.

A 32-kHz external clock source is used for low-power operation during standby. In the absence of a 32-kHz input clock source, the application can derive it from the input reference clock.

Certain peripherals deviate from the clock derivation described above. The fast clock source of GPIO is derived from the system clocks using a CSD. The core clock is a fixed division of the fast clock. The slow clock source of GPIO is obtained directly from the reference clock using a programmable divider. The standby clock is used to implement wakeup on GPIO. The I2S block can be run off an internal clock derived from the system clocks or from an external clock sourced through the I2S_MCLK pin of the device.

Exceptions to the general clock derivation strategy are blocks that contain their own PLL, because they include a PHY that provides its clock reference, for example, the USB2PHY and the USB3PHY. Refer to Universal Serial Bus (USB) chapter on page 81 for more details on the USB block clocking.

The CPU, DMA, and MMIO clock domains are synchronous to each other. However, every peripheral assumes its core clock to be fully asynchronous from other peripheral core clocks, the computing clock, or the wakeup clock.

Table 4-3. Registers Associated with Clock Management

CY_U3P_GCTL_PLL_CFG	CY_U3P_GCTL_GPIO_FAST_CLK
CY_U3P_GCTL_CPU_CLK_CFG	CY_U3P_GCTL_GPIO_SLOW_CLK
CY_U3P_GCTL_UIB_CORE_CLK	CY_U3P_GCTL_I2C_CORE_CLK
CY_U3P_GCTL_PIB_CORE_CLK	CY_U3P_GCTL_UART_CORE_CLK
CY_U3P_GCTL_SIB0_CORE_CLK	CY_U3P_GCTL_SPI_CORE_CLK
CY_U3P_GCTL_SIB1_CORE_CLK	CY_U3P_GCTL_I2S_CORE_CLK

4.3 Power Management

Table 4-4. Registers Associated with Power Management

CY_U3P_GCTL_CONTROL	CY_U3P_GCTL_FREEZE
CY_U3P_GCTL_WAKEUP_EN	CY_U3P_GCTL_WATCHDOG_CS
CY_U3P_GCTL_WAKEUP_POLARITY	CY_U3P_GCTL_WATCHDOG_TIMER0
CY_U3P_GCTL_WAKEUP_EVENT	CY_U3P_GCTL_WATCHDOG_TIMER1

4.3.1 Power Domains

Power supply domains in FX3 can be classified in four categories: core power domain, memory power domain, I/O power domain, and always-on power domain.

The core power domain encompasses a large section of the device, including the CPU, peripheral logic, and interconnect fabric. The system SRAM resides in the memory power domain. I/O logic dwells in the respective peripheral I/O power domain. The peripheral I/O power domain includes the I2C-IO power domain, I2S-IO power domain, UART-IO power domain, SPI power domain, IO-GPIO power domain, Clock IO power domain, USB IO power domain, and Processor Port IO power domain. The always-on power domain hosts the power management controller, the wakeup sources, and their associated logic.

Wakeup sources force a system in the suspend or standby state to switch to the normal power operation mode. These are distributed across peripherals and configured in the always-on global configuration block. Some of them include level match on level sensitive wakeup I/Os, toggle on edge sensitive wakeup I/Os, activity on the USB 2.0 data lines, OTG ID change, LFPS detection on USB 3.0 RX lines, USB connect event, and watchdog timer-timeout event.

The always-on global configuration block runs off the standby clock and is turned off only in the lowest power state (core power down).

4.3.2 Power Modes

At any instant, FX3 is in one of the four power modes: normal, suspend, standby, or core power down. When FX3 is actively executing its tasks, the system is in normal mode. The clock gating techniques in peripherals minimize the overall power consumption.

On detecting prolonged periods of inactivity, the firmware can place FX3 in suspend mode. All ongoing port (peripheral) activities/transfers are completed, ports are disabled, and wakeup sources are set before entering the suspend state. In applications involving USB 3.0, the USB3 PHY is forced into the U3 state. USB2PHY, if used, is forced into suspend. The system RAM transitions to a low-power mode in which read and write to RAM cannot be performed. The CPU is forced into the halt state. The ARM core retains its state, including its program counter. All clocks except the 32-kHz standby are turned off by disabling the system PLL through the global configuration block. In the absence of clocks, the I/O pins can be frozen to retain their state as long as the I/O power domain is not turned off The INT# pin can be configured to indicate the presence of FX3 in low-power mode.

Further reduction in power is achieved by having the firmware place FX3 into the standby state, where in addition to disabling clocks, the core power domain is turned off. As in suspend mode, the I/O states of powered peripheral I/O domains are frozen and the ports are disabled. The essential configuration registers of logic blocks are first saved to the system RAM. Then the system RAM itself is forced into the low-power memory retention only mode. The warm boot setting is enabled in the global configuration block. Finally, the core is powered down. When FX3 comes out of standby, the CPU goes through a reset; the bootloader senses the warm boot mode and restores the system to its original state after reloading the configuration values (including the firmware resume point) from the system RAM.

Optionally, FX3 can be placed in core powered-down mode from standby mode, which also involves removing power from the VDD pins. The contents of system SRAM are lost, and I/O pins retain their states, if suitably configured in the firmware. When power is reapplied to the VDD pins, FX3 performs the normal power-on reset (POR) sequence.

4.3.3 Reset

Resets in FX3 are classified into two categories: hard reset and soft reset.

4.3.4 Hard Reset

A POR or a Reset# pin assertion initiates a hard reset. This sets all register bits to their default states and restarts the program but retains the states of all register bits.

4.3.5 Soft Reset

A soft reset is generated by setting the appropriate bits in the GCTL_CONTROL register. There are two types of soft resets: CPU reset and whole device reset.

- CPU resets the CPU program counter. The firmware does not need to be reloaded following a CPU reset.
- Whole device reset is identical to hard reset. The firmware must be reloaded following a whole device reset.

5. FX3 DMA Subsystem

5.1 DMA Introduction

At the heart of the FX3 is a sophisticated, distributed DMA controller that is capable of moving data at 800 MBps that allows high-performance data transfers between memories and peripherals without CPU intervention. Multiple Advanced High-performance Buses (AHB, as defined by the ARM System Architecture) are used to interconnect the system elements. The EZ-USB FX3 device architecture includes a DMA fabric that is used to route data between various peripheral interfaces and/or the system memory of the device.

This chapter focuses on FX3 DMA transfer basics and the registers FX3 firmware uses to initialize and initiate DMA transfers. For a more advanced and practical DMA usage model, including types of DMA channels and common data transfer scenarios, refer to the "DMA Engine" section in the "FX3 Firmware" chapter of the *FX3 Programmers Manual*.

5.2 DMA Features

The DMA subsystem in the FX3 device includes the following features:

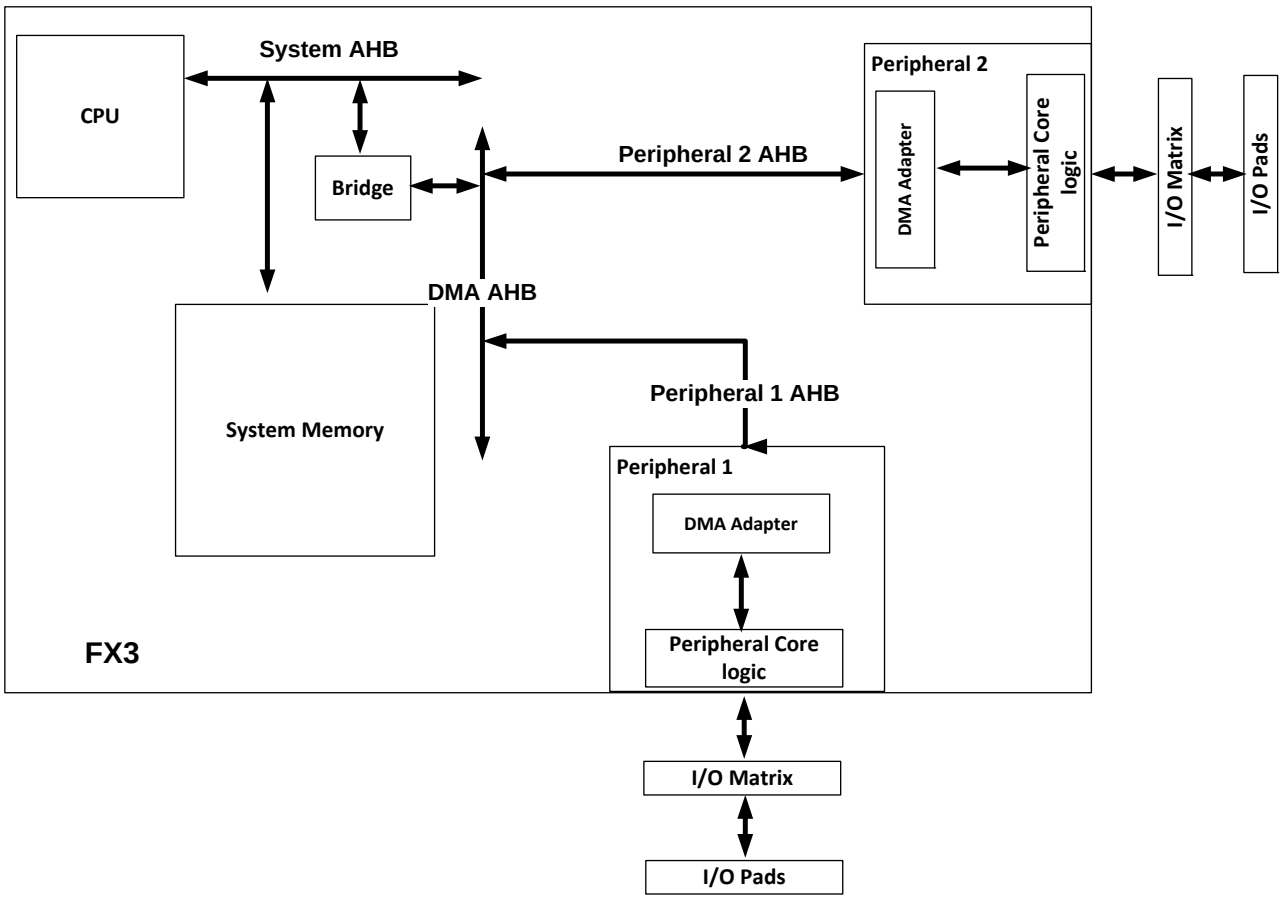
- Distributed DMA controllers
- Data transfer support in either direction between:
- Memory and peripheral
- Peripheral and peripheral
- Two gateways (virtual ports) of the same peripheral
- Localized DMA adapter (local DMA controller) to each peripheral

5.3 DMA Block Diagram

Non-CPU-intervened data transfers between a peripheral and CPU (system memory) or between two different peripherals or between two different gateways of the same peripheral are collectively referred to as DMA in FX3. All the data in the DMA subsystem flows through the system memory.

The Advanced Microcontroller Bus Architecture - Advanced High Performance Bus (AMBA AHB) interconnect forms the central nervous system of FX3. More details on AHB can be understood from the section "Interconnect Fabric" under chapter "FX3 Overview" in *FX3 Programmer's Manual*. Figure 5-1 shows how the CPU accesses the System Memory using the System AHB. All peripheral DMA paths connect to the DMA AHB. Bridges between the System bus and the DMA bus are essential in routing the DMA traffic through the System memory. The width of a peripheral connection to the AHB determines its throughput. The peripheral core implements the actual logic of the peripheral (I2C, GPIF, and USB).

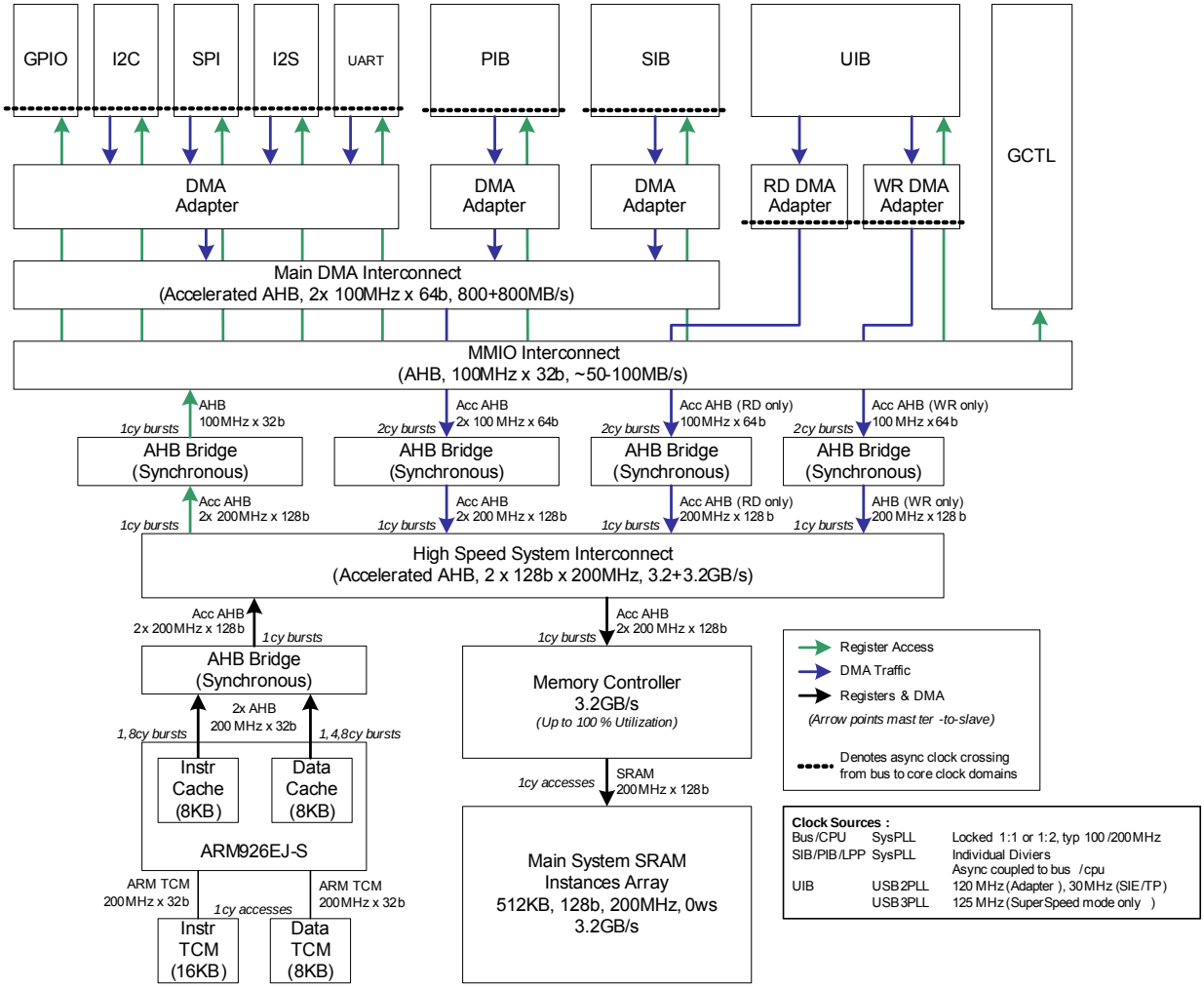
Figure 5-1. Block Diagram of FX3 DMA Subsystem



5.4 DMA Overview

FX3 contains a standard, configurable, DMA adapter that is replicated for each DMA-capable peripheral, as depicted in Figure 5-2. This architecture provides the FX3 distributed DMA controllers. There is only one DMA adapter for all low-performance peripherals. The DMA adapter is essentially a local DMA controller that initiates DMA transactions to and from the system memory on behalf of the peripheral that it services. With hardware synchronization between DMA adapters, data transfers can occur seamlessly between peripherals.

Figure 5-2. Distributed DMA Controllers



5.5 DMA Subsystem Components

5.5.1 Clocking

The FX3 DMA subsystem runs on an internal DMA bus clock, dma_bus_clk_i, that is divided down from the CPU clock. The DMA bus clock divider value is determined by the DMA_DIV field of GCTL_CPU_CLK_CFG, as shown below:

DMA Bus clock divider = (GCTL_CPU_CLK_CFG.DMA_DIV + 1)

See GCTL_CPU_CLK_CFG register on page 267

Divide by 1 (i.e. GCTL_CPU_CLK_CFG.DMA_DIV = 0) is illegal and will result in undefined behavior. Thus, the range of allowed divider values is 2 to 16 (GCTL_CPU_CLK_CFG.DMA_DIV => 1 to 15).

A typical dma_bus_clk_i frequency is set to one-half the CPU clock during device initialization. For example, if the CPU clock is set to 192 MHz, the register setting GCTL_CPU_CLK_CFG.DMA_DIV=1 (divider = 2) will result in a 96 MHz of dma_bus_clk_i frequency. Table 5-1 summarizes the DMA clock information.

Note: The maximum DMA clock frequency is half of the CPU clock frequency, as the minimum allowed divider value is '2'. Therefore, reducing the CPU clock frequency will result in reducing the DMA clock frequency and limit system performance. The default clock settings for FX3 are:

- CPU clock = System clock / 2
- DMA clock = CPU clock / 2

It is recommended that the default clock settings be retained in all cases.

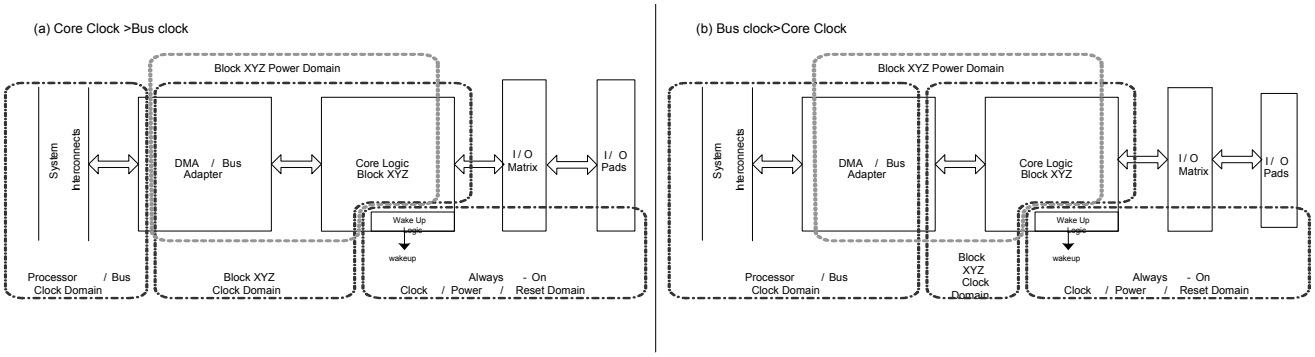
Table 5-1. DMA Clock

Domain	Typ/ Max Freq	Configuration Register Source	Description
dma_bus_clk_i	100 MHz	GCTL_CPU_CLK_CFG.DMA_DIV	DMA access clock

The CPU, DMA, and MMIO clock domains are synchronous to each other. However, every peripheral assumes its core clock to be fully asynchronous from other peripheral core clocks, the computing clock, or the wakeup clock.

If the core (peripheral) clock is faster than the bus clock, the DMA adapter for the block runs in the core clock domain and the DMA adapter reconciles the clocks on its interconnect side. If the core clock is slower than the bus clock, the DMA adapter for that block runs in the bus clock domain and the DMA adapter reconciles the clocks on its core IP side. This is shown in Figure 5-3.

Figure 5-3. DMA Adapter Clock



5.5.2 Descriptors Buffers, and Sockets

DMA descriptors are DMA instructions in a set of registers allocated in the FX3 RAM. A DMA descriptor holds information about the address and size of the DMA buffer as well as pointers to the next DMA Descriptor. These pointers create DMA descriptor chains. Descriptors enable the synchronization between sockets as described below.

A DMA buffer is a section of RAM used for intermediate storage of data transferred through the FX3 device. DMA buffers are allocated from the system RAM by the FX3 firmware; their addresses are stored as part of DMA descriptors. Every buffer created in the system memory has a descriptor associated with it that contains buffer information such as its address, empty/ full status, and the next buffer/descriptor in the chain.

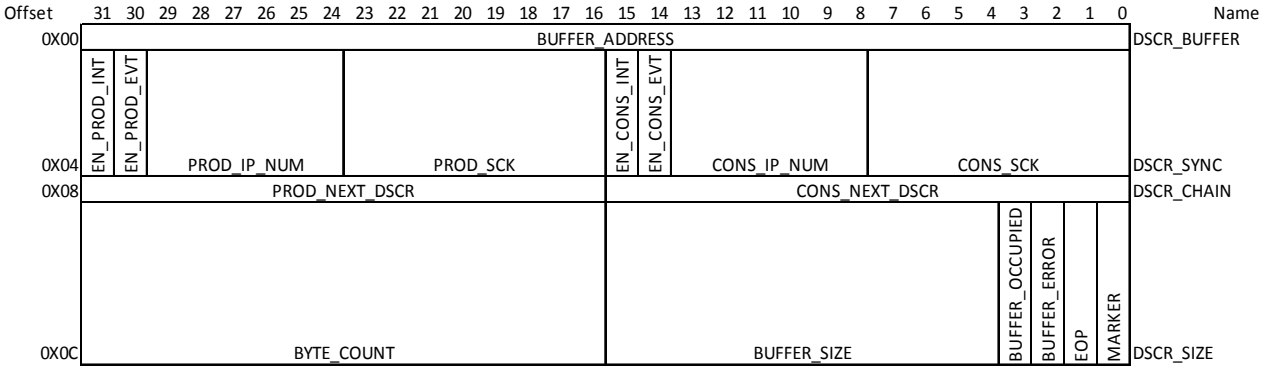
A socket is a point of connection between a peripheral hardware block and the FX3 RAM. Each peripheral hardware block on FX3 such as USB, GPIF, UART, and SPI has a fixed number of sockets associated with it. The number of parallel data channels through a peripheral is equal to the number of its sockets. The socket implementation includes a set of registers that point to the active DMA descriptor and the enable or flag interrupts associated with the socket. Sockets can directly signal each other through events or they can signal the FX3 CPU via interrupts. This signaling is configured by firmware.

5.5.3 DMA Descriptors

Descriptors are data structures that keep track of the resources (memory buffers and sockets) used for a DMA transfer. This data structure is directly interpreted by the DMA hardware on FX3, and has to be located in a specific memory region of the

FX3 RAM as described in the Memory and System Interconnect chapter on page 45. During a transfer, descriptors are loaded into the active socket one at a time for execution.

Figure 5-4. FX3 DMA Descriptor Structure



Each descriptor contains four 32-bit words. Figure 5-4 details the fields of the data structure.

DSCR_BUFFER provides the pointer of the buffer used for this transfer.

DSCR_SYNC defines the source and destination of this transfer. Depending on the use case, the event generation and interrupt can be enabled for either or both the consumer and producer half. Events are used to signal the peer socket, whereas interrupts are used to notify the CPU

A unique IP_NUM, which is assigned to each DMA-capable peripheral, is used for the source (producer) and destination (consumer) of the transfer. A unique IP_NUM, which is assigned to each DMA-capable peripheral, is used for the source (producer) and destination (consumer) of the transfer.

IP_NUM together with the socket number determines the actual socket identity. Table 5-2 details how sockets are identified by IP identification and socket number.

DSCR_CHAIN contains the pointers to next descriptors for the producer and consumer respectively.

DSCR_SIZE defines the size of the buffer and transfer byte count. The specific transfer status can also be set or monitored by the following bits:

- BUFFER_OCCUPIED indicates that data is available in the associated buffer.
- BUFFER_ERROR indicates whether the data is valid or in error.
  - EOP marks this transfer with end-of-packet. The socket can use this to suspend an EOP condition.
- MARKER is simply a software indicator for this specific transfer.

See the following register descriptions:

- DSCR_BUFFER on page 625
- DSCR_SYNC on page 626
- DSCR_CHAIN on page 628
- DSCR_SIZE on page 629

The following C code example is the DMA descriptor data structure used in the FX3 SDK.

```
/** \brief Descriptor data structure.

    **Description**\n
    This data structure contains the fields that make up a DMA descriptor on
    the FX3 device.

    Each structure member is composed of multiple fields as shown below. Refer
    to the sock_regs.h header file for the definitions used.
```

```
    buffer: (CY_U3P_BUFFER_ADDR_MASK)

    sync: (CY_U3P_EN_PROD_INT | CY_U3P_EN_PROD_EVENT | CY_U3P_PROD_IP_MASK |
        CY_U3P_PROD_SCK_MASK | CY_U3P_EN_CONS_INT | CY_U3P_EN_CONS_EVENT |
        CY_U3P_CONS_IP_MASK | CY_U3P_CONS_SCK_MASK)

    chain: (CY_U3P_WR_NEXT_DSCR_MASK | CY_U3P_RD_NEXT_DSCR_MASK)

    size: (CY_U3P_BYTE_COUNT_MASK | CY_U3P_BUFFER_SIZE_MASK | CY_U3P_BUFFER_OCCUPIED |
        CY_U3P_BUFFER_ERROR | CY_U3P_EOP | CY_U3P_MARKER)

    **\see\n
     *\see CyU3PDmaDscrGetConfig
     *\see CyU3PDmaDscrSetConfig
 */
typedef struct CyU3PDmaDescriptor_t
{
    uint8_t    *buffer;    /**< Pointer to buffer used. */
    uint32_t    sync;      /**< Consumer, Producer binding. */
    uint32_t    chain;     /**< Next descriptor links. */
    uint32_t    size;      /**< Current and maximum sizes of buffer. */
} CyU3PDmaDescriptor_t;
```

The actual DMA descriptors are maintained in the FX3 System RAM starting from address 0x40000010. The end of this space is not defined by hardware and is determined by the firmware. There are a fixed number of fixed-size descriptors in a fixed location in the main memory. Because of this, the hardware can directly access a descriptor with its descriptor number.

Firmware is responsible for setting up each of the descriptors and linking them to each other and to the sockets. DMA adapters will load the content of the active DMA descriptor in the socket register region as and when required.

5.5.4 DMA Buffer

DMA buffers are data buffers allocated in the system memory used for DMA. They can be of any size within the memory region and byte aligned. However, if the ARM data cache is enabled, it requires that the full buffer must be 32-byte aligned and of a size that is a multiple of 32 bytes.

A data packet contained in the buffer may be smaller than one buffer or even of zero size. A packet may be split over multiple buffers, or multiple packets can be in one buffer.

Every buffer created in the System memory has a descriptor associated with it. Descriptors are thus associated with buffers and have a producer, a consumer or both, or neither. A descriptor without a producer and consumer is called "free". Synchronization between producers and consumers happens at the level of descriptors and buffers. Multiple descriptors may refer to the same buffer. When more than one descriptor is used to describe the same buffer, it is the responsibility of FIRMWARE to make sure that the descriptors are coherent and synchronized with respect to each other. Descriptors contain meta-data about the data in the buffer (in addition to the location and size), in particular the number of valid bytes (refer to Figure 5-4 and DSCR_SIZE on page 629).

Since a 12-bit field (DSCR_SIZE.BUFFER_SIZE) is used to indicate the size of the DMA buffers in multiples of 16 bytes, the maximum size of an individual DMA buffer can be as much as 0xFFF0 bytes, as depicted below. Multiple such DMA buffers can be allocated for a single DMA transfer.

```
Maximum DMA buffer size allowed = (2^14 - 1) * 16 = 64Kb - 16B = 0xFFF0 bytes
```

5.5.4.1 Implications of Data Cache Usage

FX3 uses the same 512 KB of system RAM for code and data storage as well as for memory buffers that are used for DMA transfers into or out of the device. The ARM 926EJ-S core on the device also includes an 8 KB data cache. The data cache is four-way set associative with a cache-line size of 32 bytes and two dirty bits (one for each 16-byte region) per cache line. The

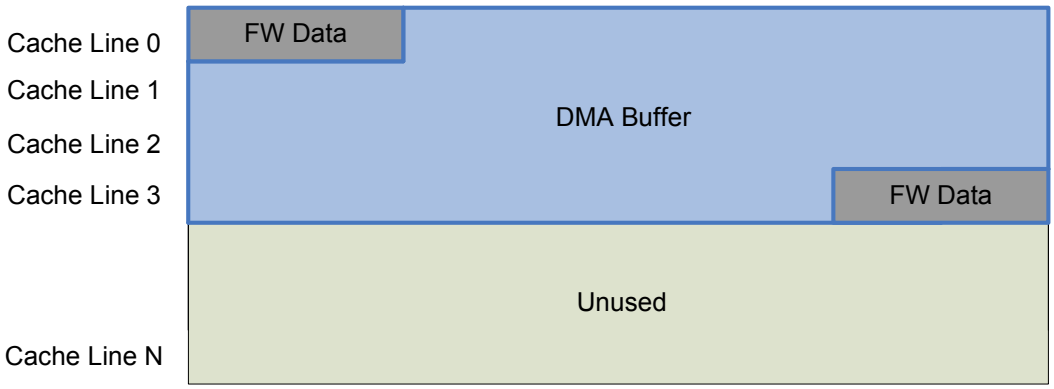
use of the data cache will give a good performance boost to any application that involves firmware access of the data buffer contents. However, this also leads to a risk of memory corruption in the cases where a cache line contains both software data structures and DMA data content. The following three sub-sections deal with this issue.

5.5.4.2 Memory Corruption Due to Cache Line Overlap

Consider the scenario represented in Figure 5-5. In this case, both cache lines 0 and 3 have an overlap of firmware data along with the DMA buffer space. If the DMA buffer is being filled with data by any of the hardware blocks on the device, and the firmware needs to access this data, memory corruption can happen as described below.

1. If the firmware tries to access the data in the buffer directly, stale memory content from the cache will be retrieved.
2. If the firmware flushes the cache line(s) and then accesses the data in the buffer, any updates to the firmware data structures will be lost.
3. If the firmware cleans the cache line(s) and then accesses the data buffer, the part of DMA buffer that shares a cache line with the firmware data will get corrupted.

Figure 5-5. Unsafe Overlap of Firmware Data with DMA Buffer

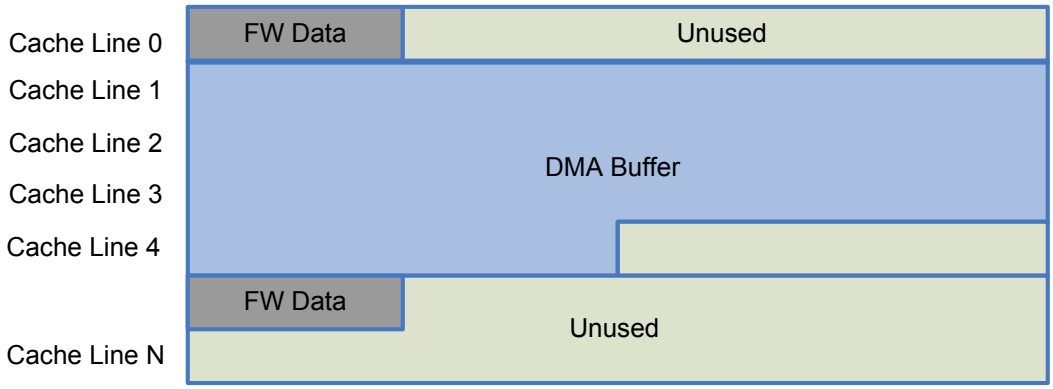


The firmware needs to ensure that the software data structures and DMA buffers do not share cache lines to prevent these problems.

5.5.4.3 Safe Usage of Data Cache

Consider the scenario in Figure 5-6. In this case, the DMA buffer does not share any cache lines with the firmware data.

Figure 5-6. Safe Overlap of Firmware Data with DMA Buffer



In this case, there is no possibility of the CPU and/or hardware seeing bad data due to the data cache. Whenever the CPU wants to read a data buffer that has been filled by the DMA hardware, it can flush the corresponding region from the cache and then initiate a read. Whenever the CPU wants to commit a buffer containing data for an egress DMA operation, it can clean the region from the cache and then initiate the DMA operation.

5.5.4.4 ALIGNMENT REQUIREMENT - How Not To Share Cache Lines

The basic requirement is that DMA buffers that may be modified by the hardware should not share a cache line with software data elements. This translates to a requirement that all DMA buffers which may be used for ingress (data coming in to FX3 and being written into memory by the DMA hardware) transfers should be 32-byte aligned and occupy an integral number of cache lines (32 bytes each).

This restriction only applies to DMA buffers that may be used for ingress data transfers. The restriction does not apply to DMA buffers that are used only as a source of egress data.

5.5.5 Sockets

A socket is the unidirectional virtual port (gateway) used by a peripheral (IP) block to transfer data to/ from the system SRAM. Each DMA transfer involves one or two sockets. A socket represents either the consuming or the producing half of a transfer. For a transfer from one peripheral to another, two sockets are involved. A socket is either a consuming socket or a producing socket at any point in time-not both at the same time.

An FX3 DMA-capable peripheral has multiple sockets in the DMA adapter. The number of sockets and their properties depend on the specific DMA adapter to the peripheral. Each peripheral block (IP block) in the device can support a predefined number of sockets which is the maximum number of independent data flows that can be done through that IP at a given point of time. A producer (ingress) socket is one which moves data from the IP block to the system SRAM. A consumer (egress) socket is one which takes data from the system SRAM and moves it out through the IP block. Each socket can be identified with the IP number and the socket number. Table 5-2 is the socket summary for each FX3 peripheral.

Note: Separate DMA adapters are used for USB IN and OUT endpoints to allow greater USB data bandwidth. Interrupts from both adapters are combined into a single interrupt vector.

Table 5-2. Peripheral DMA Sockets

FX3 Peripheral/ IP block	IP_NUM	Socket Count	Specific Property	Notes
GPIF II	0x01	32	Socket 0-15: Bidirectional Socket 16-31: Ingress only	
USB	0x03	16	Socket 0-15 for USB Egress: Egress Only	This maps to IN endpoints where FX3 sends data out.
USB-IN	0x04	16	Socket 0-15 for USB Ingress: Ingress Only	This maps to OUT endpoints where FX3 receives data.
Storage	0x02	8	All are Bidirectional	FX3S Only
Serial Peripherals (UART, I2C, I2S, SPI)	0x00	8	Socket 0-1 for I2S: Egress only Socket 2 for I2C data out: Egress only Socket 3 for UART data out: Egress only Socket 4 for SPI data out: Egress only Socket 5 for I2C data in: Ingress only Socket 6 for UART data in: Ingress only Socket 7 for SPI data in: Ingress only	The purpose of each socket in this adapter is fixed and cannot be changed.
CPU	0x3F	2	Socket 0 for CPU data in Socket 1 for CPU data out	

Each socket has its own register set that the firmware uses to control the DMA operations. The socket register base address is located at offset 0x8000 from the base address of the peripheral.

Each register set occupies a 128-byte address space with some gaps in between. Figure 5-7 details the socket registers and their field definitions.

SCK_DSCR on page 614 describes the current descriptor to be loaded for this socket.

SCK_SIZE on page 616 sets the amount of data to be transferred. A zero value in this register means the data amount is infinite.

SCK_COUNT on page 617 reports the amount of data transferred.

SCK_STATUS on page 618 is the socket control and status register.

SCK_INTR on page 621 is the interrupt request register that indicates the status of the interrupt source. The same register is used to clear the interrupt status.

SCK_INTR_MASK on page 623 is used to enable the interrupt source to CPU.

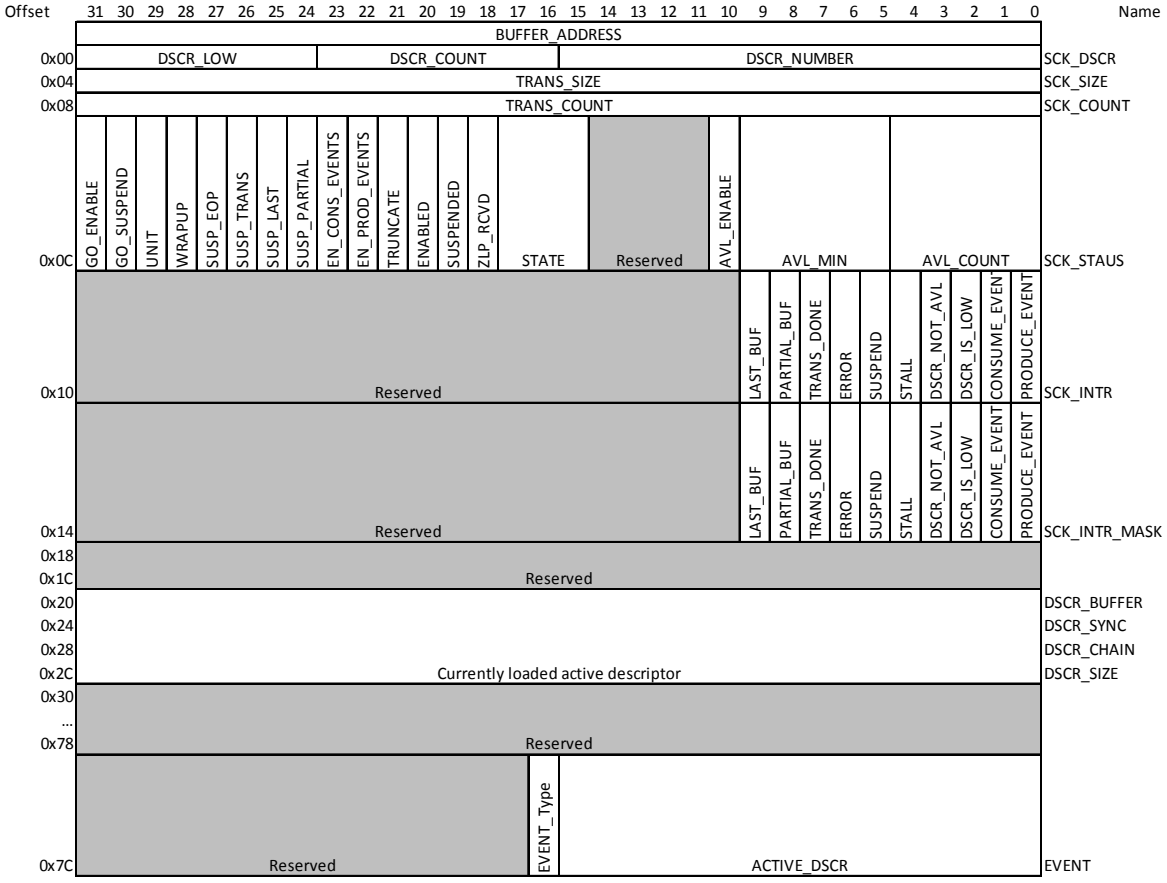
DSCR_BUFFER on page 625, DSCR_SYNC on page 626, DSCR_CHAIN on page 628, and DSCR_SIZE on page 629 indicate the currently loaded active descriptor in the socket. Figure 5-4 details the field definition of the descriptor.

EVENT is the socket-event communication register. FX3 DMA supports two possible events:

- Consume event: Data has been read out of the buffer.
- Produce event: Data has been filled into the buffer.

When a DMA transfer involves two sockets, the producer socket sends a produce event to the peer consumer socket after data is written to the buffer. Similarly, the consumer socket sends a consume event to the peer producer socket after the data is read from the buffer. The event signaling between the two sockets is usually handled by hardware, and the CPU is not involved through the entire transfer. However, there are situations when the firmware needs to generate the event manually by writing the appropriate event in the EVENT register.

Figure 5-7. FX3 DMA Socket Registers



For detailed field description, see the following:

- SCK_DSCR on page 614
- SCK_SIZE on page 616
- SCK_COUNT on page 617

- SCK_STATUS on page 618
- SCK_INTR on page 621
- SCK_INTR_MASK on page 623

Each socket can be identified to be in one of the states listed in Table 5-2. The SCK_STATUS.STATE field can be read to understand the socket state. More on the socket states will be discussed later.

Table 5-3. Socket States

Socket State	SCK_STATUS.STATE Value	Description
DESCR	0	Descriptor state. This is the default initial state indicating the descriptor registers are NOT valid in the adapter. The adapter will start loading the descriptor from the memory if the socket becomes enabled and not suspended. Suspend has no effect on any other state.
STALL	1	Stall state. The socket is stalled, waiting for data to be loaded into the Fetch Queue or waiting for an event.
ACTIVE	2	Active state. The socket is available for core data transfers.
EVENT	3	Event state. Core transfer is done. The descriptor is being written back into the memory and an event is being generated if enabled.
CHECK1	4	Check states. An active socket gets here based on the core's EOP request to check the transfer size and determine whether the buffer should be wrapped up. Depending on result, the socket will either go back to the Active state or move to the Event state.
SUSPENDED	5	The socket is suspended
CHECK2	6	Check states. An active socket gets here based on the core's EOP request to check the transfer size and determine whether the buffer should be wrapped up. Depending on result, the socket will either go back to the Active state or move to the Event state.
WAITING	7	Waiting for confirmation that the event was sent.

The following C code example shows the DMA socket data structure used in the FX3 SDK.

```
/** \brief DMA socket register structure.

    **Description**\n
    Each hardware block on the FX3 device implements a number of DMA sockets through
    which it handles data transfers with the external world. Each DMA socket serves as
    an endpoint for an independent data stream going through the hardware block.

    Each socket has a set of registers associated with it, that reflect the configuration
    and status information for that socket. The CyU3PDmaSocket structure is a replica
    of the config/status registers for a socket and is designed to perform socket configura-
tion
    and status checks directly from firmware.

    See the sock_regs.h header file for the definitions of the fields that make up each
    of these registers.

    **\see
    *\see CyU3PDmaSocketConfig_t
 */
typedef struct CyU3PDmaSocket_t
{
    uvint32_t dscrChain;              /**< The descriptor chain associated with the socket
*/
    uvint32_t xferSize;              /**< The transfer size requested for this socket. The
size can
                                           be specified in bytes or in terms of number of
buffers,
                                      depending on the UNIT field in the status value. */
    uvint32_t xferCount;             /**< The completed transfer count for this socket. */
    uvint32_t status;                 /**< Socket configuration and status register. */
```

```
    uvint32_t intr;                    /**< Interrupt status register. */
    uvint32_t intrMask;                /**< Interrupt mask register. */
    uvint32_t unused2[2];              /**< Reserved register space. */
    CyU3PDmaDescriptor_t activeDscr;   /**< Active descriptor information. See
cyu3descriptor.h for definition. */
    uvint32_t unused19[19];            /**< Reserved register space. */
    uvint32_t sckEvent;                /**< Generate event register. */
} CyU3PDmaSocket_t;
```

5.5.5.1 Software Manipulation of Sockets

Sockets are to be initialized, inspected, and modified by firmware as described in this section.

5.5.5.2 Initializing a Socket

Sockets can be initialized only when in the IDLE state, that is, SCK_STATUS.ENABLED=0. If this is not the case, the socket must first be terminated (see 5.5.5.3 Terminating a Socket on page 71).

The general procedure is as follows:

1. Descriptors are allocated or located and initialized in memory. These descriptors are chained appropriately.
2. The SCK_xxx registers are initialized with the proper configuration values. This includes SCK_DSCR, which contains the number of the first descriptor to be loaded. DSCR_xxx registers are not initialized - the DMA adapter will load those by itself.
3. SCK_STATUS.GO_ENABLE is set to '1' to activate the socket (if so desired).

5.5.5.3 Terminating a Socket

Sockets can be terminated at any time. If a socket is active, its activities will be aborted after an unspecified amount of time.

The general procedure is as follows:

1. SCK_STATUS.GO_ENABLE is cleared to '0'. The IP will continue to perform an unspecified amount of its pending activity.
2. It is permissible to write '0' multiple times to SCK_STATUS.GO_ENABLE while the socket is being terminated, but it is illegal to write '1' to it during this time.
3. SCK_STATUS.ENABLED is read back until its value changes to '0'. No further activity emanates from this socket after SCK_STATUS.enabled is observed as cleared.

5.5.5.4 Modifying or Suspending a Socket

Sockets are normally modified safely only when in the suspend state. A socket that is active or stalled must first be suspended. A socket that is suspended will not complete an ongoing transfer, but rather go into the suspended state almost immediately. It is possible though, that going into the suspend state safely takes a noticeable (but small) number of cycles.

The general procedure is as follows:

1. SCK_STATUS.GO_SUSPEND is set to '1'.
2. SCK_STATUS.SUSPENDED is read back until its value changes to '1' or SCK_INTR.SUSPEND is used as the interrupt source to indicate the the suspend status.
3. Any changes are made to SCK_xxx or DSCR_xxx registers.
4. SCK_STATUS.GO_SUSPEND is cleared. If SCK_DSCR.active_dscr was modified, the socket will load the new descriptor from the memory[otherwise it will resume the operation using the current contents of DSCR_xxx.

Note that SCK_STATUS.SUSPENDED will only take effect after the transfer of the current buffer completes. This may take a long time.

Note that it is also possible to modify sockets that are in the stalled state, provided that it is known that no synchronization EVENT(s) will occur. This is normally the case when the socket is waiting for firmware to generate an event, that is, the socket is not coupled to another socket in another adapter. In this case it is not necessary to first suspend the socket.

5.5.5.5 *Inspecting a Socket*

Sockets can be inspected at any time. When a socket is active, values are not guaranteed to be accurate due to the activity and clock domain crossing issues. The general procedure is as follows:

1. Any value in SCK_xxx or DSCR_xxx registers is read twice in succession.
2. If the values of two reads match they are accurate. If they are different, go back to step 1.

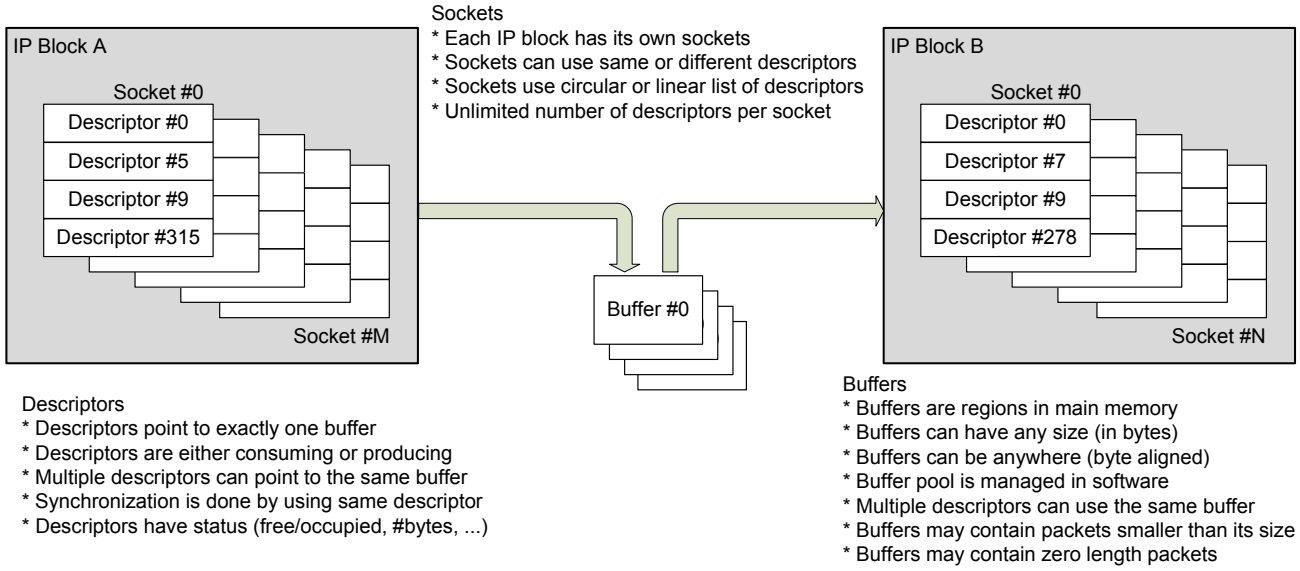
5.5.5.6 *Wrapping Up a Socket*

A socket that is in the Active state, but that is not expected to send/receive any more data can be forcibly wrapped up by asserting SCK_STATUS.WRAPUP. This should never be done for sockets that are not in the Active state or that may send/receive data. This option is normally used when the core IP has transitioned into an error or partial completion state and the firmware is required to clean up the remaining, unexecuted, portion of the transfer.

5.5.6 Illustration of Descriptor, Buffer and Socket Usage

Figure 5-8 below sums up the concepts discussed in sections 5.5.2 Descriptors Buffers, and Sockets on page 64 to 5.5.5 Sockets on page 68.

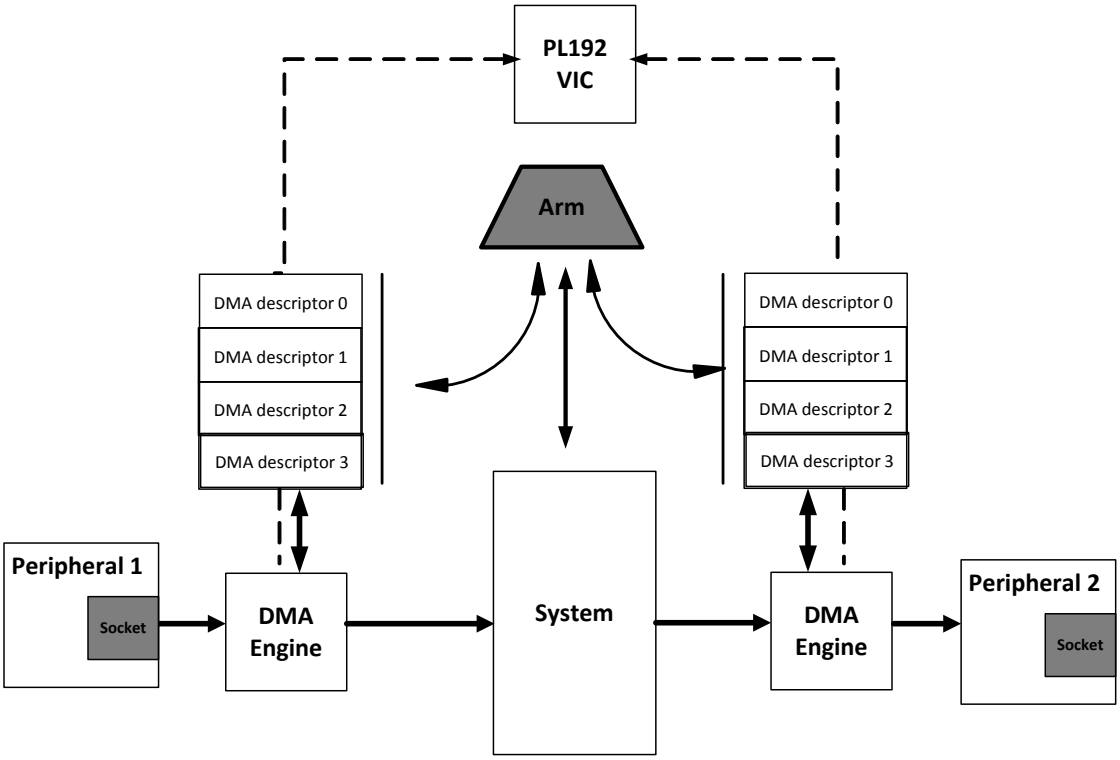
Figure 5-8. Sockets, Descriptors, and Buffers



5.5.7 Understanding DMA Operation: Peripheral to Peripheral

This section explains in a high level how DMA descriptors, buffers, and sockets are tied together to achieve the required DMA operation, with the help of an example peripheral-to-peripheral DMA operation.

Figure 5-9. FX3's DMA System



The producer behaves in the following way:

1. When a producer has filled a buffer (an end-of-packet or a buffer-full event occurred - note that packets can be empty), it updates the descriptor associated with the buffer in the main memory to indicate that the buffer is full (DMA-to-memory write transaction).
2. The producer then sends a produce event to the consuming socket of its descriptor (DMA-to-MMIO write transaction). This event contains the number of the descriptor to which it relates.
3. The producer then loads the next descriptor for the socket and makes it active (DMA-to-memory read transaction).
4. If the new descriptor indicates its buffer space is occupied, the socket stalls. If a consume event is received for the current descriptor, the descriptor is either updated using the data in the event or reloaded from the memory. (DMA-to-memory read transaction). DMA data write transfers may resume. Go to step (1)
5. If no descriptor is available (next_dscr=0xFFFF), the socket is suspended. When the software extends the descriptor list and explicitly 'resumes', the IP block operation continues. Go to step (3).

The consumer behavior is exactly symmetric:

1. When a consumer has emptied a buffer (an end-of-packet or a buffer empty event occurred), it updates the descriptor associated with the buffer in the main memory to indicate that the buffer is empty and available (DMA-to memory write transaction).
2. The consumer then sends a consume event to the producing socket of its descriptor (DMA-to-MMIO write transaction). This event contains the number of the descriptor to which it relates.
3. The consumer then loads the next descriptor for the socket and makes it active (DMA-to-memory read transaction).
4. If the new descriptor indicates its buffer is empty, the socket stalls. If a produce event is received for the current descriptor, the descriptor is either updated using the data from the event or reloaded from the memory. (DMA-to-memory read transaction). DMA data read transfers may resume. Go to step (1)
5. If no descriptor is available the socket is suspended. When the software extends the descriptor list and explicitly 'resumes', the IP block operation continues. Go to step (3).

Refer to the section "Setting up the DMA System" in *AN75779*, where it is explained graphically in a high-level how the socket, descriptor, and buffers are tied together in the FX3 DMA system.

5.5.8 Interrupt Requests

Each DMA-capable peripheral block has dedicated global interrupt request lines to the PL192 VIC. Refer to 2.3.1.8 Vectored Interrupt Controller on page 41 for more details. Table 2-4 on page 42 in 2.3.1.8 Vectored Interrupt Controller on page 41 lists the various FX3 interrupt sources. Table 5-4 describes the DMA interrupt lines.

Table 5-4. Global DMA Interrupt Request to VIC

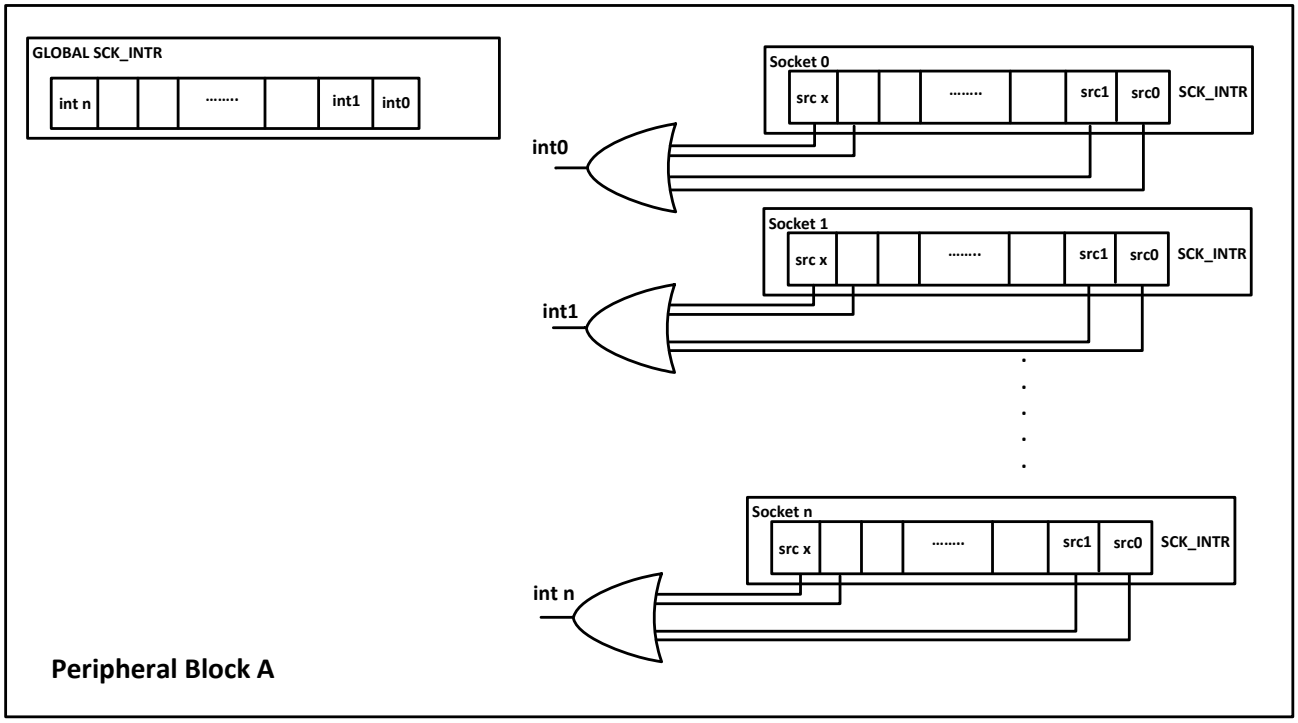
Interrupt	VIC Line	Interrupt Source	Description
GPIF_DMA	6	GPIF DMA adapter	DMA socket interrupt from the GPIF block
USB_DMA	8	USB DMA adapter	Applies to both USB device and host mode operation
STORAGE_DMA	11	Storage (SD/MMC) interface DMA adapter	Applies only to FX3S devices
PERIPH_DMA	20	Serial peripheral block DMA adapter	

5.5.9 DMA Interrupts

Although DMA transfer takes place independently from CPU execution, the CPU may need to be notified when certain transfer conditions are met, when an error has occurred, or when the transfer is complete. See SCK_INTR on page 632.

Each peripheral has a global SCK_INTR register, in which each bit represents the socket number that generates the interrupt. The bit description is as described above. There is also a per-socket SCK_INTR in which each bit represents the interrupt source from the corresponding socket. Bit description is explained in 5.5.5 Sockets on page 68. The logical OR of all socket interrupts presented in the per-socket SCK_INTR register represents the corresponding bit in the peripheral global SCK_INTR register. This is illustrated in Figure 5-10.

Figure 5-10. Global and Per-Socket SCK_INTR Register



When a DMA interrupt occurs, the CPU is notified by VIC with the DMA interrupt line specific to the peripheral, as shown in Table 5-4. Then the CPU can check the peripheral's global SCK_INTR register to find the socket number that needs attention.

The Global SCK_INTR register is read-only and the interrupt bits are cleared by clearing the interrupt cause or bit in the per-socket SCK_INTR register itself. Once the CPU finds the socket number, it can find the source of the interrupt from the per-socket SCK_INTR register of the corresponding socket.

5.6 Programming Sequence

5.6.1 Initialization

The default state of most sockets is indicated in the register map. Briefly, in the default state the socket is disabled and holds no descriptor. Sockets are initialized as described in the 5.5.5.2 Initializing a Socket on page 71 .The same steps are detailed below.

1. Initialize and allocate descriptor(s) in the main memory. The base address for descriptor allocation starts at 0x40000010. In addition to specifying where the data buffer is located, each descriptor data structure must be configured properly with its associated peripherals, sockets, and event/interrupt flags for both consumer and producer halves. If a list of descriptors is used, descriptors can be chained with DSCR_CHAIN field of the descriptor structure. The DSCR_CHAIN specifies the next descriptor number in the chain for both consumer and producer. A value of 0xFFFF terminates the descriptor chain.
2. Initialize sockets with SCK_xxx registers. In addition to specifying the associated descriptor (or top of the descriptor chain) for the socket, these registers control how the socket behaves during the transfer, that is, ., interrupt/event generation and current status of the socket.
3. Enable socket(s). Sockets can be enabled by writing the SCK_STATUS.go_enable bit. Once socket is enabled, it loads the descriptor specified in SCK_DSCR register (top of descriptor chain) and starts the transfer accordingly.

5.6.1.1 Producer Half

When the socket for the producer half is enabled, it loads the descriptor specified in the SCK_DSCR register and makes it active. With a valid buffer specified by the active descriptor, the socket goes to the active state and starts to fill the data buffer.

When the producer socket has filled the buffer, it updates the active descriptor associated with the buffer in main memory and then sends a produce event to its peer consuming socket defined in the dscr_sync field of the descriptor structure along with the descriptor number itself.

If there is a valid peripheral consumer socket specified in the descriptor, the producer notifies the consumer socket by writing to the EVENT register in the consumer socket. The EVENT value will indicate the DMA descriptor that has been updated and specify that a PRODUCE event is being sent.

If the descriptor does not specify a valid consumer socket, it is the firmware's responsibility to identify and work on the descriptor that has been produced. The SCK_INTR_MASK.PRODUCE_EVENT bit can be set to enable notification of produce events to the CPU through the DMA interrupt. The firmware can then take appropriate actions to use the data that has been received.

The producer socket then loads the next descriptor in the chain and makes it active. If the descriptor indicates its buffer space is not available, the socket goes to the stall state. When the buffer is available as indicated from the socket's EVENT register, the socket becomes active again and starts to fill the buffer pointed by the active descriptor.

If a consume event is received for the current descriptor, the descriptor is either updated using data in the EVENT register or reloaded from memory.

If no descriptor is available (next_dscr=0xFFFF), the socket goes to the suspend state.

A DMA transfer from a peripheral to the system memory involves only the producer half.

5.6.1.2 Consumer Half

Similar to the producer half, when the socket for the consumer half is enabled, it loads the descriptor specified in the SCK_DSCR register and makes it active. If the consumer socket is enabled at the same time as the producer socket, most likely the buffer is waiting to be filled by the producer and is not available for the consumer socket. In this case, the consumer socket goes to the stall state and waits for the buffer to become available upon a produce event.

When the buffer is available, it goes to the active state and starts consuming data in the buffer. When the consumer socket has emptied a buffer, it updates the descriptor associated with the buffer in main memory and then sends a consume event to its peer producer socket defined in the dscr_sync field of the descriptor structure, along with the descriptor number itself.

If the descriptor does not specify a valid producer socket, it is the firmware's responsibility to populate the data buffer in the next descriptor in the descriptor chain. The SCK_INTR_MASK.CONSUME_EVENT bit can be set to enable notification of consume events to the CPU through the DMA interrupt. The firmware can then take appropriate actions to populate the next buffer.

The consumer socket then loads the next descriptor in the chain and makes it active. If the descriptor indicates its buffer space is empty, the socket goes to the stall state. Until the buffer is filled as indicated from the socket's EVENT register, the socket becomes active again and starts to empty the buffer pointed by the active descriptor.

If a produce event is received for the current descriptor, the descriptor is either updated using data from the event or reloaded from memory.

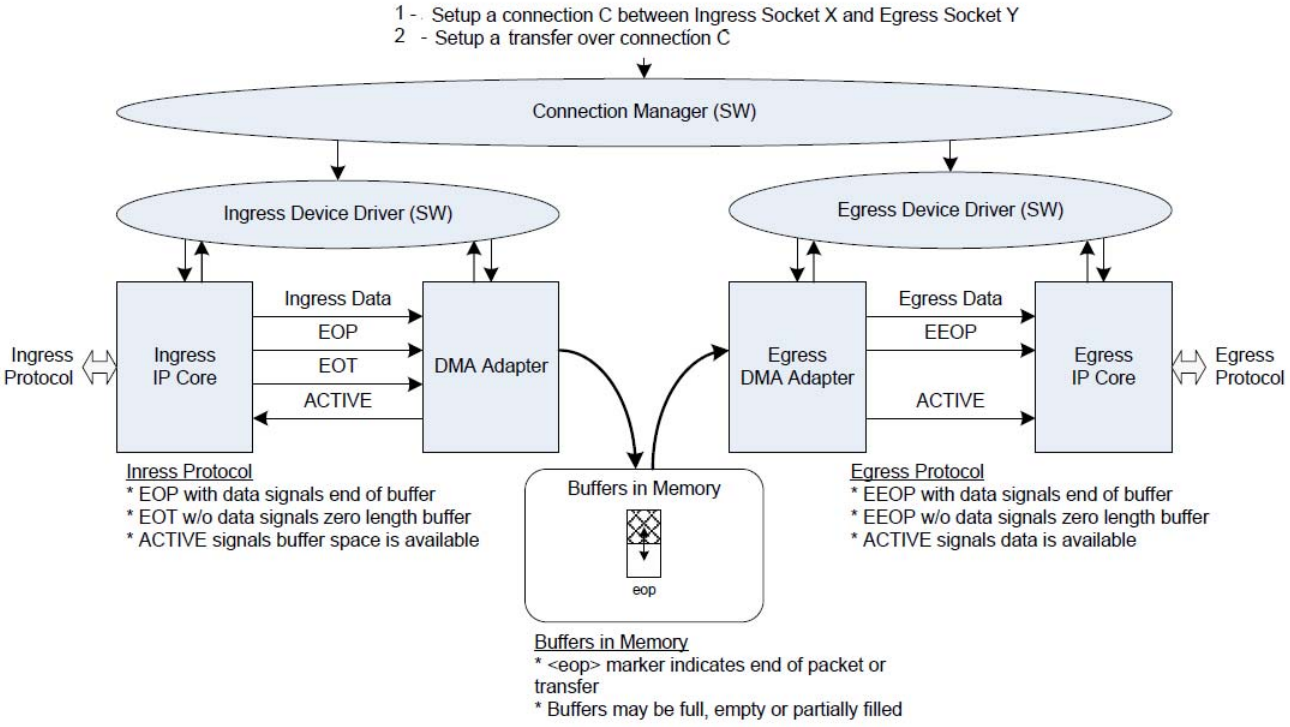
If no descriptor is available (next_dscr=0xFFFF), the socket goes to the suspend state.

A DMA transfer from the system memory to a peripheral involves only the consumer half.

5.6.2 Peripheral to Peripheral Transfer

A DMA transfer from one peripheral to another peripheral involves both the producer and consumer halves. The sequence of events for producer and consumer are identical to that when they are standalone, except when the producer half completes the transfer, a produce event will be sent from the producer socket to the peer consumer socket to trigger the consumer half. In this case, the whole peripheral to peripheral transfer can take place without CPU intervention in the transfer itself. Figure 5-11 depicts the connection model used to describe peripheral to peripheral transfers. It also considers the software drivers for the ingress and egress peripherals (that are mutually independent) and the higher level s/w that manages the endpoint.

Figure 5-11. Peripheral - Peripheral DMA Transfer



The example code shows how to setup transfers and process corresponding interrupts.

```
/* Summary:
   A one shot dma setup and transfer function.
 */
CyFx3BootErrorCode_t
CyFx3BootUsbDmaXferData (
        uint8_t epNum,
        uint32_t address,
        uint32_t length ,
        uint32_t timeout
        )
{
    uint8_t direction = epNum & 0x80;
    PSCK_T s;
    uint16_t id, sc, ch;
    PDSCR_T dscr = DSCR(0);
    uint32_t size = length;
    uint32_t *buf;

    CyFx3BootErrorCode_t status;

    if (((epNum & 0x0F) == 0) && (IsNewCtrlRqtReceived ()))
        return CY_FX3_BOOT_ERROR_ABORTED;

    buf = (uint32_t*)address;
    if (direction)
        s = (PSCK_T)&UIB->sck[epNum & 0x0F];
    else
        s = (PSCK_T)&UIBIN->sck[epNum & 0x0F];

    /* Initializing socket */
    id = ((uint32_t)s>>16) & 0x1f;              /* use bit16-20 to decode IP_NUM */
    ch = ((uint32_t)s>>7) & 0x1f;               /* use bit7-11  to decode socket number */

   s->status = CY_U3P_LPP_SCK_STATUS_DEFAULT;  /* Default SCK_STATUS register setting. Refer
                                                  SCK_STATUS register description */
    while (s->status & CY_U3P_LPP_ENABLED)      /* checking if the socket is active */
       __nop ();
    s->status = CY_U3P_LPP_SCK_STATUS_DEFAULT | CY_U3P_LPP_UNIT;  /* setting SCK_STATUS.unit
= 1*/
    s->intr   = 0xFF;                     /* clear all previous interrupt */
    s->dscr   = DSCR_ADDR(dscr);         /* Loading the first descriptor (top of descriptor
chain) */
    s->size   = 1;
    s->count  = 0;

    dscr->buffer = (uint32_t)buf;
    sc = (size & 0xf) ? (size & cSizeMask) + 0x10 : (size & cSizeMask);
    if (sc == 0)
        sc = 16;

    if (direction) /* 1=TX: SYSMEM to device     */
    {
        dscr->sync = (  (ch << CY_U3P_LPP_CONS_SCK_POS) | (id << CY_U3P_LPP_CONS_IP_POS) |
                (CPU_SCK_NUM << CY_U3P_LPP_PROD_SCK_POS) | (CPU_IP_NUM <<
CY_U3P_LPP_PROD_IP_POS) |
```

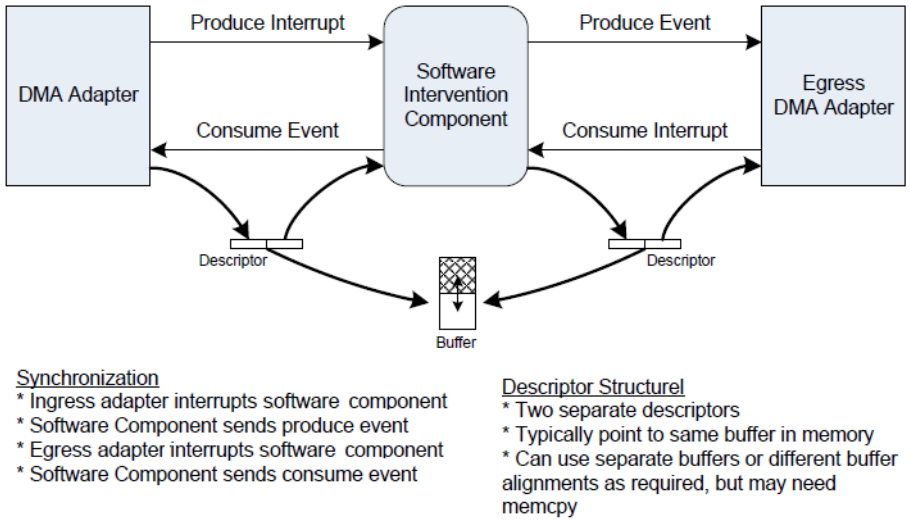
```
                (CY_U3P_LPP_EN_CONS_EVENT|CY_U3P_LPP_EN_CONS_INT)   |
(CY_U3P_LPP_EN_PROD_EVENT|CY_U3P_LPP_EN_PROD_INT)
                );
        dscr->size = ((size & (cTX_EN-1)) << CY_U3P_LPP_BYTE_COUNT_POS) | sc |
CY_U3P_LPP_BUFFER_OCCUPIED;
    }
    else   /* 0=RX: Device to SYSMEM          */
    {
        dscr->sync = (  (CPU_SCK_NUM << CY_U3P_LPP_CONS_SCK_POS) | (CPU_IP_NUM <<
CY_U3P_LPP_CONS_IP_POS) |
                (ch << CY_U3P_LPP_PROD_SCK_POS) | (id << CY_U3P_LPP_PROD_IP_POS) |
                (CY_U3P_LPP_EN_CONS_EVENT|CY_U3P_LPP_EN_CONS_INT) |
(CY_U3P_LPP_EN_PROD_EVENT|CY_U3P_LPP_EN_PROD_INT)
                );
        dscr->size = sc;
    }

    dscr->chain = (0xFFFFFFFF);

    s->status |= CY_U3P_LPP_GO_ENABLE;
    status = CY_FX3_BOOT_ERROR_TIMEOUT;

    do
    {
        if (s->intr & CY_U3P_LPP_ERROR)
        {
            status = CY_FX3_BOOT_ERROR_XFER_FAILURE;
            break;
        }

        UsbDelayFunction (100);

        if (direction)
        {
            if (s->intr & CY_U3P_LPP_CONSUME_EVENT)
            {
                status = CY_FX3_BOOT_SUCCESS;
                break;
            }
        }
        else
        {
            if (s->intr & CY_U3P_LPP_PRODUCE_EVENT)
            {
                status = CY_FX3_BOOT_SUCCESS;
                break;
            }
        }

        if ((timeout != CY_FX3_BOOT_NO_WAIT) && (timeout != CY_FX3_BOOT_WAIT_FOREVER))
        {
            timeout --;
        }

        if (((epNum & 0x0F) == 0) && (IsNewCtrlRqtReceived ()))
        {
            /* This request has been aborted due to a new control request. Just reset
               the USB socket and return an error. */
```

```
            s->status &= ~(CY_U3P_LPP_GO_ENABLE | CY_U3P_LPP_WRAPUP);
            s->intr    = 0xFF;
            while (s->status & CY_U3P_LPP_ENABLED)
                __nop ();

            /* Flush EP0-IN as well. */
            UIB->eepm_endpoint[0] |= CY_U3P_UIB_SOCKET_FLUSH;
            CyFx3BootBusyWait (10);
            UIB->eepm_endpoint[0] &= ~CY_U3P_UIB_SOCKET_FLUSH;

            status = CY_FX3_BOOT_ERROR_ABORTED;
            break;
        }

    } while (timeout > 0);

    return status;
}
```

5.7 CPU Intervention In Between Ingress and Egress

Although it is rarely needed, it is possible to interpose a CPU intervention in between an ingress and egress DMA adapter. Conceptually, this means the CPU (the software component) acts as the consumer agent to the producing (ingress) adapter and acts as the producer agent to the consuming (egress) adapter. This model is depicted in Figure 5-12.

Figure 5-12. CPU Intervention in DMA Transfer Path



This mode will have a significant negative performance impact on high-bandwidth transfers because the CPU gets into the critical path of every buffer transferred. This mode should only be used to handle special case stream requirements or to implement processing of the actual data by the CPU such as DSP applications.

5.8 Concept of DMA Channels

A DMA channel is a software construct that encapsulates all of the DMA elements used (discussed so far in this chapter) in a single data flow. The DMA manager in the FX3 firmware library introduces the notion of a DMA channel that encapsulates the hardware resources such as sockets, buffers and descriptors used for handling a data flow through the device. The channel concept is used to hide the complexity of configuring all of these resources in a consistent manner. The DMA manager provides API functions that can be used to create data flows between any two interfaces on the FX3 device.

The DMA channel implementation where sockets can directly signal each other through events or can signal the FX3 CPU via interrupts, when configured by firmware, is called automatic DMA channel. Alternatively, when there is CPU intervention as explained in section DMA Features on page 61, it is called a manual DMA channel. For more details on DMA channels and types of DMA channels supported, refer DMA Engine section in FX3 Programmer's Manual.

6. Universal Serial Bus (USB)

6.1 Introduction

USB is a successful peripheral interconnect defined and heavily adopted in consumer electronics and PC peripherals. The first version of the specification, USB 1.0, released in 1996, defined two transfer speeds to address the different types of devices available at that time: 1.5 Mbps (Low Speed) to address devices such as keyboards and joysticks; and 12 Mbps (Full Speed) to address devices such as disk drives. The USB 2.0 specification, released in 2000, supports a maximum signaling rate of 480 Mbps (High Speed), which is 40 times the signaling rate of Full Speed. With the introduction of USB OTG, USB went beyond a peripheral interconnect. It enabled printers to use USB to directly connect to cameras and PDAs to use USB-connected keyboards and mice. The new generation USB 3.0 is the next revolution in device interconnect technology. It features the same ease of use and flexibility that users expect, at a much higher (5-Gbps) data rate and advanced power management.

6.2 Features

The FX3 USB subsystem features the following controllers to support many advanced features of the USB standard:

- USB Interface Block (UIB)
  - USB 3.0 function controller
  - USB 2.0 function controller
  - USB 2.0 embedded host controller
  - USB 2.0 OTG controller
  - USB charger detect controller
- USB I/O system
  - Dedicated USB 2.0 OTG PHY and USB 3.0 PHY
  - Accessory Charger Adaptor (ACA) and integrated voltage regulator

The USB 3.0 and USB 2.0 subsystems have dedicated transceivers, but they share the same I/O interconnect in the back end. This means that only one of the USB 3.0 or USB 2.0 controller can be active at a time.

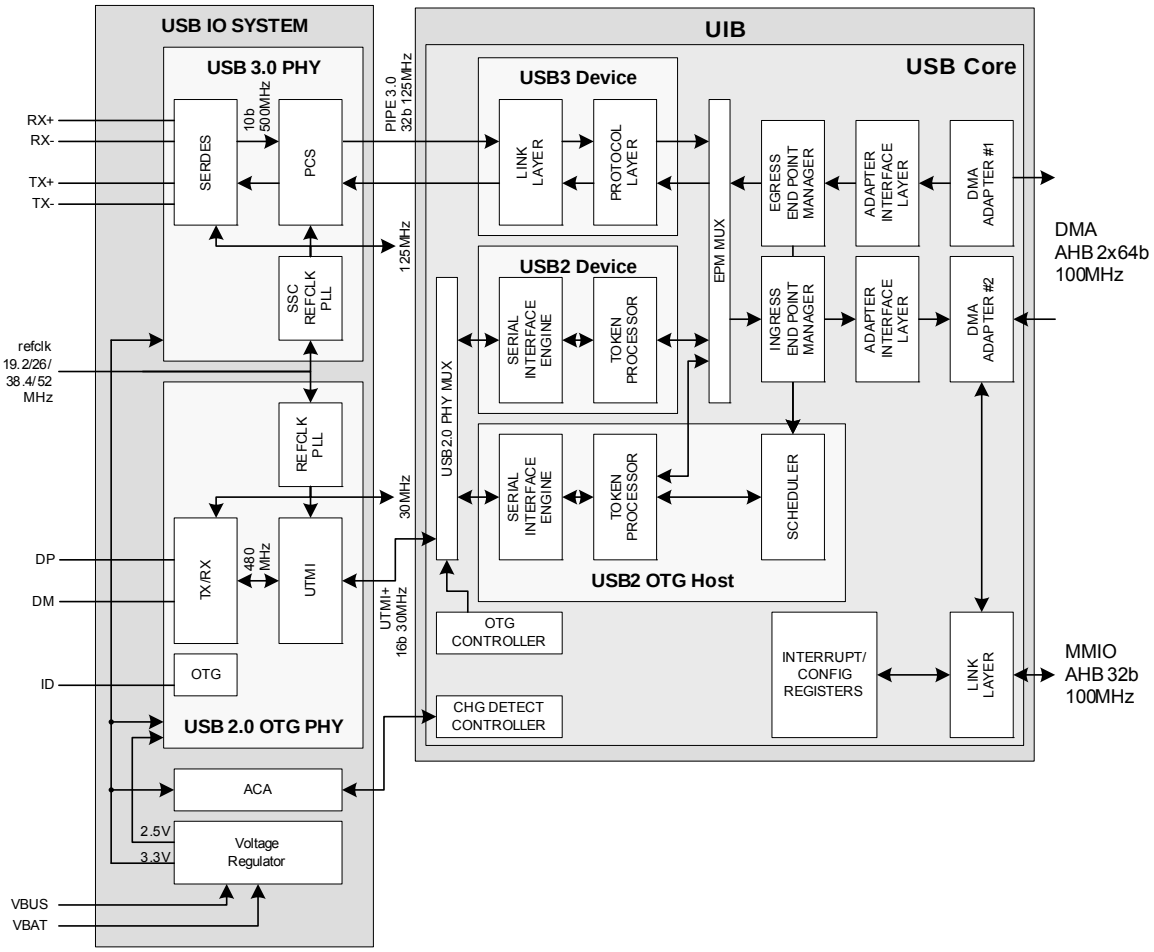
The FX3 USB subsystem supports the following modes of operation:

- USB 3.0 peripheral in SuperSpeed (5 Gbps)
- USB 2.0 peripheral in High/Full Speed (480/12 Mbps, respectively)
- USB 2.0 host in High/Full/Low Speed (480/12/1.5 Mbps respectively), with one downstream port; the hub is not supported
- USB 2.0 OTG dual-role device (DRD), with Host Negotiation Protocol (HNP) and Session Request Protocol (SRP) support

6.3 Block Diagram

Figure 6-1 shows the top-level block diagram of the FX3 USB subsystem.

Figure 6-1. EZ-USB FX3 USB Subsystem



6.4 Overview

6.4.1 USB Interface Block

The FX3 USB Interface Block (UIB) includes the following components:

6.4.2 USB 3.0 Function Controller

The USB 3.0 function controller implements both the link and protocol layers of the USB 3.0 specification.

6.4.3 USB 2.0 Function Controller

The USB 2.0 function controller implements the protocol layer of the USB 2.0 specification with an integrated serial interface engine (SIE) and a token processor (TP).

6.4.4 USB 2.0 Embedded Host

The FX3 USB 2.0 embedded host is simpler than a full-featured PC-based controller. Embedded USB hosts are defined to support a limited peripheral list and to operate with limited memory (compared to a PC). In essence, the host controller functions like a device controller, with added scheduling capability to control and initiate data traffic on the bus. The USB 2.0 embedded host includes the following features:

- EHCI-like interface in high-speed mode (EHCI is the PC-based Enhanced Host Controller Interface)

- OHCI-like interface in full- and low-speed modes (OHCI is the PC-based Open Host Controller Interface)
- Support for point-to-point communications with one downstream port
- Performance of all transaction scheduling in hardware
- DMA adapter interface common to other FX3 peripherals
- Fixed, one-to-one unidirectional endpoint to DMA socket mapping
- Shared hardware endpoint managers (EPMs) among USB 2.0 device, USB 2.0 host, and USB 3.0 device controllers

6.4.5 USB OTG Controller

The FX3 USB subsystem is capable of supporting a USB 2.0 host, USB 2.0 peripheral, and a USB 3.0 peripheral. The FX3 OTG controller has global control of these functions. Dynamic role swapping is limited to USB 2.0 only by enabling the appropriate USB 2.0 host or peripheral controller. The necessary control interface of the OTG controller facilitates:

- Global control of the embedded host, and the USB 2.0 device function
- Session Request Protocol (SRP) support per the OTG 2.0 specification
- Host Negotiation Protocol (HNP) support per the OTG 2.0 specification

6.4.6 Charger Detect Controller

The FX3 charger detect controller provides a control interface to the USB Accessory Charger Adapter supporting:

- EZ-Detect USB Battery Charging Revision 1.1
- Selectable standard ACA mode or Motorola Enhanced Mini-USB mode
- ID pin resistance value decoding
- Car-kit pass through UART functionality

6.4.7 End-Point Memory

The end-point memory (EPM) supports data transfers through the USB 2.0 host controller, USB 2.0 function controller, and USB 3.0 function controller blocks. It also supports the USB 3.0 bulk stream protocol. Two EPM units are available, dedicated to each data direction.

6.4.8 DMA Adapters

The UIB has two dedicated DMA adapters that manage all DMA data flow in and out of the UIB block, one for each direction. These DMA adapters are shared among the USB 3.0 device, USB 2.0 device, and USB 2.0 host controllers. Endpoint to DMA socket mapping is fixed and unidirectional; that is, ingress endpoints 0 to 16 are mapped to UIB ingress DMA sockets 0 to 16, and egress endpoints 0 to 16 are mapped to UIB DMA sockets 0 to 16. Hence the terms "socket" and "endpoint" are interchangeable within the USB block. The DMA adapter is identical to those within other FX3 peripherals. Refer to the DMA Subsystem chapter of this TRM to learn how the FX3 DMA works.

6.4.9 USB I/O System

6.4.9.1 USB 2.0 OTG PHY

Within USB 2.0 subsystems, FX3 has a USB 2.0 transceiver with a UTMI+ interface to the back-end, multiplexed between the USB 2.0 function and USB 2.0 embedded host controllers. It contains the required transceiver and OTG functionality, including:

- Standard four-wire signaling (VBUS, D+, D-, GND)
- USB 2.0 High-/Full-/Low-Speed data transmission rate
- USB 2.0 test modes fully supported
- VBUS sensing for connection detection
- Sampling of the USB_ID input for detection of A-device or B-device connection

- Charging and discharging of DP line for starting a session as B-device

6.4.9.2 USB 3.0 PHY

For USB 3.0, FX3 has a separate, dedicated USB 3.0 transceiver with a PIPE 3-compliant interface directly connected to the back-end USB 3.0 function controller. It includes these features:

- Dedicated, dual-simplex differential pairs for data transmit (SSTX+/-) and receive (SSRX+/-)
- Sideband functionality (such as reset, wake) with Low Frequency Periodic Signaling (LFPS)
- USB hot plug with receiver termination for connect/disconnect detection
- 5-Gbps SuperSpeed data transmission rate over 3-meter USB 3.0 cable
- Integrated transmitter, receiver, spread-spectrum clock (SSC) generation, PLL and ESD protection
- Full support for USB 3.0 test modes

6.5 UIB Top-Level Register Interface

FX3 features a common top-level register interface shared among all UIB functional blocks, as shown in the following code. Some functional blocks may also have their own specific register interface, which is described in their respective sections.

```
/* FX3 UIB Top Level Register Interface */

#define UIB_BASE_ADDR                           (0xe0030000)

typedef struct
{
    uvint32_t intr;                              /* 0xe0030000 */
    uvint32_t intr_mask;                         /* 0xe0030004 */
    uvint32_t rsrvd0[1024];
    uvint32_t phy_clk_and_test;                  /* 0xe0031008 */
    uvint32_t reserved[2];
    uvint32_t phy_chirp;                         /* 0xe0031014 */
    uvint32_t rsrvd1[250];
    uvint32_t dev_cs;                            /* 0xe0031400 */
    uvint32_t dev_framecnt;                      /* 0xe0031404 */
    uvint32_t dev_pwr_cs;                        /* 0xe0031408 */
    uvint32_t dev_setupdat0;                     /* 0xe003140c */
    uvint32_t dev_setupdat1;                     /* 0xe0031410 */
    uvint32_t dev_toggle;                        /* 0xe0031414 */
    uvint32_t dev_epi_cs[16];                    /* 0xe0031418 */
    uvint32_t dev_epi_xfer_cnt[16];              /* 0xe0031458 */
    uvint32_t dev_epo_cs[16];                    /* 0xe0031498 */
    uvint32_t dev_epo_xfer_cnt[16];              /* 0xe00314d8 */
    uvint32_t dev_ctl_intr_mask;                 /* 0xe0031518 */
    uvint32_t dev_ctl_intr;                      /* 0xe003151c */
    uvint32_t dev_ep_intr_mask;                  /* 0xe0031520 */
    uvint32_t dev_ep_intr;                       /* 0xe0031524 */
    uvint32_t rsrvd2[182];
    uvint32_t chgdet_ctrl;                       /* 0xe0031800 */
    uvint32_t chgdet_intr;                       /* 0xe0031804 */
    uvint32_t chgdet_intr_mask;                  /* 0xe0031808 */
    uvint32_t otg_ctrl;                          /* 0xe003180c */
    uvint32_t otg_intr;                          /* 0xe0031810 */
    uvint32_t otg_intr_mask;                     /* 0xe0031814 */
    uvint32_t otg_timer;                         /* 0xe0031818 */
    uvint32_t rsrvd3[249];
    uvint32_t eepm_cs;                           /* 0xe0031c00 */
```

```
uvint32_t iepm_cs;                          /* 0xe0031c04 */
uvint32_t iepm_mult;                        /* 0xe0031c08 */
uvint32_t rsrvd4[13];
uvint32_t eepm_endpoint[16];                /* 0xe0031c40 */
uvint32_t iepm_endpoint[16];                /* 0xe0031c80 */
uvint32_t iepm_fifo;                        /* 0xe0031cc0 */
uvint32_t rsrvd5[207];
uvint32_t host_cs;                          /* 0xe0032000 */
uvint32_t host_ep_intr;                     /* 0xe0032004 */
uvint32_t host_ep_intr_mask;                /* 0xe0032008 */
uvint32_t host_toggle;                      /* 0xe003200c */
uvint32_t host_shdl_cs;                     /* 0xe0032010 */
uvint32_t host_shdl_sleep;                  /* 0xe0032014 */
uvint32_t host_resp_base;                   /* 0xe0032018 */
uvint32_t host_resp_cs;                     /* 0xe003201c */
uvint32_t host_active_ep;                   /* 0xe0032020 */
uvint32_t ohci_revision;                    /* 0xe0032024 */
uvint32_t ohci_control;                     /* 0xe0032028 */
uvint32_t ohci_command_status;              /* 0xe003202c */
uvint32_t ohci_interrupt_status;            /* 0xe0032030 */
uvint32_t ohci_interrupt_enable;            /* 0xe0032034 */
uvint32_t ohci_interrupt_disable;           /* 0xe0032038 */
uvint32_t ohci_fm_interval;                 /* 0xe003203c */
uvint32_t ohci_fm_remaining;                /* 0xe0032040 */
uvint32_t ohci_fm_number;                   /* 0xe0032044 */
uvint32_t ohci_periodic_start;              /* 0xe0032048 */
uvint32_t ohci_ls_threshold;                /* 0xe003204c */
uvint32_t reserved1;
uvint32_t ohci_rh_port_status;              /* 0xe0032054 */
uvint32_t ohci_eof;                         /* 0xe0032058 */
uvint32_t ehci_hccparams;                   /* 0xe003205c */
uvint32_t ehci_usbcmd;                      /* 0xe0032060 */
uvint32_t ehci_usbsts;                      /* 0xe0032064 */
uvint32_t ehci_usbintr;                     /* 0xe0032068 */
uvint32_t ehci_frindex;                     /* 0xe003206c */
uvint32_t ehci_configflag;                  /* 0xe0032070 */
uvint32_t ehci_portsc;                      /* 0xe0032074 */
uvint32_t ehci_eof;                         /* 0xe0032078 */
uvint32_t shdl_chng_type;                   /* 0xe003207c */
uvint32_t shdl_state_machine;               /* 0xe0032080 */
uvint32_t shdl_internal_status;             /* 0xe0032084 */
uvint32_t rsrvd6[222];
struct
{
    uvint32_t shdl_ohci0;                   /* 0xe0032400 */
    uvint32_t shdl_ohci1;                   /* 0xe0032404 */
    uvint32_t shdl_ohci2;                   /* 0xe0032408 */
} ohci_shdl[64];
uvint32_t rsrvd7[64];
struct
{
    uvint32_t shdl_ehci0;                   /* 0xe0032800 */
    uvint32_t shdl_ehci1;                   /* 0xe0032804 */
    uvint32_t shdl_ehci2;                   /* 0xe0032808 */
} ehci_shdl[64];
uvint32_t rsrvd8[5376];
uvint32_t id;                               /* 0xe0037f00 */
uvint32_t power;                            /* 0xe0037f04 */
```

```
    uvint32_t rsrvd9[62];
    struct
    {
        uvint32_t dscr;                         /* 0xe0038000 */
        uvint32_t size;                         /* 0xe0038004 */
        uvint32_t count;                        /* 0xe0038008 */
        uvint32_t status;                       /* 0xe003800c */
        uvint32_t intr;                         /* 0xe0038010 */
        uvint32_t intr_mask;                    /* 0xe0038014 */
        uvint32_t rsrvd10[2];
        uvint32_t dscr_buffer;                  /* 0xe0038020 */
        uvint32_t dscr_sync;                    /* 0xe0038024 */
        uvint32_t dscr_chain;                   /* 0xe0038028 */
        uvint32_t dscr_size;                    /* 0xe003802c */
        uvint32_t rsrvd11[19];
        uvint32_t event;                        /* 0xe003807c */
    } sck[16];
    uvint32_t rsrvd12[7616];
    uvint32_t sck_intr0;                        /* 0xe003ff00 */
    uvint32_t sck_intr1;                        /* 0xe003ff04 */
    uvint32_t sck_intr2;                        /* 0xe003ff08 */
    uvint32_t sck_intr3;                        /* 0xe003ff0c */
    uvint32_t sck_intr4;                        /* 0xe003ff10 */
    uvint32_t sck_intr5;                        /* 0xe003ff14 */
    uvint32_t sck_intr6;                        /* 0xe003ff18 */
    uvint32_t sck_intr7;                        /* 0xe003ff1c */
    uvint32_t rsrvd13[56];
} UIB_REGS_T, *PUIB_REGS_T;

#define UIB        ((PUIB_REGS_T) UIB_BASE_ADDR)
```

6.6 USB Function Controllers

6.6.1 USB 3.0 Function

6.6.1.1 Clocking

There are five independent clock domains in the UIB block supporting the USB 3.0 function, as listed in Table 6-1.

Table 6-1. USB 3.0 Function Clocks

Domain	Typ Freq	Configuration Source	Description
dma_bus_clk_i	100 MHz	GCTL_CPU_CLK_CFG.DMA_DIV	DMA access clock
mmio_bus_clk_i	100 MHz	GCTL_CPU_CLK_CFG.MMIO_DIV	MMIO register access clock
uib_ssc_i	125 MHz	Driven from USB 3.0 PHY	USB 3.0 spread-spectrum clock
standby_clk_i	32 kHz	Always-on	Always-on clock for low-power modes
uib_ref_clk_i	19.2/26/38.4/52 MHz	External input clock	USB3.0 PHY reference clock

In addition, the EPM uses a clock, uib_epm_clk_i, that is 125 MHz when the USB 3.0 function controller is active. The uib_epm_clk_i configuration source is from GCTL_UIB_CORE_CLK.EPMCLK_SRC and enabled by GCTL_UIB_CORE_CLK.CLK_EN.

6.6.1.2 Interrupt Requests

The UIB block has three global interrupt sources to the VIC, listed in Table 6-2, which are shared among USB 3.0, USB 2.0, and OTG controllers.

Table 6-2. USB Global Interrupt Sources to VIC

Interrupt	VIC Line	Interrupt Source	Description
usbdma_int	8	UIB DMA sockets	UIB DMA interrupt
usbcore_int	9	USB 3.0 function, USB 2.0 function, USB 2.0 host, USB 2.0 OTG, charger detect, EPM	UIB core interrupt
usbep0_int	10	USB 3.0 device or USB 2.0 device	EP0 interrupt that is used only in device mode

UIB has a global interrupt register, UIB_INTR, which contains interrupt sources from the respective functional blocks (USB 3.0 function, USB 2.0 function, USB 2.0 host, USB 2.0 OTG, charger detect, EPM). The UIB core interrupt to VIC is the logical OR of interrupt sources in UIB_INTR.

The USB 3.0 link layer interrupts are located in UIB_LNK_INTR. UIB_INTR.LNK_INT is the logical OR of the interrupt sources in UIB_LNK_INTR.

USB 3.0 protocol layer interrupts are located in UIB_PROT_INTR. UIB_INTR.PROT_INT is the logical OR of the interrupt sources in UIB_PROT_INTR.

USB 3.0 function endpoint interrupts are located in UIB_PROT_EP_INTR. UIB_INTR.PROT_EP_INT is the logical OR of the interrupt sources in UIB_PROT_EP_INTR.

6.6.1.3 USB 3.0 Functional Description

The SuperSpeed bus is a layered communications architecture that comprises the following elements:

SuperSpeed interconnect: The SuperSpeed interconnect is the manner in which devices are connected to and communicate with the host over the SuperSpeed bus. It includes the topology of devices connected to the bus, the communication layers, the relationships between them, and how they interact to accomplish information exchanges between the host and devices.

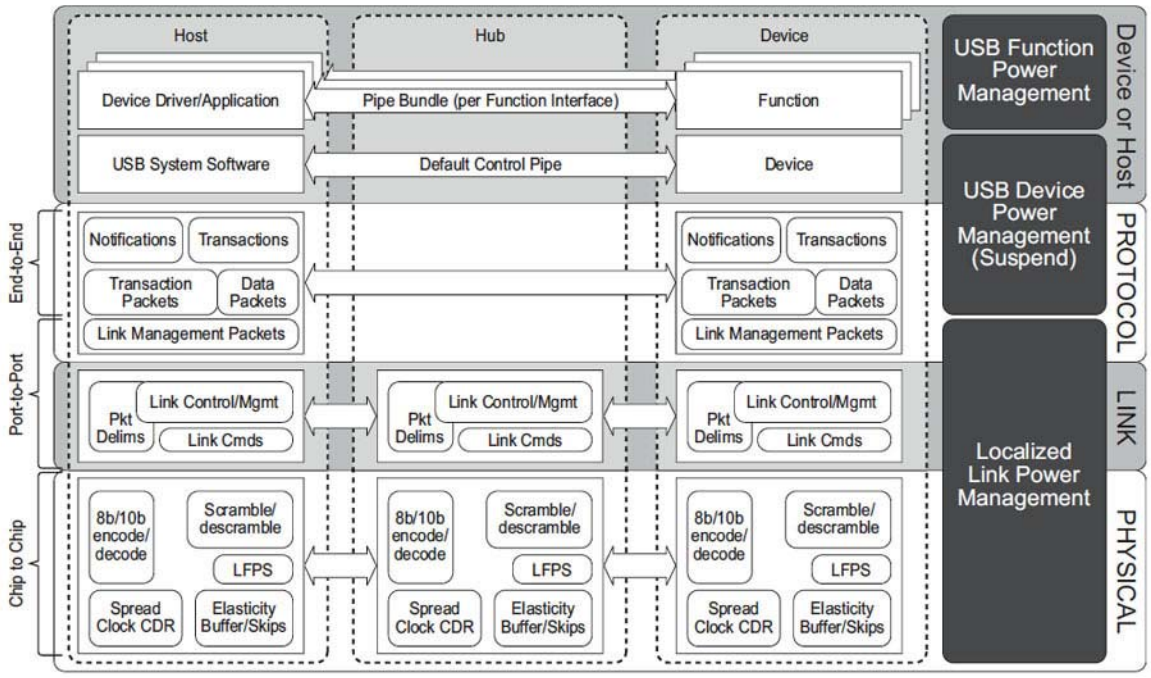
Devices: Devices implement the required function end of SuperSpeed communication layers to provide a specific function of the application, for example, a mass storage device. The terms "USB device" and "USB function" are interchangeable.

Host: The host implements the required host end of SuperSpeed communication layers to use the functions of the attached devices. It owns the SuperSpeed data activity schedule and management of the SuperSpeed bus and all devices connected to it.

As shown in Figure 6-2, the rows (device or host, protocol, link, physical) represent the communication layers of the SuperSpeed interconnect, namely:

- Physical (PHY) layer
- Link layer
- Protocol layer
- The FX3 USB 3.0 function controller design follows the same basic SuperSpeed architecture.

Figure 6-2. SuperSpeed (USB 3.0) Architecture



6.6.2 Physical Layer

The physical layer defines the PHY portion of the port and the physical connection between a downstream facing port (on a host or hub) and an upstream facing port (on a device). The FX3 USB 3.0 function physical connection comprises two differential data pairs, one transmit path and one receive path. The nominal signaling data rate is 5 Gbps. The electrical aspects of each path are characterized as a transmitter, channel, and receiver; these collectively represent a unidirectional differential link. Each differential link is AC-coupled with capacitors located on the transmitter side of the differential link. The channel includes the electrical characteristics of the cables and connectors.

At an electrical level, each differential link is initialized by enabling its receiver termination. The transmitter is responsible for detecting the far-end receiver termination as an indication of a bus connection and informing the link layer so the connect status can be factored into link operation and management. When receiver termination is present but no signaling is occurring on the differential link, it is considered to be in the electrical idle state. In this state, LFPS is used to signal initialization and power management information. The LFPS is relatively simple to generate and detect and uses very little power.

FX3 USB 3.0 PHY has its own clock domain with Spread Spectrum Clock (SSC) modulation. The USB 3.0 cable does not include a reference clock, so the clock domains on each end of the physical connection are not directly connected. Bit-level timing synchronization relies on the local receiver aligning its bit recovery clock to the remote transmitter's clock by phase-locking to the signal transitions in the received bit stream. The receiver needs enough transitions to reliably recover clock and data from the bit stream. To assure that adequate transitions occur in the bit stream independent of the data content being transmitted, the transmitter encodes data and control characters into symbols using an 8b/10b code. Control symbols are used to achieve byte alignment and are used for framing data and managing the link. Special characteristics make control symbols uniquely identifiable from data symbols.

The signal (timing, jitter tolerance, and so on) and electrical (DC characteristics, channel capacitance, and so on.) performance of SuperSpeed links is defined with compliance requirements specified in terms of transmit and receive signaling eye diagrams. The FX3 USB 3.0 function physical layer receives 8-bit data from the link layer and scrambles the data to reduce EMI emissions. It then encodes the scrambled 8-bit data into 10-bit symbols for transmission over the physical connection. The resultant data is sent at a rate that includes spread spectrum to further lower the EMI emissions. The bit stream is recovered from the differential link by the receiver, assembled into 10-bit symbols, and decoded and descrambled, producing 8-bit data that is then sent to the link layer for further processing.

FX3 PHY carries out the physical layer control settings and operations via the PHY register interface, which is combined with the link layer register interface for easy firmware implementation.

6.6.3 Link Layer

A SuperSpeed link is a logical and physical connection of two ports. The connected ports are called "link partners." A port has a physical part and a logical part. The FX3 USB 3.0 function link layer defines the logical portion of a port and the communications between link partners.

The logical portion of a port contains:

- State machines for managing its end of the physical connection. These include physical layer initialization and event management, that is, connect, removal, and power management.
- State machines and buffering for managing information exchanges with the link partner. It implements protocols for flow control, reliable delivery (port to port) of packet headers, and link power management.
- Buffering for data and protocol layer information elements.

The logical portion of a port does the following:

- Provides the correct framing of sequences of bytes into packets during transmission; for example, insertion of packet delimiters
- Detects received packets, including packet delimiters and error checks of received header packets (for reliable delivery)
- Provides an appropriate interface to the protocol layer for pass-through of protocol-layer packet information exchanges

The FX3 USB 3.0 function physical layer provides the logical port with an interface through which it can do the following:

- Manage the state of its PHY (that is, its end of the physical connection), including power management and events (connection, removal, and wake).
- Transmit and receive byte streams, with additional signals that qualify the byte stream as control sequences or data. The physical layer includes discrete transmit and receive physical links; therefore, a port can simultaneously transmit and receive control and data information.

The FX3 USB 3.0 function controller has a unified register interface for physical and link layers as in the following code. The register level programming is managed by the USB driver block in the FX3 SDK.

```
/* FX3 USB 3.0 Function Physical Layer and Link Layer Register Interface */

#define USB3LNK_BASE_ADDR                          (0xe0033000)

typedef struct
{
    uvint32_t lnk_conf;                             /* 0xe0033000 */
    uvint32_t lnk_intr;                             /* 0xe0033004 */
    uvint32_t lnk_intr_mask;                        /* 0xe0033008 */
    uvint32_t lnk_error_conf;                       /* 0xe003300c */
    uvint32_t lnk_error_status;                     /* 0xe0033010 */
    uvint32_t lnk_error_count;                      /* 0xe0033014 */
    uvint32_t lnk_error_count_threshold;            /* 0xe0033018 */
    uvint32_t lnk_phy_conf;                         /* 0xe003301c */
    uvint32_t reserved0[3];
    uvint32_t lnk_phy_mpll_status;                  /* 0xe003302c */
    uvint32_t reserved1[3];
    uvint32_t lnk_phy_tx_trim;                      /* 0xe003303c */
    uvint32_t lnk_phy_error_conf;                   /* 0xe0033040 */
    uvint32_t lnk_phy_error_status;                 /* 0xe0033044 */
    uvint32_t reserved2[2];
    uvint32_t lnk_device_power_control;             /* 0xe0033050 */
    uvint32_t lnk_ltssm_state;                      /* 0xe0033054 */
```

```
    uvint32_t reserved3[3];
    uvint32_t lnk_lfps_observe;                 /* 0xe0033064 */
    uvint32_t reserved4[52];
    uvint32_t lnk_compliance_pattern_0;         /* 0xe0033138 */
    uvint32_t lnk_compliance_pattern_1;         /* 0xe003313c */
    uvint32_t lnk_compliance_pattern_2;         /* 0xe0033140 */
    uvint32_t lnk_compliance_pattern_3;         /* 0xe0033144 */
    uvint32_t lnk_compliance_pattern_4;         /* 0xe0033148 */
    uvint32_t lnk_compliance_pattern_5;         /* 0xe003314c */
    uvint32_t lnk_compliance_pattern_6;         /* 0xe0033150 */
    uvint32_t lnk_compliance_pattern_7;         /* 0xe0033154 */
    uvint32_t lnk_compliance_pattern_8;         /* 0xe0033158 */
} USB3LNK_REGS_T, *PUSB3LNK_REGS_T;

#define USB3LNK        ((PUSB3LNK_REGS_T) USB3LNK_BASE_ADDR)
```

6.6.4 Protocol Layer

The FX3 USB 3.0 function protocol layer defines the "end-to-end" communication rules between a host and device. The SuperSpeed protocol provides for application data information exchanges between a host and a device endpoint. This communications relationship is called a "pipe." It is a host-directed protocol, which means the host determines when application data is transferred between the host and the device. SuperSpeed is not a polled protocol, as a device can asynchronously request service from the host on behalf of a particular endpoint.

All protocol layer communications are accomplished via the exchange of packets. Packets are sequences of data bytes with specific control sequences that serve as delimiters managed by the link layer. Unlike USB 2.0 which uses a broadcast mechanism, host-transmitted protocol packets are routed through intervening hubs directly to a peripheral device. A peripheral device considers itself that targeted by any protocol layer packet it receives. Device transmitted protocol packets simply flow upstream through hubs to the host.

Packet headers are the building blocks of the protocol layer. They are fixed-size packets with type and subtype field encodings for specific purposes. A small record within a packet header is utilized by the link layer (port to port) to manage the flow of the packet from port to port. Packet headers are delivered through the link layer (port to port) reliably. The remaining fields are utilized by the end-to-end protocol.

Application data is transmitted within data packet payloads, which are preceded (in the protocol) by specifically encoded data packet headers. Data packet payloads are not delivered reliably through the link layer (however, the accompanying data packet headers are delivered reliably). The protocol layer supports the reliable delivery of data packets via explicit acknowledgement (header) packets and the retransmission of lost or corrupt data. Not all data information exchanges utilize data acknowledgements.

Data may be transmitted in bursts of back-to-back sequences of data packets (depending on the scheduling by the host). The protocol allows efficient bus utilization by concurrently transmitting and receiving over the link. For example, a transmitter (host or device) can burst multiple packets of data back to back, while the receiver can transmit data acknowledgements without interrupting the burst of data packets. The number of data packets in a specific burst is scheduled by the host. Furthermore, a host may simultaneously schedule multiple OUT bursts to be active at the same time as an IN burst.

The protocol provides flow control support for some transfer types. A device-initiated flow control is signaled by a device via a defined protocol packet. A host-initiated flow control event is realized via the host schedule (host will simply not schedule information flows for a pipe unless it has data or buffering available). On receipt of a flow control event, the host removes the pipe from its schedule. Resumption of scheduling information flows for a pipe may be initiated by the host or device. A device endpoint notifies a host of its readiness (to source or sink data) via an asynchronously transmitted "ready" packet. On receipt of the "ready" notification, the host adds the pipe to its schedule, assuming that it still has data or buffering available. Independent information streams can be explicitly delineated and multiplexed on the bulk transfer type. This means that through a single pipe instance, more than one data stream can be tagged by the source and identified by the sink. The protocol provides for the device to direct which data stream is active on the pipe.

Devices may asynchronously transmit notifications to the host. These notifications are used to convey a change in the device state. A host transmits a special packet header to the bus that includes the host's timestamp. The value in this packet is used to keep devices in synchronization with the host. In contrast to other packet types, the timestamp packet is forwarded down all paths not in a low-power state. The timestamp packet transmission is scheduled by the host at a specification-determined period.

The FX3 USB 3.0 function controller has a dedicated register interface for the protocol layer as shown in the following code. The register level programming and the interrupt handling is performed by the USB driver in the FX3 SDK.

```
/* FX3 USB 3.0 Function Protocol Layer Register Interface */

#define USB3PROT_BASE_ADDR                        (0xe0033400)

typedef struct
{
    uvint32_t prot_cs;                             /* 0xe0033400 */
    uvint32_t prot_intr;                           /* 0xe0033404 */
    uvint32_t prot_intr_mask;                      /* 0xe0033408 */
    uvint32_t reserved0[3];
    uvint32_t prot_lmp_port_capability_timer;      /* 0xe0033418 */
    uvint32_t prot_lmp_port_configuration_timer;   /* 0xe003341c */
    uvint32_t reserved1[2];
    uvint32_t prot_framecnt;                       /* 0xe0033428 */
    uvint32_t reserved2;
    uvint32_t prot_itp_time;                       /* 0xe0033430 */
    uvint32_t prot_itp_timestamp;                  /* 0xe0033434 */
    uvint32_t prot_setupdat0;                      /* 0xe0033438 */
    uvint32_t prot_setupdat1;                      /* 0xe003343c */
    uvint32_t prot_seq_num;                        /* 0xe0033440 */
    uvint32_t reserved3[6];
    uvint32_t prot_lmp_received;                   /* 0xe003345c */
    uvint32_t reserved4[5];
    uvint32_t prot_ep_intr;                        /* 0xe0033474 */
    uvint32_t prot_ep_intr_mask;                   /* 0xe0033478 */
    uvint32_t rsrvd0[33];
    uvint32_t prot_epi_cs1[16];                    /* 0xe0033500 */
    uvint32_t prot_epi_cs2[16];                    /* 0xe0033540 */
    uvint32_t prot_epi_unmapped_stream[16];        /* 0xe0033580 */
    uvint32_t prot_epi_mapped_stream[16];          /* 0xe00335c0 */
    uvint32_t prot_epo_cs1[16];                    /* 0xe0033600 */
    uvint32_t prot_epo_cs2[16];                    /* 0xe0033640 */
    uvint32_t prot_epo_unmapped_stream[16];        /* 0xe0033680 */
    uvint32_t prot_epo_mapped_stream[16];          /* 0xe00336c0 */
    uvint32_t prot_stream_error_disable;           /* 0xe0033700 */
    uvint32_t prot_stream_error_status;            /* 0xe0033704 */
} USB3PROT_REGS_T, *PUSB3PROT_REGS_T;

#define USB3PROT        ((PUSB3PROT_REGS_T) USB3PROT_BASE_ADDR)
```

6.7 USB 2.0 Function

6.7.1 Clocking

There are five independent clock domains in the UIB block supporting the USB 2.0 function, as listed in Table 6-3.

Table 6-3. USB 2.0 Function Clocks

Domain	Typ Freq	Configuration Source	Description
dma_bus_clk_i	100 MHz	GCTL_CPU_CLK_CFG.DMA_DIV	DMA access clock
mmio_bus_clk_i	100 MHz	GCTL_CPU_CLK_CFG.MMIO_DIV	MMIO register access clock
uib_sieclock_i	30 MHz	Driven from USB 2.0 OTG PHY	USB 2.0 serial interface engine clock
standby_clk_i	32 kHz	Always-on	Always-on clock for low-power modes
uib_ref_clk_i	19.2/26/38.4/52 MHz	External input clock	USB 2.0 PHY reference clock

In addition, the EPM uses a clock uib_epm_clk_i that is 100 MHz when the USB 2.0 function controller is active. The uib_epm_clk_i configuration source is from GCTL_UIB_CORE_CLK.EPMCLK_SRC and enabled by GCTL_UIB_CORE_CLK.CLK_EN.

6.7.2 Interrupt Requests

The UIB block has three global interrupt sources to the VIC, listed in Table 6-2, which are shared among USB 3.0, USB 2.0 and OTG controllers.

UIB has a global interrupt register, UIB_INTR, which contains interrupt sources from the respective functional blocks (USB 3.0 function, USB 2.0 function, USB 2.0 host, USB 2.0 OTG, charger detect, EPM). The UIB core interrupt to VIC is the logical OR of interrupt sources in UIB_INTR.

USB 2.0 function controller interrupts are located in UIB_DEV_CTL_INTR. UIB_INTR.DEV_CTL_INT is the logical OR of the interrupt sources in UIB_DEV_CTL_INTR.

USB 2.0 function endpoint interrupts are located in UIB_DEV_EP_INTR. UIB_INTR.DEV_EP_INT is the logical OR of the interrupt sources in UIB_DEV_EP_INTR.

6.7.3 USB 2.0 Functional Description

The USB 2.0 function controller hardware includes an a Serial Interface Engine (SIE) and Token Processor (TP). It does the following:

- Handles the handshake between the endpoint and the host device
- Generates an interrupt when valid data packets are received
- Generates an interrupt when an error in transmission occurs
- Moves valid data to/from the endpoint
- Handles all the bit stuffing required

6.7.3.1 Serial Interface Engine

The SIE is responsible for handling the USB traffic at the byte-level and for detecting the suspend, reset, and resume USB bus states. It parses the traffic, decoding the types of packets that have been received and translating the packet types into bytes for transmission back to the host on the USB bus.

6.7.3.2 Token Processor

The TP handles most of the protocol described in chapter 8 of the USB specification. It receives the USB basic protocol commands from the host and generates the appropriate sequence of responses by synchronizing the frame timer, receiving/

transmitting data, or receiving/transmitting handshake responses to the commands themselves. It does all of this based on the information provided by the SIE, which is responsible for decoding the bytes into those commands.

6.7.4 USB 2.0 Function Registers

The FX3 USB 2.0 function registers can be accessed directly from the UIB top-level register interface. These registers are shown below and are programmed by the USB driver in the FX3 SDK.

```
/* FX3 USB 2.0 Function Register Interface */
/* These definitions extracted from the Top Level UIB register interface */

    uvint32_t phy_clk_and_test;                 /* 0xe0031008 */
    uvint32_t phy_chirp;                        /* 0xe0031014 */
    uvint32_t dev_cs;                           /* 0xe0031400 */
    uvint32_t dev_framecnt;                     /* 0xe0031404 */
    uvint32_t dev_pwr_cs;                       /* 0xe0031408 */
    uvint32_t dev_setupdat0;                    /* 0xe003140c */
    uvint32_t dev_setupdat1;                    /* 0xe0031410 */
    uvint32_t dev_toggle;                       /* 0xe0031414 */
    uvint32_t dev_epi_cs[16];                   /* 0xe0031418 */
    uvint32_t dev_epi_xfer_cnt[16];             /* 0xe0031458 */
    uvint32_t dev_epo_cs[16];                   /* 0xe0031498 */
    uvint32_t dev_epo_xfer_cnt[16];             /* 0xe00314d8 */
    uvint32_t dev_ctl_intr_mask;                /* 0xe0031518 */
    uvint32_t dev_ctl_intr;                     /* 0xe003151c */
    uvint32_t dev_ep_intr_mask;                 /* 0xe0031520 */
    uvint32_t dev_ep_intr;                      /* 0xe0031524 */
```

6.7.5 USB Reset

USB 2.0 reset is detected by the SIE and is reported to the TP. The URESET bit in the DEV_CTL_INTR register is set, and the corresponding interrupt is generated (if not masked).

6.7.6 USB Suspend

USB 2.0 suspend is detected by the SIE and is reported to the TP. The SUSP bit in the DEV_CTL_INTR register is set, and the corresponding interrupt is generated (if not masked).

6.7.7 USB Resume

USB 2.0 resume is detected by the SIE and is reported to the TP. The SUSP bit in the DEV_CTL_INTR register is cleared, and the corresponding interrupt is generated (if not masked). A resume by the device is generated by the firmware, setting the SIGRSUME bit in the PWR_CS register. The firmware must also clear this bit to end the resume signaling.

6.7.8 Start of Frame

Start of frame (SOF) timer packets are received by the SIE and reported to the TP. The SOF bit in the UIB_DEV_CTL_INTR register is set, and the corresponding interrupt is generated (if not masked). The frame number is stored in the FRAMECNT register. The SIE is capable to generating synthetic SOF notifications to replace any SOF packets that get lost. The feature can be controlled using the NOSYNSOF bit in the DEV_PWR_CS register.

6.7.9 SETUP Packet

SETUP packets (to endpoint 0) are received by the SIE and reported to the TP. The SETUP data is stored in registers DEV_SETUPDAT0 and DEV_SETUPDAT0 1. The SUDAV bit in the DEV_CTL_INTR register is set, as is the SUTOK bit if the

packet is received correctly, and their corresponding interrupts are generated if not masked. Upon receipt of a SetAddress command from the host, the DEV_CS register field DEVICEADDR is updated with the new device address.

6.7.10 IN Packet

IN packets are received by the SIE and reported to the TP, which generates the appropriate responses to the SIE and updates the corresponding DEV_EPO_CS register for the endpoint to which the packet was sent.

6.7.11 OUT Packet

OUT packets are received by the SIE and reported to the TP, which generates the appropriate responses to the SIE and updates the corresponding DEV_EPO_CS register for the endpoint to which the packet was sent.

6.8 USB 3.0 and USB 2.0 Function Coordination

When the FX3 is functioning as a USB device, the USB 3.0 PHY or the USB 2.0 PHY needs to be turned on based on the capabilities of the USB host to which FX3 is connected. The USB 3.0 specification requires that only one of the PHY layers be operational at most times during the USB device operation. The exception to this rule is a small time window in which the device is attempting to move from USB 2.0 mode to USB 3.0 mode. The firmware application on FX3 is responsible for identifying the host capabilities and setting the USB connection accordingly. The following procedure should be used by the FX3 firmware for USB connection negotiation:

1. Wait for a valid VBus voltage (GCTL_IOPWR interrupt).
2. Turn on the USB 3.0 PHY to start 3.0 receiver detection.
   a. If receiver detection succeeds, the LNK_LTSSM_CONNECT interrupt will be received. If this interrupt is received, the device will proceed with enumeration in USB 3.0 mode.
3. If receiver detection fails, the LNK_LTSSM_DISCONNECT interrupt will be received. If this interrupt is received:
   a. Turn off USB 3.0 PHY and turn on USB 2.0 PHY.
   b. A USB 2.0 bus reset will be received as part of USB 2.0 connection startup.
   c. The 3.0 PHY should be re-enabled on receiving the URESET interrupt that is triggered on a 2.0 bus reset. Both the 2.0 and 3.0 PHYs will be active at this time.
   d. If the 3.0 receiver detection succeeds (LNK_LTSSM_CONNECT):
      i.Turn off the USB 2.0 PHY.
      ii.Proceed with enumeration as a USB 3.0 device.
   e. If the 3.0 receiver detection fails (LNK_LTSSM_DISCONNECT):
      i.Turn off the USB 3.0 PHY.
      ii.Check number of times that 3.0 receiver detection has failed. If this count is greater than 3:
4. Proceed with enumeration as a USB 2.0 device.
5. There is no need to attempt 3.0 enumeration on any further bus resets.

Any PHYs that are enabled need to be disabled when the VBus voltage is removed. The entire previous procedure needs to be repeated when valid VBus is detected again.

Note that USB 3.0 PHY on the FX3 needs to be turned off when VBus is removed or a host disconnect is discovered by other means. If the 3.0 PHY is left turned on, the 3.0 link startup is liable to fail when connected again to the host.

6.9 USB Function Programming Model

6.9.1 USB 3.0 Initialization

Before USB 3.0 is operational, the function controller needs to be initialized. The following code example from the FX3 SDK implements the UIB initialization sequence for the USB 3.0 function.

```
static void
CyU3PUibInit (
        void)
{
    uint8_t ep = 0;

    /* Enable the Power regulators*/
    GCTLAON->wakeup_en       = 0;
    GCTLAON->wakeup_polarity = 0;

    /* Link_phy_conf enable Rx terminations */
    USB3LNK->lnk_phy_conf   = 0xE0000001;
    USB3LNK->lnk_error_conf = 0xFFFFFFFF;
    USB3LNK->lnk_intr       = 0xFFFFFFFF;
    USB3LNK->lnk_intr_mask  = CY_U3P_UIB_LGO_U3 | CY_U3P_UIB_LTSSM_CONNECT |
        CY_U3P_UIB_LTSSM_DISCONNECT | CY_U3P_UIB_LTSSM_RESET | CY_U3P_UIB_LTSSM_STATE_CHG;

    USB3PROT->prot_ep_intr_mask = 0;
    USB3PROT->prot_intr         = 0xFFFFFFFF;
    USB3PROT->prot_intr_mask    = (CY_U3P_UIB_STATUS_STAGE |
            CY_U3P_UIB_SUTOK_EN | CY_U3P_UIB_EP0_STALLED_EN |
            CY_U3P_UIB_TIMEOUT_PORT_CAP_EN | CY_U3P_UIB_TIMEOUT_PORT_CFG_EN |
CY_U3P_UIB_LMP_RCV_EN |
            CY_U3P_UIB_LMP_PORT_CAP_EN | CY_U3P_UIB_LMP_PORT_CFG_EN);


    /* Disable all EPs except EP0 */
    for (ep = 1; ep < 16; ep++)
    {
        UIB->dev_epo_cs[ep] &= ~CY_U3P_UIB_EPO_VALID;
        USB3PROT->prot_epo_cs1[ep] = 0;
        UIB->dev_epi_cs[ep] &= ~CY_U3P_UIB_EPI_VALID;
        USB3PROT->prot_epi_cs1[ep] = 0;
    }

    /* Disable all UIB interrupts at this stage. */
    UIB->intr_mask &= ~(CY_U3P_UIB_DEV_CTL_INT | CY_U3P_UIB_DEV_EP_INT | CY_U3P_UIB_LNK_INT
|
            CY_U3P_UIB_PROT_INT | CY_U3P_UIB_PROT_EP_INT | CY_U3P_UIB_EPM_URUN);

    /* Enable the Vbus detection interrupt at this stage. */
    GCTL->iopwr_intr      = 0xFFFFFFFF;
    GCTL->iopwr_intr_mask = CY_U3P_VBUS;
    CyU3PVicEnableInt (CY_U3P_VIC_GCTL_PWR_VECTOR);

}
```

6.9.2 USB 3.0 Enable

Once the USB 3.0 function controller is initialized, it needs to be enabled. The following code snippet from the FX3 SDK implements the sequence to enable the USB 3.0 function controller.

```
static void
CyU3PUsbEnableUsb3 (
        void)
{
    UIB->intr_mask &= ~(CY_U3P_UIB_DEV_CTL_INT | CY_U3P_UIB_DEV_EP_INT |
   CY_U3P_UIB_LNK_INT | CY_U3P_UIB_PROT_INT | CY_U3P_UIB_PROT_EP_INT | CY_U3P_UIB_EPM_URUN);

    /* Make sure that all relevant USB 3.0 interrupts are enabled. */
    USB3LNK->lnk_intr = 0xFFFFFFFF;
    USB3LNK->lnk_intr_mask = CY_U3P_UIB_LGO_U3 |
        CY_U3P_UIB_LTSSM_CONNECT | CY_U3P_UIB_LTSSM_DISCONNECT | CY_U3P_UIB_LTSSM_RESET |
CY_U3P_UIB_LTSSM_STATE_CHG;
    USB3PROT->prot_intr = 0xFFFFFFFF;
    USB3PROT->prot_intr_mask = (CY_U3P_UIB_STATUS_STAGE | CY_U3P_UIB_SUTOK_EN |
CY_U3P_UIB_EP0_STALLED_EN |
            CY_U3P_UIB_TIMEOUT_PORT_CAP_EN | CY_U3P_UIB_TIMEOUT_PORT_CFG_EN |
CY_U3P_UIB_LMP_RCV_EN |
            CY_U3P_UIB_LMP_PORT_CAP_EN | CY_U3P_UIB_LMP_PORT_CFG_EN);

    /* Set port config and capability timers to their initial values. */
    USB3PROT->prot_lmp_port_capability_timer    = CY_U3P_UIB_PROT_LMP_PORT_CAP_TIMER_VALUE;
    USB3PROT->prot_lmp_port_configuration_timer = CY_U3P_UIB_PROT_LMP_PORT_CFG_TIMER_VALUE;

    /* Turn on AUTO response to LGO_U3 command from host. */
    USB3LNK->lnk_compliance_pattern_8 |= CY_U3P_UIB_LFPS;
    USB3LNK->lnk_phy_conf = 0xE0000001;

    CyU3PSetUsbCoreClock (1, 0);
    CyU3PBusyWait (10);

    /* Force LTSSM into SS.Disabled state for 100us after the PHY is turned on. */
    USB3LNK->lnk_ltssm_state = (CY_U3P_UIB_LNK_STATE_SSDISABLED <<
CY_U3P_UIB_LTSSM_OVERRIDE_VALUE_POS) |
        CY_U3P_UIB_LTSSM_OVERRIDE_EN;
    UIB->otg_ctrl |= CY_U3P_UIB_SSDEV_ENABLE;
    CyU3PBusyWait (100);
    USB3LNK->lnk_ltssm_state &= ~CY_U3P_UIB_LTSSM_OVERRIDE_EN;

    USB3LNK->lnk_conf = (USB3LNK->lnk_conf & ~CY_U3P_UIB_EPM_FIRST_DELAY_MASK) |
        (12 << CY_U3P_UIB_EPM_FIRST_DELAY_POS) | CY_U3P_UIB_LDN_DETECTION;
    USB3LNK->lnk_phy_mpll_status   = 0x00310018 | CY_U3P_UIB_SSC_EN;

    CyFx3UsbWritePhyReg (0x0030, 0x00C0);

    CyU3PBusyWait (20);
    UIB->intr_mask |= (CY_U3P_UIB_DEV_CTL_INT | CY_U3P_UIB_DEV_EP_INT |
   CY_U3P_UIB_LNK_INT | CY_U3P_UIB_PROT_INT | CY_U3P_UIB_PROT_EP_INT | CY_U3P_UIB_EPM_URUN);
}
```

6.9.3 USB 3.0 Fallback to USB 2.0

When the USB 3.0 termination detection or link training fails, the FX3 UIB can fall back to USB 2.0 mode. The following function implements this fallback mechanism.

```
static void
CyU3PUsbFallBackToUsb2 (
     void)
{
    CyFx3UsbWritePhyReg (0x1005, 0x0000);

    /* Force the link state machine into SS.Disabled. */
    USB3LNK->lnk_ltssm_state = (CY_U3P_UIB_LNK_STATE_SSDISABLED <<
        CY_U3P_UIB_LTSSM_OVERRIDE_VALUE_POS) |
        CY_U3P_UIB_LTSSM_OVERRIDE_EN;

    /* Keep track of the number of times the 3.0 link training has failed. */
    glUibDeviceInfo.tDisabledCount++;

    /* Change EPM config to full speed */
    CyU3PBusyWait (2);
    CyU3PSetUsbCoreClock (2, 2);
    CyU3PBusyWait (2);

    /* Switch the EPM to USB 2.0 mode, turn off USB 3.0 PHY and remove Rx Termination. */
    UIB->otg_ctrl &= ~CY_U3P_UIB_SSDEV_ENABLE;
    CyU3PBusyWait (2);
    UIB->otg_ctrl &= ~CY_U3P_UIB_SSEPM_ENABLE;

    UIB->intr_mask &= ~(CY_U3P_UIB_DEV_CTL_INT | CY_U3P_UIB_DEV_EP_INT | CY_U3P_UIB_LNK_INT
|
            CY_U3P_UIB_PROT_INT | CY_U3P_UIB_PROT_EP_INT | CY_U3P_UIB_EPM_URUN);

    USB3LNK->lnk_phy_conf        &= 0x1FFFFFFF;
    USB3LNK->lnk_phy_mpll_status  = glUsbMpllDefault;

    /* Power cycle the PHY blocks. */
    GCTLAON->control &= ~CY_U3P_GCTL_USB_POWER_EN;
    CyU3PBusyWait (5);
    GCTLAON->control |= CY_U3P_GCTL_USB_POWER_EN;
    CyU3PBusyWait (10);

    /* Clear USB 2.0 interrupts. */
    UIB->dev_ctl_intr = 0xFFFFFFFF;
    UIB->dev_ep_intr  = 0xFFFFFFFF;
    UIB->otg_intr     = 0xFFFFFFFF;

    /* Reset the EP-0 DMA channels. */
    CyU3PDmaChannelReset (&glUibChHandle);
    CyU3PDmaChannelReset (&glUibChHandleOut);

    /* Clear and disable USB 3.0 interrupts. */
    USB3LNK->lnk_intr_mask   = 0x00000000;
    USB3LNK->lnk_intr        = 0xFFFFFFFF;
    USB3PROT->prot_intr_mask = 0x00000000;
    USB3PROT->prot_intr      = 0xFFFFFFFF;

    UIB->intr_mask |= (CY_U3P_UIB_DEV_CTL_INT | CY_U3P_UIB_DEV_EP_INT |
```

```
   CY_U3P_UIB_LNK_INT | CY_U3P_UIB_PROT_INT | CY_U3P_UIB_PROT_EP_INT | CY_U3P_UIB_EPM_URUN);

    /* Disable EP0-IN and EP0-OUT (USB-2). */
    UIB->dev_epi_cs[0] &= ~CY_U3P_UIB_EPI_VALID;
    UIB->dev_epo_cs[0] &= ~CY_U3P_UIB_EPO_VALID;

    glUibDeviceInfo.usbSpeed = CY_U3P_FULL_SPEED;
    glUibDeviceInfo.isLpmDisabled = CyFalse;
    UIB->ehci_portsc = CY_U3P_UIB_WKOC_E;

    /* Enable USB 2.0 PHY. */
    CyU3PBusyWait (2);
    UIB->otg_ctrl |= CY_U3P_UIB_DEV_ENABLE;

    CyU3PBusyWait (100);
    CyFx3Usb2PhySetup ();
    UIB->phy_clk_and_test = (CY_U3P_UIB_DATABUS16_8 | CY_U3P_UIB_VLOAD |
CY_U3P_UIB_SUSPEND_N |
            CY_U3P_UIB_EN_SWITCH);
    CyU3PBusyWait (80);

    CyU3PSetUsbCoreClock (2, 0);

    /* For USB 2.0 connections, enable pull-up on D+ pin. */
    CyU3PConnectUsbPins ();
}
```

6.9.4 USB Reset

The following code example implements the USB 3.0 reset handler to handle the USB reset event. The USB 3.0 reset event is detected by the LTSSM_RESET bit of the LNK_INTR link layer interrupt register.

```
void CyU3PUsbSSReset(void)
{
    uint8_t ep = 0;

    /* Reset the EPM mux. */
    UIB->iepm_cs |= (CY_U3P_UIB_EPM_FLUSH | CY_U3P_UIB_EPM_MUX_RESET);
    CyU3PBusyWait (1);
    UIB->iepm_cs &= ~(CY_U3P_UIB_EPM_FLUSH | CY_U3P_UIB_EPM_MUX_RESET);
    CyU3PBusyWait (1);

    CyU3PUsbFlushEp (0x00);
    CyU3PUsbFlushEp (0x80);

    /* Enable USB 3.0 control eps. */
    USB3PROT->prot_epi_cs1[0] |= CY_U3P_UIB_SSEPI_VALID;
    UIB->eepm_endpoint[0] = 0x200;                      /* Control EP transfer size is 512
bytes. */
    USB3PROT->prot_epo_cs1[0] |= CY_U3P_UIB_SSEPO_VALID;
    UIB->iepm_endpoint[0] = 0x200;                      /* Control EP transfer size is 512
bytes. */

    CyU3PUsbResetEp (0x00);
    CyU3PUsbFlushEp (0x00);
    CyU3PUsbResetEp (0x80);
    CyU3PUsbFlushEp (0x80);
```

```
    UIB->eepm_endpoint[0] = 0x200;                      /* Control EP transfer size is 512
bytes. */
    UIB->iepm_endpoint[0] = 0x200;                      /* Control EP transfer size is 512
bytes. */

    for (ep = 1; ep < 16; ep++)
    {
        /* Reset, flush and clear stall condition on all valid endpoints. */
        if (glPcktSizeIn[ep].valid == CyTrue)
        {
            CyU3PUsbFlushEp (ep | 0x80);
            CyU3PUsbStall (ep | 0x80, CyFalse, CyTrue);
        }
        if (glPcktSizeOut[ep].valid == CyTrue)
        {
            CyU3PUsbFlushEp (ep);
            CyU3PUsbStall (ep, CyFalse, CyTrue);
        }
    }

}
```

6.9.5 USB Connect

The following code example implements the USB connect handler to handle the USB connect event. This event is detected by the LTSSM_CONNECT bit of the LNK_INTR link layer interrupt register.

```
void CyU3PUsbSSConnecthandler (void)
{
    uint32_t state;
    uint32_t glUsb3TxTrimVal = 0x0B569011;

    /* hardware specific setting for USB 3.0 Phy */
    USB3LNK->lnk_phy_tx_trim = glUsb3TxTrimVal;
    CyFx3UsbWritePhyReg (0x1006, 0x180);
    CyFx3UsbWritePhyReg (0x1024, 0x0080);

    /* If USB 2.0 PHY is enabled, switch it off and take out the USB 2.0 pullup. */
    if (UIB->otg_ctrl & CY_U3P_UIB_DEV_ENABLE)
    {
        state = USB3LNK->lnk_ltssm_state & CY_U3P_UIB_LTSSM_STATE_MASK;
        while ((UIB->otg_ctrl & CY_U3P_UIB_SSDEV_ENABLE)
                       && (state == CY_U3P_UIB_LNK_STATE_POLLING_LFPS))
        {
            CyU3PThreadRelinquish ();
            state = USB3LNK->lnk_ltssm_state & CY_U3P_UIB_LTSSM_STATE_MASK;
        }

        if (state == CY_U3P_UIB_LNK_STATE_COMP)
        {
            if (!glUibDeviceInfo.ssCmdSeen)
            {
                CyU3PUsbAddToEventLog (CYU3P_USB_LOG_USBSS_DISCONNECT);
                CyU3PUsbSSDisConnecthandler ();
            }
            return;
```

```
        }

        if ((UIB->otg_ctrl & CY_U3P_UIB_SSDEV_ENABLE) == 0)
        {
            return;
        }

        CyU3PDisconUsbPins ();
        UIB->otg_ctrl &= ~CY_U3P_UIB_DEV_ENABLE;            /* Disable USB 2.0 PHY. */
        UIB->dev_ctl_intr = UIB->dev_ctl_intr;
    }

    /* Switch EPM clock to USB 3.0 mode. */
    CyU3PSetUsbCoreClock (1, 1);

    USB3LNK->lnk_phy_conf = 0xE0000001;

    glUibDeviceInfo.usbSpeed = CY_U3P_SUPER_SPEED;      /* Remember connection speed. */
    UIB->otg_ctrl |= CY_U3P_UIB_SSEPM_ENABLE;           /* Switch EPMs for USB-SS. */

    /* Reset the EPM mux. */
    UIB->iepm_cs |= (CY_U3P_UIB_EPM_FLUSH | CY_U3P_UIB_EPM_MUX_RESET);
    CyU3PBusyWait (1);
    UIB->iepm_cs &= ~(CY_U3P_UIB_EPM_FLUSH | CY_U3P_UIB_EPM_MUX_RESET);
    CyU3PBusyWait (1);

    /* Update the PHY to not send spurious LFPS. */
    CyFx3UsbWritePhyReg (0x0030, 0x00C0);
    CyFx3UsbWritePhyReg (0x1010, 0x0080);

    CyU3PUsbFlushEp (0x00);
    CyU3PUsbFlushEp (0x80);

    /* Enable USB 3.0 control eps. */
    USB3PROT->prot_epi_cs1[0] |= CY_U3P_UIB_SSEPI_VALID;
    UIB->eepm_endpoint[0] = 0x200;                      /* Control EP transfer size is 512
bytes. */
    USB3PROT->prot_epo_cs1[0] |= CY_U3P_UIB_SSEPO_VALID;
    UIB->iepm_endpoint[0] = 0x200;                      /* Control EP transfer size is 512
bytes. */

    CyU3PUsbResetEp (0x00);
    CyU3PUsbFlushEp (0x00);
    CyU3PUsbResetEp (0x80);
    CyU3PUsbFlushEp (0x80);

    UIB->eepm_endpoint[0] = 0x200;                      /* Control EP transfer size is 512
bytes. */
    UIB->iepm_endpoint[0] = 0x200;                      /* Control EP transfer size is 512
bytes. */

    /* Propagate the event to the application. */
    if (glUsbEvtCb != NULL)
    {
        glUsbEvtCb (CY_U3P_USB_EVENT_CONNECT, 0x01);
    }

    /* Configure the EPs for super-speed operation. */
```

```
    CyU3PUsbEpPrepare (CY_U3P_SUPER_SPEED);
}
```

6.9.6 USB Disconnect

The following code example implements the USB disconnect handler to handle the USB disconnect event. This event is detected by the LTSSM_DISCONNECT bit of the LNK_INTR link layer interrupt register.

```
static void
CyU3PUsbSSDisConnecthandler (
        void)
{
    /* If we still have VBUS, try to connect in USB 2.0 mode. */
    if (CyU3PUsbCanConnect ())
    {
        if (UIB->otg_ctrl & CY_U3P_UIB_DEV_ENABLE)
        {
            /* If the 2.0 PHY is already on, simply turn off the USB 3.0 PHY. */
            UIB->otg_ctrl &= ~(CY_U3P_UIB_SSDEV_ENABLE | CY_U3P_UIB_SSEPM_ENABLE);
            CyU3PBusyWait (2);

            /* Need to disable interrupts while updating the MPLL. */
            UIB->intr_mask &= ~(CY_U3P_UIB_DEV_CTL_INT | CY_U3P_UIB_DEV_EP_INT |
CY_U3P_UIB_LNK_INT |
                    CY_U3P_UIB_PROT_INT | CY_U3P_UIB_PROT_EP_INT | CY_U3P_UIB_EPM_URUN);
            CyU3PBusyWait (1);
            USB3LNK->lnk_phy_conf        &= 0x1FFFFFFF;
            USB3LNK->lnk_phy_mpll_status  = glUsbMpllDefault;
            CyU3PBusyWait (1);
            UIB->intr_mask |= (CY_U3P_UIB_DEV_CTL_INT | CY_U3P_UIB_DEV_EP_INT |
                    CY_U3P_UIB_LNK_INT | CY_U3P_UIB_PROT_INT | CY_U3P_UIB_PROT_EP_INT |
                    CY_U3P_UIB_EPM_URUN);

            CyU3PSetUsbCoreClock (2, 0);
        }
        else
        {
            glUibDeviceInfo.usbSpeed = CY_U3P_FULL_SPEED;
            CyU3PUsbPhyDisable (CyTrue);

            if (glUibDeviceInfo.usb2Disable)
            {
                glUibDeviceInfo.usbState       = CY_U3P_USB_CONFIGURED;
                glUibDeviceInfo.usbSpeed       = CY_U3P_NOT_CONNECTED;
                glUibDeviceInfo.isConnected    = CyFalse;
                if (glUsbEvtCb != NULL)
                    glUsbEvtCb (CY_U3P_USB_EVENT_USB3_LNKFAIL, 0);
            }
            else
            {
                CyU3PUsbPhyEnable (CyFalse);
            }
        }
    }
    else
    {
        /* If no VBUS, disconnect and turn off PHYs. */
```

```
        CyU3PUibVbusChangeHandler ();
    }
}
```

6.9.7 Control Request

The following code example implements the USB control request handler to handle setup commands from the host. Note that this function is implemented for both USB 3.0 and USB 2.0 functions.

For USB 3.0, the control request event is detected by the SUTOK_EV bit of the PROT_INTR protocol layer interrupt register. When the control event occurs, setup data from the host is stored in the protocol layer registers PROT_SETUPDAT0 and PROT_SETUPDAT1.

For USB 2.0, the control request event is detected by the SUDAV bit of the USB 2.0 device register DEV_CTL_INTR. When the control even occurs, setup data from host is stored in the USB 2.0 device registers DEV_SETUPDAT0 and DEV_SETUPDAT1.

Once the function obtains the setup data from the host, it parses the command class and type and executes the command accordingly.

```
/* Parses the USB setup command received. */
static void
CyU3PUsbSetupCommand (
        void)
{
    uint32_t setupdat0;
    uint32_t setupdat1;
    uint32_t status = 0;
    CyBool_t isHandled = CyFalse;

    uint8_t  bRequest, bReqType;
    uint8_t  bType, bTarget;
    uint16_t wValue, wIndex, wLength;

    /* For super speed handling. */
    if (glUibDeviceInfo.usbSpeed == CY_U3P_SUPER_SPEED)
    {
        setupdat0 = USB3PROT->prot_setupdat0;
        setupdat1 = USB3PROT->prot_setupdat1;

        glUibDeviceInfo.ssCmdSeen  = CyTrue;
        glUibStatusSendErdy        = CyFalse;
        CyU3PTimerStop (&glUibStatusTimer);

        /* Clear the status stage interrupt. We later check for this. */
        USB3PROT->prot_intr = CY_U3P_UIB_STATUS_STAGE;

        /* If the LTSSM is currently in U1/U2, set an event to trigger a wakeup. */
        status = USB3LNK->lnk_ltssm_state & CY_U3P_UIB_LTSSM_STATE_MASK;
        if ((status == CY_U3P_UIB_LNK_STATE_U1) || (status == CY_U3P_UIB_LNK_STATE_U2))
            CyU3PEventSet (&glUibEvent, CY_U3P_UIB_EVT_TRY_UX_EXIT, CYU3P_EVENT_OR);
        status = 0;
    }
    else
    {
        /* If USB-SS is enabled, set a flag indicating that the 3.0 PHY should
         * be turned on at the next bus reset. */
```

```
        if ((glUibDeviceInfo.enableSS) && (glUibDeviceInfo.tDisabledCount < 3))
        {
            glUibDeviceInfo.enableUsb3 = CyTrue;
        }

        setupdat0 = UIB->dev_setupdat0;
        setupdat1 = UIB->dev_setupdat1;
    }

    status = CyU3PDmaChannelWaitForCompletion (&glUibChHandleOut, 100);
    if ((status != CY_U3P_SUCCESS) && (status != CY_U3P_ERROR_NOT_STARTED))
    {
        /* The endpoint needs to be NAKed before the channel is reset. */
        CyU3PUsbSetEpNak (0x00, CyTrue);
        CyU3PBusyWait (100);
        CyU3PDmaChannelReset (&glUibChHandleOut);
        CyU3PUsbSetEpNak (0x00, CyFalse);
    }

    status = CyU3PDmaChannelWaitForCompletion (&glUibChHandle, 100);
    if ((status != CY_U3P_SUCCESS) && (status != CY_U3P_ERROR_NOT_STARTED))
    {
        /* The endpoint needs to be NAKed before the channel is reset. */
        CyU3PUsbSetEpNak (0x80, CyTrue);
        CyU3PBusyWait (100);
        CyU3PDmaChannelReset (&glUibChHandle);
        CyU3PUsbFlushEp (0x80);
        CyU3PUsbSetEpNak (0x80, CyFalse);
    }
    status = 0;

    /* Decode the fields from the setup request. */
    bReqType = ((setupdat0 & CY_U3P_UIB_SETUP_REQUEST_TYPE_MASK) >>
CY_U3P_UIB_SETUP_REQUEST_TYPE_POS);
    bType    = (bReqType & CY_U3P_USB_TYPE_MASK);
    bTarget  = (bReqType & CY_U3P_USB_TARGET_MASK);
    bRequest = ((setupdat0 & CY_U3P_UIB_SETUP_REQUEST_MASK) >>
CY_U3P_UIB_SETUP_REQUEST_POS);
    wValue   = ((setupdat0 & CY_U3P_UIB_SETUP_VALUE_MASK) >> CY_U3P_UIB_SETUP_VALUE_POS);
    wIndex   = ((setupdat1 & CY_U3P_UIB_SETUP_INDEX_MASK) >> CY_U3P_UIB_SETUP_INDEX_POS);
    wLength  = ((setupdat1 & CY_U3P_UIB_SETUP_LENGTH_MASK) >> CY_U3P_UIB_SETUP_LENGTH_POS);

    if((setupdat0 & CY_U3P_UIB_SETUP_REQUEST_TYPE_MASK) & 0x80)
    {
        UIB->dev_epi_xfer_cnt[0] = wLength;
    }
    else
    {
        UIB->dev_epo_xfer_cnt[0] = wLength;
    }

    /* Default setting: Don't send status event notifications. */
    glUibDeviceInfo.sendStatusEvent = CyFalse;

    /* Clear the inReset flag. */
    UIB->dev_ctl_intr_mask &= ~CY_U3P_UIB_URESET;
    glUibDeviceInfo.inReset    = 0;
    glUibDeviceInfo.newCtrlRqt = CyFalse;
```

```
    UIB->dev_ctl_intr_mask |= CY_U3P_UIB_URESET;

    /* Forward handling to the callback function iff it is not fast enumeration,
       or if it is not a standard request.
     */
    if ((glUibDeviceInfo.enumMethod == CY_U3P_USBENUM_PPORT) || (bType !=
CY_U3P_USB_STANDARD_RQT))
    {
        if ((bType == CY_U3P_USB_STANDARD_RQT) && (bRequest ==
CY_U3P_USB_SC_SET_CONFIGURATION))
        {
            if (wValue == 1)
            {
                /* Make sure that all EPs are cleared from stall condition and sequence num-
bers are cleared
                   at this stage. */
                CyU3PUsbEpPrepare ((CyU3PUSBSpeed_t)glUibDeviceInfo.usbSpeed);
                glUibDeviceInfo.usbState         = CY_U3P_USB_ESTABLISHED;
                glUibDeviceInfo.usbActiveConfig  = (uint8_t)wValue;
                glUibDeviceInfo.usbDeviceStat   &= CY_U3P_USB_DEVSTAT_SELFPOWER;
            }
            else
            {
                glUibDeviceInfo.usbState = CY_U3P_USB_CONNECTED;
            }
        }

        if (glUsbSetupCb != NULL)
        {
            isHandled = glUsbSetupCb (setupdat0, setupdat1);
        }

        if (isHandled == CyTrue)
        {
            /* If the request is already handled, then return from this function. */
            glUibDeviceInfo.sendStatusEvent = CyTrue;
            return;
        }
    }

    /* Handle the standard requests iff: fast enumeration or if the callback
     * failed to handle the request. */
    if (bType == CY_U3P_USB_STANDARD_RQT)
    {
        /* Identify and handle setup request. */
        switch (bRequest)
        {
        case CY_U3P_USB_SC_GET_STATUS:
            {
                /* Let the setup callback handle GET_STATUS requests addressed to the inter-
face. */
                if (bTarget == CY_U3P_USB_TARGET_INTF)
                {
                    if (glUsbSetupCb)
                    {
                        isHandled = glUsbSetupCb (setupdat0, setupdat1);
                        if (isHandled)
                            break;
```

```
                    }

                    /* If the callback did not handle the request, fall back to the default
handler. */
                }

                isHandled = CyU3PUsbHandleGetStatus (bTarget, wIndex);
            }
            break;

        case CY_U3P_USB_SC_CLEAR_FEATURE:
            {
                /* Handle device level feature requests implicitly. */
                if (bTarget == CY_U3P_USB_TARGET_DEVICE)
                {
                    CyU3PUsbHandleClearFeature ((uint8_t)wValue);
                    return;
                }

                isHandled = CyFalse;

                /* Clear feature request is forwarded to the application regardless of the
enumeration mode.
                   If the application returns false, then the request is handled here. */
                if ((glUibDeviceInfo.enumMethod == CY_U3P_USBENUM_WB) && (glUsbSetupCb !=
NULL))
                {
                    isHandled = glUsbSetupCb (setupdat0, setupdat1);
                    if (isHandled)
                        glUibDeviceInfo.sendStatusEvent = CyTrue;
                }

               /* If the application has not handled this request and this is a clear feature
(EP HALT),
                   handle it here. */
                if ((!isHandled) && (bTarget == CY_U3P_USB_TARGET_ENDPT) && (wValue ==
CY_U3P_USBX_FS_EP_HALT))
                {
                    if (CyU3PUsbStall (wIndex, CyFalse, CyTrue) == CY_U3P_SUCCESS)
                    {
                        /* Reset the endpoint as well */
                        CyU3PUsbResetEp (wIndex);
                        CyU3PUsbAckSetup ();
                        isHandled = CyTrue;
                        break;
                    }
                }
            }
            break;

        case CY_U3P_USB_SC_SET_FEATURE:
            {
                /* Handle all Device specific SET Feature commands automatically. */
                if (bTarget == CY_U3P_USB_TARGET_DEVICE)
                {
                   /* All device level SET_FEATURE requests except for OTG requests are han-
dled in firmware. */
```

```
                    if ((wValue != CY_U3P_USB2_OTG_B_HNP_ENABLE) && (wValue !=
CY_U3P_USB2_OTG_A_HNP_SUPPORT))
                    {
                        CyU3PUsbHandleSetFeature ((uint8_t)wValue);
                        return;
                    }
                    else
                    {
                        /* Send the request to the application. */
                     if ((glUibDeviceInfo.enumMethod == CY_U3P_USBENUM_WB) && (glUsbSetupCb
!= NULL))
                        {
                            isHandled = glUsbSetupCb (setupdat0, setupdat1);
                            if (isHandled)
                            {
                                glUibDeviceInfo.sendStatusEvent = CyTrue;
                                return;
                            }
                        }
                    }
                    break;
                }

                isHandled = CyFalse;

                /* Set feature requests are forwarded to the application regardless of the
enumeration mode.
                   If the application returns false, then the request is handled here. */
                if ((glUibDeviceInfo.enumMethod == CY_U3P_USBENUM_WB) && (glUsbSetupCb !=
NULL))
                {
                    isHandled = glUsbSetupCb (setupdat0, setupdat1);
                    if (isHandled)
                        glUibDeviceInfo.sendStatusEvent = CyTrue;
                }

               /* If the application has not handled this request and this is a set feature
(EP HALT),
                   handle it here. */
                if ((!isHandled) && (bTarget == CY_U3P_USB_TARGET_ENDPT) && (wValue ==
CY_U3P_USBX_FS_EP_HALT))
                {
                    if ((CyU3PUsbStall (wIndex, CyTrue, CyFalse)) == CY_U3P_SUCCESS)
                    {
                        CyU3PUsbAckSetup ();
                        isHandled = CyTrue;
                    }
                }
            }
            break;

        case CY_U3P_USB_SC_GET_DESCRIPTOR:
            {
                isHandled = CyU3PUibSendDescr (setupdat0, setupdat1);
            }
            break;

        case CY_U3P_USB_SC_GET_CONFIGURATION:
```

```
            {
                isHandled = CyTrue;
               status    = CyU3PUsbSendEP0Data (1, (uint8_t *)&glUibDeviceInfo.usbActiveCon-
fig);
            }
            break;

        case CY_U3P_USB_SC_SET_CONFIGURATION:
            {
                {
                    isHandled = CyTrue;

                    switch (wValue)
                    {
                    case 1:
                     /* Make sure that all EPs are cleared from stall condition and sequence
numbers are cleared
                           at this stage. */
                       CyU3PUsbEpPrepare ((CyU3PUSBSpeed_t)glUibDeviceInfo.usbSpeed);

                    case 0:
                      glUibDeviceInfo.usbState       = (wValue == 0) ? CY_U3P_USB_CONNECTED
: CY_U3P_USB_ESTABLISHED;
                        glUibDeviceInfo.usbActiveConfig = (uint8_t)wValue;
                        glUibDeviceInfo.usbDeviceStat  &= CY_U3P_USB_DEVSTAT_SELFPOWER;

                        if (glUsbEvtCb != NULL)
                        {
                            glUsbEvtCb (CY_U3P_USB_EVENT_SETCONF, wValue);
                        }

                        CyU3PUsbAckSetup ();
                        break;

                    default:
                        status = CY_U3P_ERROR_BAD_ARGUMENT;
                        break;
                    }
                }
            }
            break;

        case CY_U3P_USB_SC_GET_INTERFACE:
            {
                if (glUsbSetupCb)
                {
                    isHandled = glUsbSetupCb (setupdat0, setupdat1);
                    if (isHandled)
                        break;
                }

                isHandled = CyTrue;
                status    = CyU3PUsbSendEP0Data (1, (uint8_t *)&glUibDeviceInfo.usbAltSet-
ting);
            }
            break;

        case CY_U3P_USB_SC_SET_INTERFACE:
```

```
            {
                /* First send the command to the Setup callback. Accept the request if not
handled by the callback. */
                if (glUsbSetupCb)
                {
                    isHandled = glUsbSetupCb (setupdat0, setupdat1);
                    if (isHandled)
                        break;
                }

                glUibDeviceInfo.usbAltSetting = wValue;

                if (glUsbEvtCb != NULL)
                {
                    glUsbEvtCb (CY_U3P_USB_EVENT_SETINTF, CY_U3P_MAKEWORD ((uint8_t)wIndex,
(uint8_t)wValue));
                }

                isHandled = CyTrue;
                CyU3PUsbAckSetup ();
            }
            break;

        case CY_U3P_USB_SC_SYNC_FRAME:
            {
                isHandled = CyTrue;
                CyU3PUsbAckSetup ();
            }
            break;

        case CY_U3P_USB_SC_SET_SEL:
            {
                if ((CyU3PUsbGetSpeed () == CY_U3P_SUPER_SPEED) && (wValue == 0) && (wIndex
== 0) && (wLength == 6))
                {
                    isHandled = CyTrue;
                    status = CyU3PUsbGetEP0Data (32, glUibSelBuffer, 0);
                    if ((glUsbEvtCb != NULL) && (status == CY_U3P_SUCCESS))
                    {
                        glUsbEvtCb (CY_U3P_USB_EVENT_SET_SEL, 0x00);
                    }
                }
                else
                {
                    isHandled = CyFalse;
                }
            }
            break;

        case CY_U3P_USB_SC_SET_ISOC_DELAY:
            {
               if ((CyU3PUsbGetSpeed () == CY_U3P_SUPER_SPEED) && (wIndex == 0) && (wLength
== 0))
                {
                    isHandled = CyTrue;
                    CyU3PUsbAckSetup ();
                }
            }
```

```
            break;

        default:
            break;
        }
    }

    /* If there has been an error, stall EP0 to fail the transaction. */
    if ((isHandled != CyTrue) || (status != CY_U3P_SUCCESS))
    {
        /* This is an unhandled setup command. Stall the EP. */
        CyU3PUsbStall (0, CyTrue, CyFalse);
    }
}
```

6.9.8 USB Embedded Host

The FX3 embedded host works in the same way as a standard USB 2.0 host. The FX3 firmware prepares a scheduler table with USB transfer entries that are similar to the Transfer Descriptor (TD) tables maintained by PC hosts. The actual transfer scheduling is done by hardware to eliminate the overheads caused by the firmware managed scheduling.

6.9.8.1 Clocking

There are five independent clock domains in the UIB block supporting the USB 2.0 host functions, as listed in Table 6-4.

Table 6-4. USB 2.0 Embedded Host Clocks

Domain	Typ Freq	Configuration Source	Description
dma_bus_clk_i	100 MHz	GCTL_CPU_CLK_CFG.DMA_DIV	DMA access clock
mmio_bus_clk_i	100 MHz	GCTL_CPU_CLK_CFG.MMIO_DIV	MMIO register access clock
uib_sieclock_i	30 MHz	Driven from USB 2.0 OTG PHY	USB 2.0 serial interface engine clock
standby_clk_i	32 kHz	Always-on	Always-on clock for low-power modes
uib_ref_clk_i	19.2/26/38.4/52 MHz	External input clock	USB2.0 PHY reference clock

In addition, the endpoint memory uses a clock, uib_epm_clk_i, that is 100 MHz when the USB 2.0 host is active. The uib_epm_clk_i configuration source is from GCTL_UIB_CORE_CLK.EPMCLK_SRC and enabled by GCTL_UIB_CORE_CLK.CLK_EN.

6.9.9 Interrupt Requests

The USB 2.0 host interrupts are part of the main USB core interrupt.

USB 2.0 OHCI interrupts are located in UIB_OHCI_INTERRUPT_STATUS. UIB_INTR.HOST_INT is the logical OR of the interrupt sources in UIB_OHCI_INTERRUPT_STATUS.

USB 2.0 EHCI interrupts are located in UIB_EHCI_USBINTR. UIB_INTR.HOST_INT is the logical OR of the interrupt sources in UIB_EHCI_USBINTR.

USB 2.0 host endpoint interrupts are located in UIB_HOST_EP_INTR. UIB_INTR.HOST_EP_INT is the logical OR of the interrupt sources in UIB_HOST_EP_INTR.

USB charger detect interrupts are located in UIB_CHGDET_INTR. UIB_INTR.CHGDET_INT is the logical OR of the interrupt sources in UIB_CHGDET_INTR.

6.9.10 Functional Description

6.9.10.1 Embedded Host

The FX3 USB 2.0 embedded host operates like an EHCI host when in high-speed mode and like an OHCI host when in low- or full-speed mode, but the implementation does not follow the fields of the data structures specified in the EHCI/OHCI spec.

At the lowest level, incoming and outgoing data are managed by the DMA block. All USB transfer types, namely control, bulk, isochronous and interrupt; are handled in a similar way by the DMA block. Refer to the FX3 DMA Subsystem chapter on page 61 to learn how the DMA descriptor and socket data structures are organized and used.

As a USB host, data traffic is initiated by configuring the appropriate entries in the scheduler memory areas inside the USB 2.0 host controller. The USB 2.0 host controller hardware scans the scheduler memory for valid entries that contain the active endpoint configurations and schedules data on the bus accordingly.

6.9.10.2 Scheduler Memory

Figure 6-3 shows the bit field of a scheduler entry data structure positioned in the scheduler memory. Definitions of the bit fields are provided in the register definition section. Each scheduler entry requires three 32-bit words, 12 bytes of memory. There are 32 entries for EP, which sums the total available memory of 384 bytes.

Figure 6-3. Scheduler Memory Entry

Memory bits												Addr
31:24	23	22:19	18:17	16	15:14	13:10	9	8	7	6:5	4:0	I
S_mask	PING	RL	MULT	ISO_EPM	CERR	NakCnt	Halt	T	ZLPEN	EPT	EPND	

31:30	29	28:27	26:19	18:11	10:0	I+1
EP0_code	Bypass_err	MMULT	Resp_rate	Polling_rate	Max_Packet_size	

31:25	24:17	16	15:0	I+2
Reserved	ioc_rate	trns_mode	Total Byte Count	

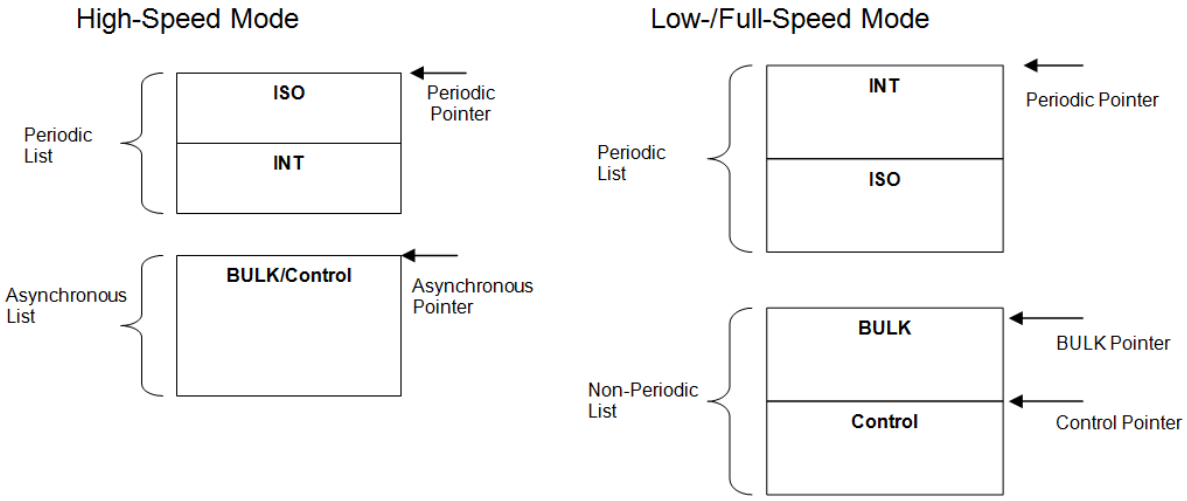
6.9.10.2.1 Scheduler Memory Organization

A host controller driver has the option to add or remove entries in the list. Two copies of the scheduler memory table are provided so that firmware can first update the table and then make it active.

Active scheduler entries in the scheduler memory form the execution list for the host controller hardware. The execution list is broken into the periodic list which services periodic (isochronous and interrupt) endpoints and the asynchronous list which services bulk and control endpoints.

Each scheduler memory area contains a periodic list and an asynchronous/non-periodic list. Figure 6-4 shows how the list is organized in high-speed mode, and low- or full-speed mode. The periodic list pointer points to the first entry in the periodic list. It always starts from the lowest address of each scheduler memory.

Figure 6-4. Scheduler Memory Entry List Data Structure Organization



The asynchronous/non-periodic list pointer points to the location of the first entry in the asynchronous/non-periodic list. It is programmed via HOST_SHDL_CS.BULK_CNTRL_PTR0 and HOST_SHDL_CS.BULK_CNTRL_PTR1 for the two areas of scheduler memory respectively.

When the firmware needs to update to the current execution list, it first finds the active area on which the host is currently running by reading HOST_SHDL_CS.PERI_SHDL_STATUS or HOST_SHDL_CS.ASYNC_SHDL_STATUS, updates the list in the nonactive area, and then issues a schedule change operation by writing a '1' to HOST_SHDL_CS.PERI_SHDL_CHNG or HOST_SHDL_CS.ASYNC_SHDL_CHNG.

The FX3 USB embedded host registers can be accessed directly from the UIB top-level register interface. Following is a list of these registers.

```
/* FX3 USB 2.0 Embedded Host Register Interface */
/* These definitions extracted from the Top Level UIB register interface */

/* Common host registers */
    uvint32_t host_cs;                          /* 0xe0032000 */
    uvint32_t host_ep_intr;                     /* 0xe0032004 */
    uvint32_t host_ep_intr_mask;                /* 0xe0032008 */
    uvint32_t host_toggle;                      /* 0xe003200c */
    uvint32_t host_shdl_cs;                     /* 0xe0032010 */
    uvint32_t host_shdl_sleep;                  /* 0xe0032014 */
    uvint32_t host_resp_base;                   /* 0xe0032018 */
    uvint32_t host_resp_cs;                     /* 0xe003201c */
    uvint32_t host_active_ep;                   /* 0xe0032020 */

/* OHCI registers */
    uvint32_t ohci_revision;                    /* 0xe0032024 */
    uvint32_t ohci_control;                     /* 0xe0032028 */
    uvint32_t ohci_command_status;              /* 0xe003202c */
    uvint32_t ohci_interrupt_status;            /* 0xe0032030 */
    uvint32_t ohci_interrupt_enable;            /* 0xe0032034 */
    uvint32_t ohci_interrupt_disable;           /* 0xe0032038 */
    uvint32_t ohci_fm_interval;                 /* 0xe003203c */
    uvint32_t ohci_fm_remaining;                /* 0xe0032040 */
    uvint32_t ohci_fm_number;                   /* 0xe0032044 */
    uvint32_t ohci_periodic_start;              /* 0xe0032048 */
    uvint32_t ohci_ls_threshold;                /* 0xe003204c */
```

```
    uvint32_t ohci_rh_port_status;              /* 0xe0032054 */
    uvint32_t ohci_eof;                         /* 0xe0032058 */

/* EHCI registers */
    uvint32_t ehci_hccparams;                   /* 0xe003205c */
    uvint32_t ehci_usbcmd;                      /* 0xe0032060 */
    uvint32_t ehci_usbsts;                      /* 0xe0032064 */
    uvint32_t ehci_usbintr;                     /* 0xe0032068 */
    uvint32_t ehci_frindex;                     /* 0xe003206c */
    uvint32_t ehci_configflag;                  /* 0xe0032070 */
    uvint32_t ehci_portsc;                      /* 0xe0032074 */
    uvint32_t ehci_eof;                         /* 0xe0032078 */
    uvint32_t shdl_chng_type;                   /* 0xe003207c */
    uvint32_t shdl_state_machine;               /* 0xe0032080 */
    uvint32_t shdl_internal_status;             /* 0xe0032084 */

/* OHCI scheduler entry */
    struct
    {
        uvint32_t shdl_ohci0;                   /* 0xe0032400 */
        uvint32_t shdl_ohci1;                   /* 0xe0032404 */
        uvint32_t shdl_ohci2;                   /* 0xe0032408 */
    } ohci_shdl[64];

/* EHCI scheduler entry */
    struct
    {
        uvint32_t shdl_ehci0;                   /* 0xe0032800 */
        uvint32_t shdl_ehci1;                   /* 0xe0032804 */
        uvint32_t shdl_ehci2;                   /* 0xe0032808 */
    } ehci_shdl[64];
```

6.9.11 Embedded Host Programming Model

6.9.11.1 Host Connect

These steps are followed when FX3 detects the attachment of a device.

1. The firmware enables the USB bus power through the external power management IC (PMIC) controller.
2. The device connect is detected by the host TP, which signals the EHCI interface. UIB_HOST_EHCI_PORTSC:PORT_CONNECT is set, which sets the UIB_HOST_EHCI_PORTSC:PORT_CONNECT_C field and the UIB_HOST_EHCI_USBSTS:PORT_CHNG_DET field and generates an interrupt request.
3. The firmware receives the interrupt, clears the interrupt, and checks whether a low-speed device has connected. If so, then the device is accessed through the OHCI register interface.
4. As high and full speed devices can only be distinguished after the port reset and chirp sequence is completed, the initial detect sequence for both are performed by the EHCI controller. This is started by issuing a USB bus reset by setting the UIB_HOST_EHCI_PORTSC:PORT_RESET bit. This bit should be cleared after 20 ms.
5. After 2 ms, the firmware checks the UIB_HOST_EHCI_PORTSC:PORT_ENABLE bit to see whether or not the port is enabled. If so, then the device is high speed and the EHCI interface is used. If not, then the device is full speed.
6. If a Full Speed device is detected,, the OHCI Interface is enabled by setting the UIB_HOST_EHCI_PORTSC:PORTOWNER bit to 1. At this point, the connect signal to the EHCI interface is cleared, and the EHCI interface sees a disconnect and generates another interrupt request.

6.9.11.2 Host Disconnect

These steps are followed when the device is disconnected from the host:

1. The device disconnect is detected by the host TP, which signals the EHCI interface. The UIB_HOST_EHCI_PORTSC:PORT_CONNECT_C field (and through that the UIB_HOST_EHCI_USBSTS:PORT_CHNG_DET field) generates an interrupt request.
2. At the same time, the UIB_HOST_EHCI_PORTSC:PORTOWNER field is cleared, transferring port ownership to the EHCI interface.
3. After disconnect, the EHCI interface resumes its role as the default port owner.

6.9.11.3 Managing Transfers

The USB 2.0 host hardware scheduler reads elements from the scheduler memory and based on its fields along with the EPM status, it decides to schedule the element for SIE. The result of the executed transaction is recorded in the status fields. The status of all the transactions for that scheduler entry are recorded as "sticky" in the status field until the scheduler entry is complete.

Every entry in the scheduler memory represents one queue head (QH) in EHCI terminology or endpoint descriptor (ED) in OHCI terminology. From this point on, this document calls it "QH" for convenience. In the fetch QH, one entry of the scheduler memory is read. If that entry cannot be scheduled for the SIE due to halt bit set or entry not active, then the scheduler skips and next QH. The software updates the active bit in the scheduler memory only during the scheduler memory update. The active bit is set to zero when the host controller successfully executes the total_byte_cnt of transactions. The USB 2.0 host controller reports the active/nonactive condition through the status bits. The bus scheduler does not execute a QH when its active bit is zero.

The status of the USB transactions is written directly into system memory based on the response rate written in scheduler memory per endpoint. The base address where scheduler responses are written is programmed via HOST_RESP_BASE.BASE_ADDRESS. The response condition is programmed via the HOST_RESP_CS register. The response rate indicates after how many either successful transactions or not NAKed transactions the status should be written into system memory for the software to process. The firmware can use the UIB_HOST_RESP_CS.WR_RESP_COND register bit to decide on the successful or not NAKed transactions. Table 6-5 provides a description of the status.

Table 6-5. USB 2.0 Host Response

Bit	Name	Description
3:0	EP_NUM	Endpoint number
4	Direction	0: IN 1: Out
5	Active	0: EP got deactivated. 1: EP is still active.
6	Halt	1: EP is not halted. 2: EP is halted.
7	Overrun/ Underrun	0: No overrun/underrun condition 1: There was an overrun/underrun when accessing the EPM.
8	Babble	0: No babble is detected. 1: Babble was detected.
9	XactErr	PhyErr or CRC16 error, or device times out, or PID error or IN-ISO XactError IN-ISO XactErr: The first transaction for the IN ISO has a data toggle equal, or More than the MULT field is specified in the scheduler memory, or There is a data toggle mismatch in the transaction excluding the first transaction, or A short IN-ISO with a data toggle greater than one was received.
10	Ping	0: No ping token has been issued. 1: Ping token has been issued when enabled.
15:11	-	Reserved
31:16	Byte Count	Total byte count left

6.9.11.3.1 Control Transfer Specific

An EP0 control transfer falls into three categories:

Setup transaction: N number of OUT data transactions followed by 1 IN token (ZLP)

Setup transaction: N number of IN data transactions followed by 1 OUT token (ZLP)

Setup transaction: 1 IN token (ZLP)

Every EP0 transaction consists of at least two transactions, but the asynchronous list in the scheduler memory has only one entry for every endpoint. One mechanism to initiate multiple transactions by the USB2.0 host controller is to setup EP0_code through the scheduler memory entry.

The ep0_code is defined as:

00: Reserved

01: Setup Phase + Data Phase(OUT) + Status Phase(IN)

10: Setup Phase + Data Phase(IN) + Status Phase(OUT)

11: Setup Phase + Status Phase(IN)

6.9.11.3.1.1 Setup Phase

This EP0 category starts once the scheduler encounters the EP0 in the scheduler memory list for the first time, and the ep0_code is "01". It loads its total byte counter from the schedule memory entry. This category indicates that after the SETUP gets ACKed, the next scheduled EP0 is an OUT transaction. The scheduler informs the TP to issue a SETUP token. If the SETUP token is successful, then the next time the scheduler encounters the EP0 in the list, it will inform the TP for an EP0-OUT transaction. If the SETUP token is not successful, then the EP0 transaction is terminated and it will retry it the next time it encounters the EP0 in the scheduler memory, as long as the EP0 is still active and not halted.

6.9.11.3.1.2 Data Phase(OUT)

The EP0 OUT transaction starts once the scheduler encounters the EP0 in the list, and EP0 state is the OUT state. The internal total byte counter is decremented by the actual OUT DATA size sent to device. Once the byte counter reaches zero and if the ZLPEN is "0" in the scheduler memory, the then next time the scheduler will request the TP to issue an EP0 IN to the device.

If the ZLPEN is "1", then the next time the scheduler will request the TP to issue an EP0 OUT zero-length packet (ZLP) to the device. After that, the scheduler will request the TP to issue an EP0 IN to the device.

6.9.11.3.1.3 Status Phase (IN)

Once a valid EP0 IN ZLP has been received, the scheduler deactivates the EP0 entry if the trns_mode of the scheduler entry is "0".

6.9.11.3.1.4 Setup Phase

This EP0 category starts once the scheduler encounters the EP0 in the scheduler memory list for the first time, and the ep0_code is "10". It loads its total byte counter from the schedule memory entry. This category indicates that after the SETUP gets ACKed, the next scheduled EP0 is an IN transaction. The scheduler informs the TP to issue a SETUP token. If the SETUP token is successful, then the next time the scheduler encounters the EP0 in the list, it will inform the TP for an EP0 IN transaction. If the SETUP token is not successful, then the EP0 transaction is terminated and it will retry it the next time it encounters the EP0 in the scheduler memory, as long as the EP0 is still active and not halted.

6.9.11.3.1.5 Data Phase (IN)

The EP0 IN transaction starts once the scheduler encounters the EP0 in the list, and the EP0 state is IN state. Processing an IN EP0 is the same as processing any other IN EP. The internal total byte counter is decremented by the actual IN DATA size received from device. Once the byte counter reaches or a short packet is received, and the ZLPEN is "0" in the scheduler memory, then next time the scheduler will request the TP to issue an EP0 OUT ZLP to the device. If the ZLPEN is "1", then the next time the scheduler will request the TP to issue another EP0 IN to the device, which should be a ZLP. After that, the scheduler will request the TP to issue an EP0 OUT ZLP to the device.

6.9.11.3.1.6 Status Phase (OUT)

Once an EP0 OUT ZLP has been successfully received by the device, the scheduler deactivates the EP0 entry if the trns_mode of the scheduler entry is "0".

6.9.11.3.1.7 Setup Phase

This EP0 category starts once the scheduler encounters the EP0 in the scheduler memory list for the first time, and the ep0_code is "11". This category indicates that after the SETUP gets ACKed, the next scheduled EP0 is an IN transaction. The scheduler informs the TP to issue a SETUP token. If the SETUP token is successful, then the next time the scheduler encounters the EP0 in the list, it will inform the TP for an EP0-IN transaction. If the SETUP token is not successful, then the EP0 transaction is terminated and it will retry it the next time it encounters the EP0 in the scheduler memory, as long as the EP0 is still active and not halted.

6.9.11.3.1.8 Status Phase (IN)

Once a valid EP0 IN ZLP has been received, the scheduler deactivates the EP0 entry if the trns_mode of the scheduler entry is "0".

6.10 USB OTG Controller

The FX3 USB OTG controller implements the hardware interface between the processor and the USB PHY to allow the firmware to implement the OTG features. The OTG function requires the firmware implementation to handle the protocol. The controller hardware generates interrupts on the events, and the firmware handles the event and responds to it as needed.

The FX3 USB OTG controller is compatible with USB OTG 2.0 specifications.

USB OTG defines two protocols: SRP and HNP. SRP is a method for a peripheral to request that the host enable the VBus bus power. Some hosts may wish to disable the VBus bus power to conserve system power, and this protocol allows peripherals to connect to the host under such conditions.

The HNP protocol is a method of allowing a peripheral and host to switch roles, with the peripheral serving as the host and the host receiving commands from the peripheral. For hosts and peripherals, it is necessary to have the D- pull-down resistor enabled at all times. When a device is acting as a host, it must also enable the D+ pull-down resistor.

6.10.1 Interrupt Requests

USB OTG interrupts are located in UIB_OTG_INTR. UIB_INTR.OTG_INT is the logical OR of the interrupt sources in UIB_OTG_INTR.

6.10.2 USB OTG Programming Model

FX3 USB OTG controller registers can be accessed directly from the UIB top-level register interface. Following is the list of these registers.

```
/* FX3 USB OTG Register Interface */
/* These definitions extracted from the Top Level UIB register interface */

    uvint32_t otg_ctrl;                         /* 0xe003180c */
    uvint32_t otg_intr;                         /* 0xe0031810 */
    uvint32_t otg_intr_mask;                    /* 0xe0031814 */
    uvint32_t otg_timer;                        /* 0xe0031818 */
```

6.10.2.1 USB OTG Start and Stop

The FX3 OTG controller needs to be initialized before it can handle OTG events. The following code example implements the OTG controller start and stop sequence.

```
CyU3PReturnStatus_t
CyU3POtgStart (
        CyU3POtgConfig_t *cfg)
{
```

```
/* Verify that the part in use supports the USB OTG functionality. */
if (!CyFx3DevIsOtgSupported ())
    return CY_U3P_ERROR_NOT_SUPPORTED;

if (glIsOtgEnable)
{
    return CY_U3P_ERROR_ALREADY_STARTED;
}

/* This call is not allowed if the device mode is already active. */
if ((CyU3PUsbIsStarted ()) || (glSdk_UsbIsOn))
{
    return CY_U3P_ERROR_INVALID_SEQUENCE;
}

/* Check for parameter validity. */
if (cfg == NULL)
{
    return CY_U3P_ERROR_NULL_POINTER;
}
if (cfg->otgMode >= CY_U3P_OTG_NUM_MODES)
{
    return CY_U3P_ERROR_BAD_ARGUMENT;
}
if ((cfg->cb == NULL) && (cfg->otgMode != CY_U3P_OTG_MODE_CARKIT_PPORT) &&
            (cfg->otgMode != CY_U3P_OTG_MODE_CARKIT_UART))
{
    return CY_U3P_ERROR_NULL_POINTER;
}
if (cfg->chargerMode >= CY_U3P_OTG_CHARGER_DETECT_NUM_MODES)
{
    return CY_U3P_ERROR_BAD_ARGUMENT;
}

/* Register the handlers for OTG interrupt events. */
CyU3PUsbSetOTGIntHandler (CyU3PUsbOtgIntHandler);
CyU3PUsbSetChgDetIntHandler (CyU3PUsbChgdetIntHandler);

/* Reset the connected peripheral to invalid entry. */
glPeripheralType = CY_U3P_OTG_TYPE_DISABLED;
/* Reset the state machine variables. */
glIsHnpEnable = CyFalse;

if (cfg->otgMode == CY_U3P_OTG_MODE_OTG)
{
    /* Enable the USB PHY connections. */
    GCTLAON->control &= ~CY_U3P_GCTL_ANALOG_SWITCH;

    /* Enable and set the EPM clock to bus clock (100MHz). */
    GCTL->uib_core_clk = (CY_U3P_GCTL_UIBCLK_CLK_EN | CY_U3P_GCTL_UIB_CORE_CLK_DEFAULT);
    GCTLAON->control |= CY_U3P_GCTL_USB_POWER_EN;
    CyU3PBusyWait (100);

    CyU3PUsbPowerOn ();

    /* Enable OTG and charger detection interrupts. */
    UIB->otg_intr = ~CY_U3P_UIB_OTG_INTR_DEFAULT;
    UIB->otg_intr_mask = CY_U3P_UIB_OTG_TIMER_TIMEOUT;
```

```
        UIB->chgdet_intr = ~CY_U3P_UIB_CHGDET_INTR_DEFAULT;
        UIB->chgdet_intr_mask = CY_U3P_UIB_OTG_ID_CHANGE;

        /* Enable OTG mode and charger detection. */
        UIB->otg_ctrl = (CY_U3P_UIB_OTG_ENABLE);

        /* Enable ID pin detection. */
    UIB->chgdet_ctrl = CY_U3P_UIB_ACA_ENABLE;

        /* Enable the UIB interrupts. */
        UIB->intr = ~CY_U3P_UIB_INTR_DEFAULT;
        UIB->intr_mask = (CY_U3P_UIB_OTG_INT | CY_U3P_UIB_CHGDET_INT);
        CyU3PVicEnableInt (CY_U3P_VIC_UIB_CORE_VECTOR);

        /* Enable the VBUS interrupts. */
        GCTL->iopwr_intr = ~CY_U3P_GCTL_IOPWR_INTR_DEFAULT;
        GCTL->iopwr_intr_mask = CY_U3P_VBUS;
        glUibDeviceInfo.vbusDetectMode = CY_U3P_VBUS;
        CyU3PVicEnableInt (CY_U3P_VIC_GCTL_PWR_VECTOR);
    }
    else if (cfg->otgMode == CY_U3P_OTG_MODE_CARKIT_PPORT)
    {
        /* Enable the USB PHY connections. */
        GCTLAON->control &= ~CY_U3P_GCTL_ANALOG_SWITCH;

        /* Enable and set the EPM clock to bus clock (100MHz). */
        GCTL->uib_core_clk = (CY_U3P_GCTL_UIBCLK_CLK_EN | CY_U3P_GCTL_UIB_CORE_CLK_DEFAULT);
        GCTLAON->control |= CY_U3P_GCTL_USB_POWER_EN;
        CyU3PBusyWait (100);

        CyU3PUsbPowerOn ();

        UIB->otg_ctrl = CY_U3P_UIB_OTG_CTRL_DEFAULT;
        GCTL->iomatrix |= CY_U3P_CARKIT;
        UIB->chgdet_ctrl = CY_U3P_UIB_CARKIT;
    }
    else if (cfg->otgMode == CY_U3P_OTG_MODE_CARKIT_UART)
    {
        /* Verify if the configuration is possible. */
        if ((CyFx3DevIOIsSib8BitWide (1)) || (CyFx3DevIOIsUartConfigured ()))
        {
            return CY_U3P_ERROR_BAD_ARGUMENT;
        }

        /* Enable the USB PHY connections. */
        GCTLAON->control &= ~CY_U3P_GCTL_ANALOG_SWITCH;

        /* Enable and set the EPM clock to bus clock (100MHz). */
        GCTL->uib_core_clk = (CY_U3P_GCTL_UIBCLK_CLK_EN | CY_U3P_GCTL_UIB_CORE_CLK_DEFAULT);
        GCTLAON->control |= CY_U3P_GCTL_USB_POWER_EN;
        CyU3PBusyWait (100);

        CyU3PUsbPowerOn ();

        UIB->otg_ctrl = CY_U3P_UIB_OTG_CTRL_DEFAULT;
        GCTL->iomatrix &= ~CY_U3P_CARKIT;
        UIB->chgdet_ctrl = CY_U3P_UIB_CARKIT;
    }
```

```
    else
    {
        /* Do nothing. */
    }

    /* Save the current OTG configuration. */
    glOtgInfo = *cfg;
    glIsOtgEnable = CyTrue;

    return CY_U3P_SUCCESS;
}

CyU3PReturnStatus_t
CyU3POtgStop (void)
{
    if (!glIsOtgEnable)
    {
        return CY_U3P_ERROR_NOT_STARTED;
    }

    /* Do not disable the block if any of the modes are running. */
    if ((CyU3PUsbIsStarted ()) || (CyU3PUsbHostIsStarted ()))
    {
        return CY_U3P_ERROR_INVALID_SEQUENCE;
    }

    /* Disable all interrupts. */
    UIB->intr_mask = 0;
    CyU3PVicDisableInt (CY_U3P_VIC_UIB_CORE_VECTOR);
    CyU3PVicDisableInt (CY_U3P_VIC_UIB_DMA_VECTOR);

    /* Reset all state machine variables to default. */
    glOtgInfo.otgMode = CY_U3P_OTG_MODE_DEVICE_ONLY;
    glOtgInfo.chargerMode = CY_U3P_OTG_CHARGER_DETECT_ACA_MODE;
    glOtgInfo.cb = NULL;
    glPeripheralType = CY_U3P_OTG_TYPE_DISABLED;
    glIsHnpEnable = CyFalse;
    UIB->chgdet_ctrl = CY_U3P_UIB_CHGDET_CTRL_DEFAULT;
    UIB->otg_ctrl = CY_U3P_UIB_OTG_CTRL_DEFAULT;

    /* Disable the UIB block. */
    UIB->power &= ~CY_U3P_UIB_RESETN;
    CyU3PBusyWait (10);

    /* Disable the UIB clock. */
    GCTL->uib_core_clk &= ~CY_U3P_GCTL_UIBCLK_CLK_EN;
    GCTLAON->control &= ~CY_U3P_GCTL_USB_POWER_EN;

    /* Update the state variable. */
    glIsOtgEnable = CyFalse;

    return CY_U3P_SUCCESS;
}
```

6.10.2.2 Session Request Protocol

Upon attachment to the USB bus, if the USB bus power VBUS is not on, the B device can request the host to drive VBUS and begin to communicate with the peripheral. This process is called "Session Request Protocol" (SRP).

In OTG 2.0, a SRP request is started with D+ pulsing. When FX3 is connected as a B-device, the SRP follows this sequence of steps:

1. The firmware checks that no current session is ongoing by checking that the SESS_END bit in the OTG control register is set and also that the DP and DM bits are cleared.
2. The firmware sets the DP_PU_EN bit in the OTG control register to pulse the USB D+ line and loads the OTG timer register to generate an interrupt between 5 ms and 10 ms later.
3. After receiving the OTG timer timeout interrupt, the processor clears the DP_PU_EN bit to stop pulsing the USB D+ signal.
4. The firmware enables the VBUS_VALID_INT interrupt and waits for VBUS to be powered.
5. After receiving the VBUS_VALID_INT interrupt, the firmware enables the D+ pull-up by enabling the USB 2.0 function controller, which allows it to connect to the USB host.
6. If the host is capable, it drives VBUS and allows FX3 to connect as a peripheral.
7. Optionally, the firmware may enable DSCHG_VBUS to discharge the VBUS line below the levels specified for SESS_END to begin the SRP process sooner.

As an A-device, the SRP enables the SRP interrupt to allow SRP signaling detection and waits for the SRP interrupt. Upon receiving the interrupt, the firmware enables the VBUS power, allowing the B-device to connect. Once enabled, the firmware should also enable the VBUS_VALID_INT interrupt to ensure that the peripheral is not drawing more current than the system can provide. The firmware sequence is as follows:

1. After checking that DP = 0 and DM = 0 and B_SESS_END is asserted, enable the SRP interrupt.
2. Upon receiving the SRP Interrupt, enable the VBUS bus power as well as the USB host logic and its connect interrupt.
3. Upon receiving the connect interrupt, begin normal USB traffic to the B-device, starting with USB bus reset and enumeration.

The following code example implements the function to issue an SRP request.

```
static CyU3PReturnStatus_t
CyU3POtgIssueSrp (void)
{
    uint32_t regVal;

    if (!glIsOtgEnable)
    {
        return CY_U3P_ERROR_NOT_STARTED;
    }

    /* Check if device mode is enabled. */
    if ((!CyU3POtgIsDeviceMode ()) || (CyU3PUsbIsStarted ()) || (CyU3PUsbHostIsStarted ()))
    {
        return CY_U3P_ERROR_INVALID_SEQUENCE;
    }

    /* Check if there is any valid session. */
    if (GCTL->iopower & glUibDeviceInfo.vbusDetectMode)
    {
        return CY_U3P_ERROR_INVALID_SEQUENCE;
    }

    /* This is a hack as there is no way to send a pulse on the VBUS.
     * Here we are first configuring the PHY as a device and doing a DP
     * pull-up to send a high. To send a low, the PHY is configured as host
```

```
     * and then DP is pulled down. This seems to work without the VBUS. */

    /* Enable CHG_VBUS. */
    UIB->otg_ctrl |= CY_U3P_UIB_CHG_VBUS;
    CyU3PThreadSleep (10);

    /* Pullup DP */
    regVal = UIB->otg_ctrl & (CY_U3P_UIB_OTG_ENABLE | CY_U3P_UIB_CHG_VBUS);
    UIB->otg_ctrl = (regVal | CY_U3P_UIB_DEV_ENABLE | CY_U3P_UIB_DP_PU_EN);
    CyU3PThreadSleep (10);
    /* Pull down DP */
    UIB->otg_ctrl = (regVal | CY_U3P_UIB_HOST_ENABLE | CY_U3P_UIB_DP_PD_EN);
    CyU3PThreadSleep (10);
    /* Set the PHY to be disconnected. */
    UIB->otg_ctrl = regVal;

    /* Disable CHG_VBUS. */
    UIB->otg_ctrl &= ~CY_U3P_UIB_CHG_VBUS;
    CyU3PThreadSleep (10);

    return CY_U3P_SUCCESS;
}
```

The following code example implements the function that accepts the SRP request from the A-device.

```
static void
CyU3POtgSetupPhy (void)
{
    if (CyU3POtgIsHostMode ())
    {
        /* Enable and set the EPM clock to bus clock (100MHz). */
        GCTL->uib_core_clk = (CY_U3P_GCTL_UIBCLK_CLK_EN | CY_U3P_GCTL_UIB_CORE_CLK_DEFAULT);

        /* Enable host mode. */
        UIB->otg_ctrl &= CY_U3P_UIB_OTG_ENABLE;
        UIB->otg_ctrl |= CY_U3P_UIB_HOST_ENABLE;
        /* Reset and enable the PHY. */
        UIB->phy_clk_and_test = (CY_U3P_UIB_DATABUS16_8 | CY_U3P_UIB_VLOAD |
                CY_U3P_UIB_RESET | CY_U3P_UIB_EN_SWITCH);
        CyU3PBusyWait (10);
        UIB->phy_clk_and_test &= ~CY_U3P_UIB_RESET;
        UIB->phy_clk_and_test |= CY_U3P_UIB_SUSPEND_N;
        CyU3PBusyWait (100);
        /* Switch EPM clock to 120MHz. */
        GCTL->uib_core_clk &= ~CY_U3P_GCTL_UIBCLK_EPMCLK_SRC_MASK;

        /* Make sure EHCI is the owner of the port. */
        UIB->ehci_configflag = CY_U3P_UIB_CF;
        CyU3PBusyWait (10);
    }
    if (CyU3POtgIsDeviceMode ())
    {
        UIB->otg_ctrl &= CY_U3P_UIB_OTG_ENABLE;
        UIB->otg_ctrl |= CY_U3P_UIB_DEV_ENABLE;

        /* Enable and set the EPM clock to bus clock (100MHz). */
        GCTL->uib_core_clk = (CY_U3P_GCTL_UIBCLK_CLK_EN |
                CY_U3P_GCTL_UIB_CORE_CLK_DEFAULT);
```

```
        CyU3PBusyWait (100);

        /* Reset and enable the PHY. */
        UIB->phy_clk_and_test = (CY_U3P_UIB_DATABUS16_8 | CY_U3P_UIB_VLOAD |
                CY_U3P_UIB_RESET | CY_U3P_UIB_EN_SWITCH);
        CyU3PBusyWait (10);
        UIB->phy_clk_and_test &= ~CY_U3P_UIB_RESET;
        UIB->phy_clk_and_test |= CY_U3P_UIB_SUSPEND_N;
        CyU3PBusyWait (100);

        /* Switch EPM clock to 120MHz. */
        GCTL->uib_core_clk &= ~CY_U3P_GCTL_UIBCLK_EPMCLK_SRC_MASK;
        CyU3PBusyWait (10);
        UIB->otg_ctrl &= ~CY_U3P_UIB_DEV_ENABLE;
    }
}

CyU3PReturnStatus_t
CyU3POtgSrpStart (uint32_t repeatInterval)
{
    if (!glIsOtgEnable)
    {
        return CY_U3P_ERROR_NOT_STARTED;
    }
    if ((repeatInterval > CY_U3P_OTG_SRP_MAX_REPEAT_INTERVAL) ||
            (repeatInterval == 0))
    {
        return CY_U3P_ERROR_BAD_ARGUMENT;
    }

    /* Check if device mode is enabled. */
    if ((!CyU3POtgIsDeviceMode ()) || (CyU3PUsbIsStarted ()) || (CyU3PUsbHostIsStarted ()))
    {
        return CY_U3P_ERROR_INVALID_SEQUENCE;
    }

    /* Check if there is any valid session. */
    if (GCTL->iopower & glUibDeviceInfo.vbusDetectMode)
    {
        return CY_U3P_ERROR_INVALID_SEQUENCE;
    }

    CyU3POtgSetupPhy ();

    /* Update the required timer period value. */
    glOtgTimerPeriod = (uint32_t)(repeatInterval * 32);

    /* Initialize the OTG timer with the corresponding repeat period. */
    UIB->otg_timer = glOtgTimerPeriod;

    return CY_U3P_SUCCESS;
}
```

6.10.2.3 Host Negotiation Protocol

An OTG A-device acts as the host initially in a USB session. In order to allow a peripheral to assume the role of a host, the initial host must first configure the peripheral to enable HNP support through USB commands (SetFeature(b_hnp_enable)).

To start the role change, the initial host suspends the USB bus, watches for the B-device to disconnect, and then connects itself as a peripheral. At this point, normal USB communication occurs, only now the peripheral acts as the host, and vice versa. When the B-device is finished serving as the host, it returns control to the A-device in the same manner.

At the beginning, the B-device is the peripheral and thus has its D+ pull-up resistor enabled. The A-device stops bus traffic, suspending the bus. The B-device USB device logic recognizes the suspend as the first step of transferring control of the bus, disables its D+ pull-up resistor, and enables its USB host logic. After the D+ line falls (is pulled down by the D+ pull-down resistor) to 0, the A-device recognizes the disconnect request and enables its D+ pull-up to connect as a peripheral. The B-device host logic detects the connect request and generates an interrupt to the processor, which then generates traffic to the A-device through the USB host logic.

After the B-device is finished serving as the host, it repeats the transfer process that the A-device performed, suspending the USB bus, waiting for the disconnect interrupt, disabling its USB host logic and enabling its USB device logic, and connecting to the USB bus.

The sequence of events and operations for a device (A or B) to give control of the USB bus to the other device follows:

1. Suspend the USB bus.
2. Wait for the disconnect interrupt.
3. Disable the USB host logic, and enable the USB device logic.
4. Connect to the USB bus through the USB device logic.

The A-device must first send a SetFeature(b_hnp_enable) command to the B-device to enable it to become the host.

The sequence for a device (either A or B) to become the USB host follows:

1. Wait for the suspend interrupt from the USB device logic.
2. Disconnect from the USB bus by disabling the D+ pull-up.
3. Disable the USB device logic and enable the USB host logic.
4. Wait for a connect interrupt from the USB host logic.
5. Generate a USB bus reset and begin enumerating the peripheral.

6.11 USB Charger Detect Controller

The FX3 charger detect controller implements the hardware for the system to interface to the external battery charger. The controller is responsible for battery charger detection and USB ID signal detection and monitoring. This is useful in cases where the FX3 is used in a battery powered device which can detect a charger and initiate charging action by talking to a Power Management IC (PMIC).

6.11.1 USB Charger Detect Register Interface

The FX3 USB charger detect controller registers can be accessed directly from the UIB top-level register interface. Following is the list of these registers.

```
/* FX3 USB Charger Detect Register Interface */
/* These definitions extracted from the Top Level UIB register interface */

    uvint32_t chgdet_ctrl;                      /* 0xe0031800 */
    uvint32_t chgdet_intr;                      /* 0xe0031804 */
    uvint32_t chgdet_intr_mask;                 /* 0xe0031808 */
```

6.11.2 Battery Charger Detection

When enabled by the firmware, the charger detect controller initiates a timed sequence of events to cause the USB PHY to detect the presence of connection to a USB charger. The charger detection is polled by the hardware at a firmware-defined

rate of between 1 ms and 16 ms (charger detect control register field CHG_POLL_INTERVAL) in 1-ms increments. The polling is enabled by the CHG_DET_EN setting, and the result is recorded in CHG_DETECTED.

The CHG_DET_CHANGE interrupt is generated when a change in the CHG_DETECTED field is seen. This interrupt is enabled by CHG_DET_CHANGE_EN in the charger detect interrupt enable register.

6.11.3 USB ID Signal Detection

In the OTG 2.0 specification, the ID line is a simple on/off signal indicating whether the device is connected as a host (A-device, ID = 0) or as a peripheral (B-device, ID = 1). The on state is generated by enabling a pull-up resistor and detecting whether the ID line is floating or terminated with a pull-down resistor.

In the Battery Charging Specification revision 1.1, the functionality of the ID pin is expanded to include measuring resistance values between the ID pin and ground. Using this method a device can detect connection to an ACA (Accessory Charger Adapter). An ACA is a device that enables a single USB port to be attached to a charger and to another device simultaneously. The value of the pull-down resistor also indicates to the device its role (host or peripheral) and if it is a peripheral, whether it is allowed to connect to the USB bus (by enabling a pull-up resistor on D+) and initiate communication with the USB host.

The Battery Charging Specification revision 1.1 defines three resistor values, as shown in Table 6-6.

Table 6-6. USB ACA Device Type

Resistor	Value	Device Type
RID_A_CHG	102-114 k?	USB embedded host (A-device)
RID_B_CHG	171-189 k?	USB OTG peripheral (B-device), may not connect
RID_C_CHG	256-284 k?	USB OTG peripheral (B-device), may connect

When the device is a peripheral but may not connect, the embedded host conserves power by not enabling VBUS, so the peripheral needs to attempt to activate a session by initiating the SRP, as given in the OTG 1.3 Specification.

In addition, Motorola has defined two other resistor values, as shown in Table 6-7.

Table 6-7. USB Charger Motorola-defined Device Type

Resistor	Value	Device Type
200 kΩ ±2%	196-204 kΩ	Motorola USB charger type 0
440 kΩ ±2%	431.2-448.8 kΩ	Motorola USB charger type 1

A value of less than 10 Ω is detected as an A-device without an ACA present, and a value greater than 448.8 kΩ is detected as a B-device without an ACA present.

The FX3 charger detect controller samples and decodes the strength of the pull-down resistor into a digital value, which is subsequently read by the firmware when valid. This ID resistor value is sampled at periodic intervals of 1 to 16 ms, set by OTG_ID_POLL_INTERVAL[3:0] in the charger detect control and configuration register. Setting the OTG_ID_DISABLE bit in the same register may disable polling.

The OTG_ID_CHANGE interrupt is generated when a change in the OTG_ID_VALUE field is seen. This interrupt is enabled by OTG_ID_CHANGE_EN in the charger detect interrupt enable register. Once a charger has been detected, firmware can initiate charging by configuring a PMIC device through one of the available serial interfaces (UART, I2C or SPI).

7. General Programmable Interface II (GPIF II)

EZ-USB FX3 integrates a high-performance interface, GPIF II, which enables functionality similar to but more advanced than the FX2LP GPIF and Slave FIFO interfaces. GPIF II is a programmable state machine that provides the flexibility to design a variety of interfaces to outside entities. The GPIF II interface may function either as a master or a slave in industry-standard or proprietary interfaces. GPIF II can implement both parallel and serial high-bandwidth interfaces. Some popular interfaces that can be implemented with GPIF II are Slave FIFO (asynchronous or synchronous), SRAM, and multiplexed address and data buses (ADMux).

GPIF II is part of the larger FX3 Processor Interface Block (PIB). The PIB acts as the interface to an external processor, FPGA, FIFO, memory, or other high-bandwidth device. This block supports the features required to connect an external processor or device to USB or one of the other FX3 interfaces (SPI, UART, I2C, I2S). For example, GPIF II enables FX3 to connect to an external FPGA, processor, or other device. The thread controller within the PIB and DMA adapter manages the read/write accesses performed over GPIF II to registers and sockets. To understand the data transfer in and out of FX3, it is important to know the terminology specific to FX3; refer FX3 Terminology section in AN75705 - Getting Started with EZ-USB® FX3™.

A GPIF II Designer tool provided with the FX3 SDK installation enables graphical development of GPIF II designs.

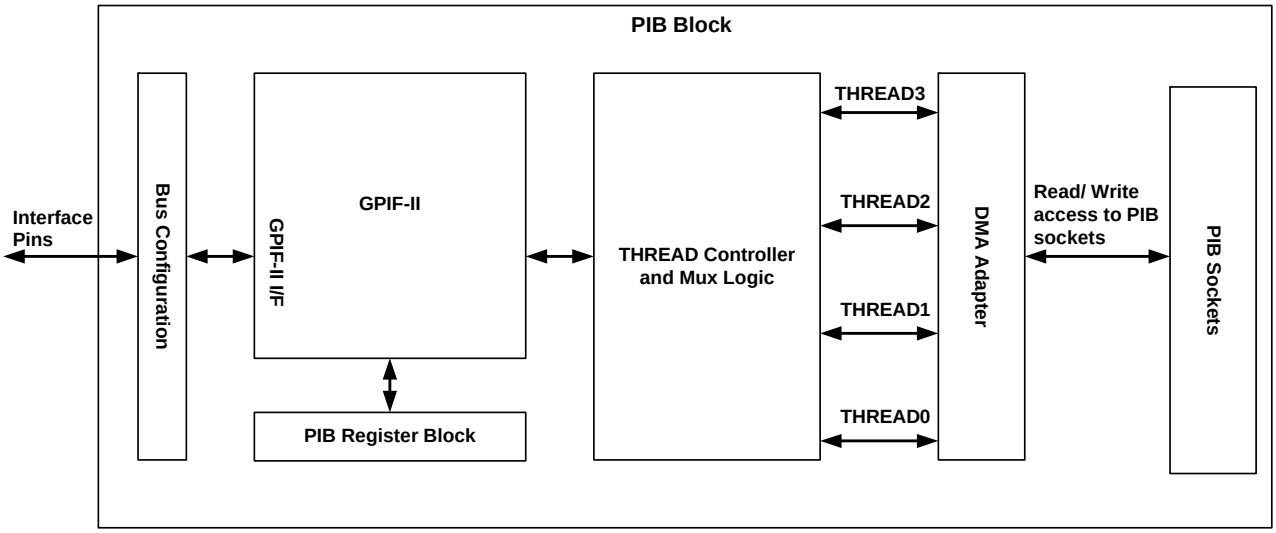
7.1 Features

The following is a summary of the GPIF II features:

- Functions as master or slave
- Provides 256 programmable states
- Implements 8-, 16-, 24- and 32-bit parallel data bus (package-dependent)
- Enables interface frequencies up to 100 MHz
- Supports 14 configurable control pins (strobes, enables, GPIO) when a 32-bit data bus is used
- Supports 16 configurable control pins when a 8-, 16-, or 24-bit data bus is used
- All control pins can be input, output, or bidirectional
- Resources such as counters and comparators
- Wide range of actions and triggers to define the behavior of the state machine

7.2 Block Diagram

Figure 7-1. Block Diagram of FX3 PIB



7.3 Typical GPIF II interface

GPIF II allows the FX3 to connect directly to external peripherals such as FPGAs, image sensors, ADCs, or any other high-bandwidth devices. It provides external pins that can operate as inputs/outputs (CTL[12:0]), a data bus (DQ[31:0]), an interface clock (PCLK), and an interrupt line.

Figure 7-2 shows a block diagram of a typical interface between the FX3 and a peripheral function. Table 7-1 lists the GPIF II interface signals.

Figure 7-2. Block Diagram of Interface Between FX3 and Peripheral

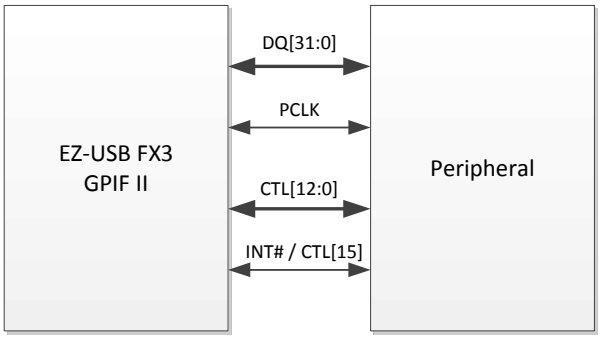


Table 7-1. GPIF II Interface Signals

Pin	IN/OUT	Description
CTL[12:0]	I/O	Programmable control signals
DQ[31:0]	I/O	Bidirectional data bus
PCLK	I/O	Interface clock
INT# /CTL[15]	I/O	Interrupt or control signal

The GPIF II control signals (CTL[12:0]) can be configured as outputs to control the external peripheral device, or as inputs to read the status from an external peripheral device.

The GPIF II interface offers a maximum of 32 bidirectional data lines:

- An 8-bit-wide GPIF II interface (DQ[7:0]) uses pins GPIO[7:0].
- A 16-bit-wide GPIF II interface (DQ[15:0]) uses pins GPIO[15:0].
- A 24-bit-wide GPIF II interface (DQ[23:0]) uses pins GPIO[40:33] and GPIO[15:0].
- A 32-bit-wide GPIF II interface (DQ[31:0]) uses pins GPIO[49:46], GPIO[44:33], and GPIO[15:0].

GPIF II can function as a master or slave. If it is configured as a master, then the GPIOs remaining after allocating for the data bus and control signals can be used as address lines. If GPIF II is configured as a slave, then the maximum number of address lines is limited to eight.

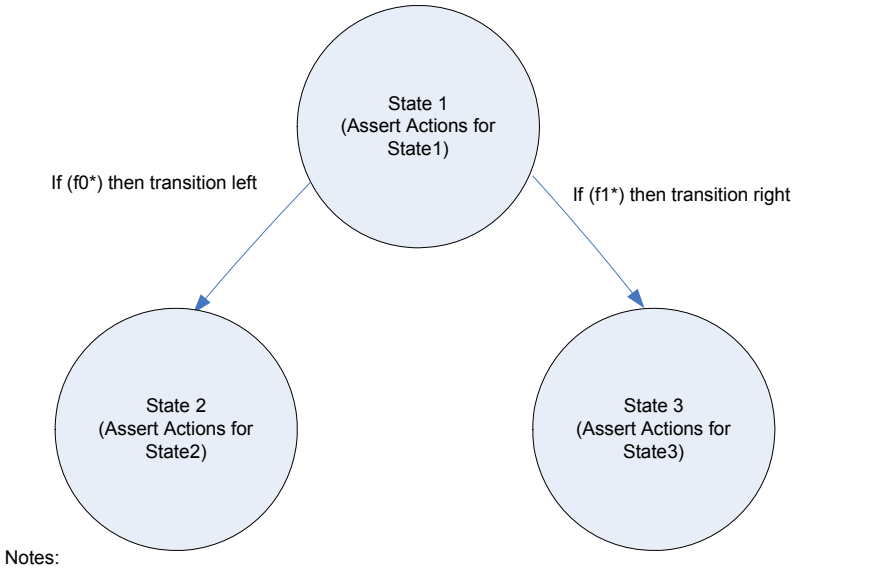
The interface clock, PCLK, can be configured as either an input or an output interface clock for synchronous interfaces to external logic. The maximum frequency supported for the GPIF II interface clock is 100 MHz.

7.4 Functional Overview

The GPIF II interface is configured by creating a GPIF II state machine, and 8 KB of memory space is allocated to store the GPIF II state machine definition. Each state is defined by 32 bytes in (SRAM) memory. These 32 bytes define the properties of a state and the trigger conditions that can cause state (or I/O) transitions. Each state has two transitions out of it. The transition out of a state is determined based on transition conditions. Transitions are caused by both external and internal triggers. Each state is programmed to perform certain actions. The transition conditions are checked on each GPIF II clock edge or after a programmable number of clock cycles.

Figure 7-3 is a simple depiction of the basic structure of a GPIF II state.

Figure 7-3. Structure of a GPIF II State



Notes:
- * f0 and f1 are logical functions of triggers
- f0 is checked first. If f0 = TRUE, then GPIFII transitions from State 1 to State 2. If f0 = FALSE, then f1 is checked.
- If f1 = TRUE, then GPIFII transitions from State 1 to State 3.
- If both f0 and f1 evaluate to FALSE, then GPIFII remains in State 1. At the next clock, the conditions are checked again.

7.4.1 Actions

Each state in a GPIF II state machine can be programmed to perform one or more actions. Actions performed in a state can be programmed to be performed once or repeated in every clock cycle until a state change occurs. Actions can be internal, such as reading or writing to a buffer. They can also be external, such as driving an output high or low. Table 7-2 lists the GPIF II actions.

Table 7-2. GPIF II Actions

No	Action	Description	Incompatibility with Other Actions
1	IN_DATA	Samples data from the data bus and moves to the destination specified. Destination can be Registers, PPRegisters, or Sockets. The data in the register can be accessed using the CY_U3P_PIB_GPIF_INGRESS_DATA(thread_number) macro. The PPRegisters cannot be accessed directly from the PPRegister space. These registers are for controlling the P-port interface in PP mode. The data in Sockets can be controlled or accessed using DMA APIs. Words from Socket can be accessed directly using the firmware API CyU3PGpifReadDataWords().	Cannot be combined with DR_DATA. Cannot be combined with IN_ADDR when the address and data buses are multiplexed.
2	IN_ADDR	Selects a thread or socket after sampling the address word from the bus. If there are two or fewer address bits, the address selects one of the four threads. If the address has more than two bits, the thread or socket can be selected depending upon the The "Address Selecting" parameter is available only when more than two address bits are used. The "PP Register Access Only" option is available only if the width of the address bus is more than or equal to 8	Cannot be combined with IN_DATA when the address and data buses are multiplexed
3	DR_DATA	Drives data onto the bus from the source specified. The source can be Registers, PPRegisters, or Sockets. The data in Register can be sourced using the CY_U3P_PIB_GPIF_EGRESS_DATA(thread_number) macro. The PPRegisters cannot be accessed directly from the PPRegister space. These registers are for controlling the P-port interface in PP mode. The data in Sockets can be controlled or sourced using DMA APIs. Words from Socket can be sourced directly using the firmware API CyU3PGpifWriteDataWords(). DR_DATA has two internal stages: update_bus and pop_data. The update_bus updates the data bus with the word present in the source, and it happens as soon as the state machine enters the particular state. The hardware state machine works using the interface clock (in Synchronous protocol) or the FX3 internal clock (in Asynchronous protocol). In every clock cycle, the update_bus happens regardless of the "Repeat actions until next transition" option in the state setting. Pop_data removes the data word from the source and happens once or multiple times depending on the "Repeat actions until next transition" option in the state setting.	Cannot be combined with IN_DATA. Cannot be combined with DR_ADDR when the address and data buses are multiplexed.
4	DR_ADDR	Drives the value from the specified source to address bus. The source can be Registers, AddressCounter, or ThreadSocket. The address in Registers can be sourced using the CY_U3P_PIB_GPIF_EGRESS_ADDRESS(thread_number) macro. The data in Sockets can be controlled or sourced using the DMA APIs. Words from Socket can be sourced directly using the firmware API CyU3PGpifWriteDataWords().An AddressCounter can be used to source the address.	Cannot be combined with DR_DATA when the address and data buses are multiplexed.
5	COMMIT	Commit or wraps up the buffer associated with the selected ingress thread and socket. The buffer is transferred to the consumer side of the pipe. Typically, this is used to wrap up the buffer in between a transaction prematurely.	None
6	DR_GPIO	Drives the GPIO signal to HIGH/LOW or toggles the value immediately or after a delay of one clock cycle. The interface observes output latency between the time the DR_GPIO action is called and the change in the GPIO signal in the interface. The "Repeat actions until next transition" in the state setting has no effect on the behavior of the DR_GPIO action. This action is repeated for every clock cycle (the clock being the interface clock or the FX3 internal clock).	None
7	LD_ADDR_COUNT	Loads the counter with initial settings. The initial settings are loaded while starting the state machine. This value needs to be the same in all states within a given state machine diagram. Multiple values are not allowed. These settings can be overridden using firmware APIs. When the counter reaches the limit, the hardware sets the ADDR_CNT_HIT internal trigger signal. This action is generally used with COUNT_ADDR and the ADDR_CNT_HIT trigger. The count value can also be programmed by the firmware application using CyU3PGpifInitAddrCounter(). The value programmed into the register at the time of execution of the state machine will be used.	Cannot be combined with COUNT_ADDR.
8	LD_DATA_COUNT	Loads the counter with initial settings. The initial settings are loaded while starting the state machine. This value needs to be the same in all states within a given state machine diagram. Multiple values are not allowed. These settings can be overridden using firmware APIs. When the counter reaches the limit, the hardware sets the DATA_CNT_HIT internal trigger signal. This action is generally used with COUNT_DATA and the DATA_CNT_HIT trigger. The count value can also be programmed by the firmware application using CyU3PGpifInitDataCounter(). The value programmed into the register at the time of execution of the state machine will be used.	Cannot be combined with COUNT_DATA.
9	LD_CTRL_COUNT	Loads the counter with initial settings. The initial settings are loaded while starting the state machine. This value needs to be the same in all states within a given state machine diagram. Multiple values are not allowed. These settings can be overridden using firmware APIs. When the counter reaches the limit, the hardware sets the CTRL_CNT_HIT internal trigger signal. This action is generally used with COUNT_CTRL and the CTRL_CNT_HIT trigger. The count value can also be programmed by the firmware application using CyU3PGpifInitCtrlCounter(). The value programmed into the register at the time of execution of the state machine will be used.	Cannot be combined with COUNT_CTRL.

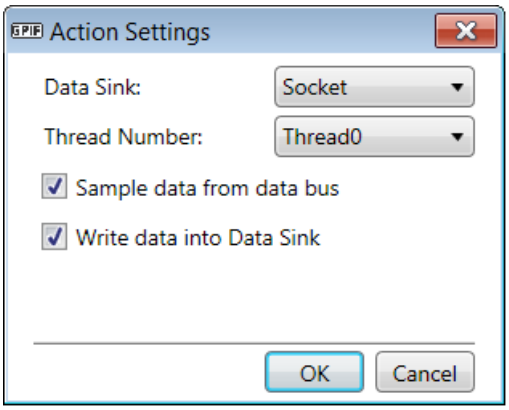
No	Action	Description	Incompatibility with Other Actions
10	COUNT_ADDR	Increments/decrements the address counter value each time the state is visited. The hardware produces an ADDR_CNT_HIT event upon reaching the limit value specified during LD_ADDR_COUNT.	Cannot be combined with LD_ADDR_COUNT.
11	COUNT_DATA	Increments/decrements the data counter value each time the state is visited. The hardware produces a DATA_CNT_HIT event upon reaching the limit value specified during LD_DATA_COUNT.	Cannot be combined with LD_DATA_COUNT.
12	COUNT_CTRL	Increments/decrements the control counter value each time the state is visited. The hardware produces a CTRL_CNT_HIT event upon reaching the limit value specified during LD_CTRL_COUNT.	Cannot be combined with LD_CTRL_COUNT.
13	CMP_ADDR	Compares the sampled address with the specified comparison value or detects any change in the address value. "Repeat actions until next transition" in the state setting has no effect on the behavior of the CMP_ADDR action. This action is repeated for every clock cycle (the clock being the interface clock or the FX3 internal clock)..	Cannot be combined with IN_DATA when the address and data buses are multiplexed.
14	CMP_DATA	Compares the sampled address with the specified comparison value or detects any change in the data value. "Repeat actions until next transition" in the state setting has no effect on the behavior of the CMP_DATA action. This action is repeated for every clock cycle (the clock being the interface clock or the FX3 internal clock)..	Cannot be combined with DR_DATA. Cannot be combined with IN_ADDR when the address and data buses are multiplexed.
15	CMP_CTRL	Compares the sampled control word with the specified comparison value or detects any change in the control value. "Repeat actions until next transition" in the state setting has no effect on the behavior of the CMP_CTRL action. This action is repeated for every clock cycle (the clock being the interface clock or the FX3 internal clock)..	None
16	INTR_CPU	Generates a CYU3P_GPIF_EVT_SM_INTERRUPT event to the FX3 CPU, which can be serviced by the registered interrupt callback in the firmware.	None
17	INTR_HOST	Drives the INTR pin in the P-port interface to indicate the presence of an interrupt signal to the external processor. The INTR pin should be connected to the external processor.	None
18	DR_DRQ	Drives the DRQ pin in the P-port interface to indicate the presence of a data request to the external processor. The signal should be connected to the external processor on which the DRQ is enabled.	None

7.4.1.1 Action - IN_DATA

The IN_DATA action samples data from the data bus and moves it to the specified destination. The destination can be the DMA channel or the firmware application. See the firmware API CyU3PGpifReadDataWords(). Note that the option for selecting the destination thread is available only when the destination is the DMA channel with the addressing mode selected as Thread selected by State machine (Number of address lines =0).

It is possible to latch only the data from the data bus and not to store it in the selected destination or to store previously latched data in the selected destination. These options are made available to satisfy certain protocols when the data on the bus may lead a strobe signal that indicates data availability.

Figure 7-4. IN_DATA Action Settings Dialog Box



The following parameters are associated with this action:

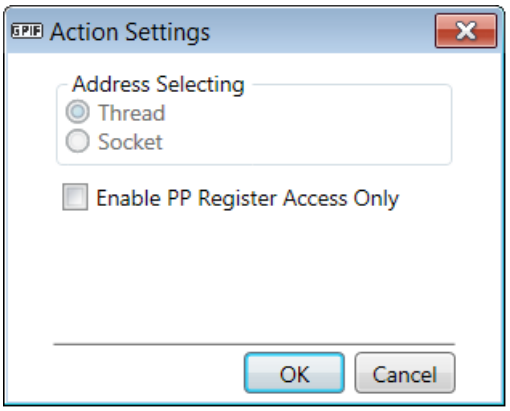
- Data Sink-Register/ Socket/ PPRegister
- Thread Number-Thread number with which the data sink is associated. Selectable from Thread0 to 3 (This feature is available when there is no address line to select the thread number or when master mode is enabled.)
- Sample data from data bus-This option, when checked, samples data from the data bus, but does not push it in to the specified data sink.
- Write data into Data Sink-This option is selected when the user wants to push the sampled data into the specified data sink.

7.4.1.2 Action - IN_ADDR

The IN_ADDR action causes the GPIF II hardware to sample the value from the address bus and use it to select a DMA thread or a socket. Refer to the FX3 Terminology section in AN75705 - Getting started with EZ-USB FX3 for more details on DMA threads and sockets.

When the address bus width is less than or equal to 2 bits, the address can select only a DMA thread. If the address is between 3 and 5 bits wide, it can select either a DMA thread or a specific socket. The Address Selecting parameter shown in Figure 7-4 is used to make this selection.

Figure 7-5. IN_ADDR Action Settings Dialog Box



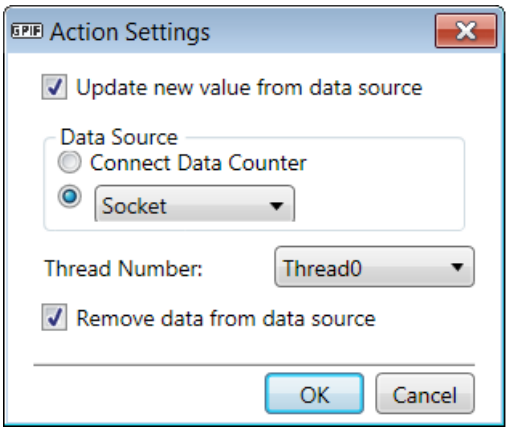
The following parameter is associated with this action:

- Address Selecting-Thread or Socket

7.4.1.3 Action - DR_DATA

The DR_DATA action drives data on the bus from the source specified. The source can be the DMA channel or the firmware application. See the firmware API CyU3PGpifWriteDataWords(). Note that the option for selecting the source as the thread number is available only when the source is the DMA channel with the addressing mode selected as Thread selected by State machine (Number of address lines =0).

Figure 7-6. DR_DATA Action Settings Dialog Box



The following parameters are associated with this action:

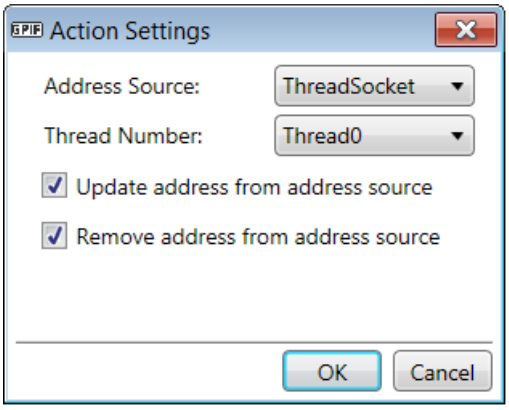
- Update new value from data source-This option updates the data bus with the data word present in the source and removes the word from the specified source.
- Data Source-This option selects between data counter and register/Socket/PPRegister.
- Thread Number-Thread number with which the data source is associated. Selectable from Thread0 to 3. (This feature is available when there is no address line to select the thread number.)
- Remove data from data source-When this option is disabled, the data source continues to point to the same data.

Note: The Update new value from data source flag overrides the Repeat actions until next transition flag in the State Settings dialog box. This means if the Update new value from data source option is selected, data will be updated on the bus in every clock cycle when FX3 is in that state, even if the Repeat actions until next transition flag is not selected.

7.4.1.4 Action - DR_ADDR

The DR_ADDR action drives the value from the specified source to the address bus. The source can be the DMA channel or the firmware application.

Figure 7-7. DR_ADDR Action Settings Dialog Box



The following parameters are associated with this action:

- Address Source-Register/AddressCounter/ThreadSocket.
- Thread Number-Thread0 to 3 (available only when number of address bits is set to 0).

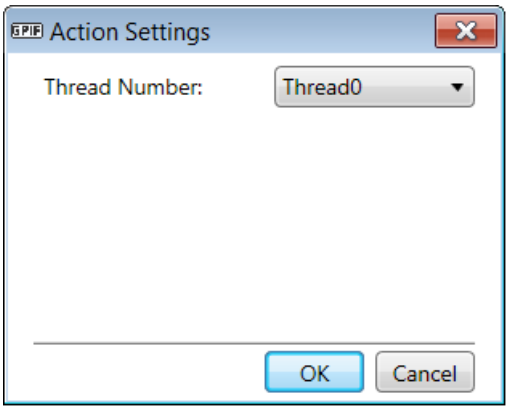
7.4.1.5 Action - COMMIT

The COMMIT action commits the data packet/buffer on the selected DMA channel.

Let say in your application, the data is transferred from GPIF II to USB and a DMA channel is created by allocating a DMA buffer. When you use COMMIT action in the GPIF II state machine, the control of DMA buffer will be given to USB block. That means the data in the DMA buffer is ready to be transmitted to USB host.

This action is typically used to force "buffer/packet end" using the state machine. For committing short buffers, this action should be used along with the IN_DATA action. If the buffer is partially filled and the COMMIT action is performed without the IN_DATA action, a zero-length buffer will be committed in addition to the partially filled buffer.

Figure 7-8. COMMIT Action Settings Dialog Box



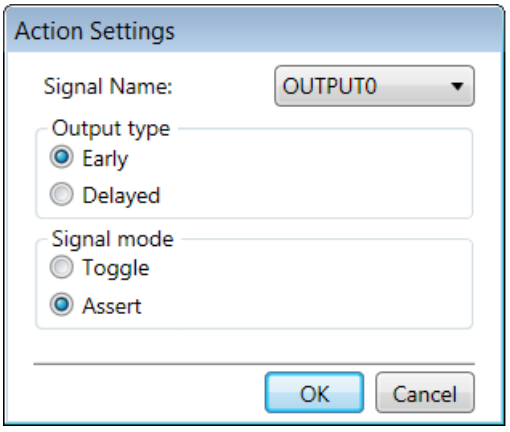
The following parameters are associated with this action:

- Thread Number-Thread 0 to 3 (available only when the number of address bits is configured to 0 or master mode is enabled).

7.4.1.6 Action - DR_GPIO

The DR_GPIO action drives a GPIO signal to HIGH/LOW or toggles the value after "Assertion Delay" cycles. In synchronous mode, the delay can be between two and three cycles, and in an asynchronous (no clock) system the delay can be 25 ns or 30 ns. The GPIO driven using the action is deasserted during the transition to the next state.

Figure 7-9. DR_GPIO Action Settings Dialog Box



The following parameters are associated with this action:

- Signal Name-A user-defined alphanumeric string to indicate the signal can be entered here. This name will appear on the state machine canvas.
- Output type: Early or Delayed-This parameter controls the delay of the output signal being driven. The delay will be 25 ns in asynchronous mode or two clock cycles in synchronous mode when the Early option is not selected. The delay will be 30 ns in asynchronous mode or three clock cycles in synchronous mode when the Delayed option is selected.
- Signal mode-In toggle mode, the signal value complements; in assert mode, the signal value is driven per the polarity of the signal. The polarity of the signal is configured in the Interface Definition window.

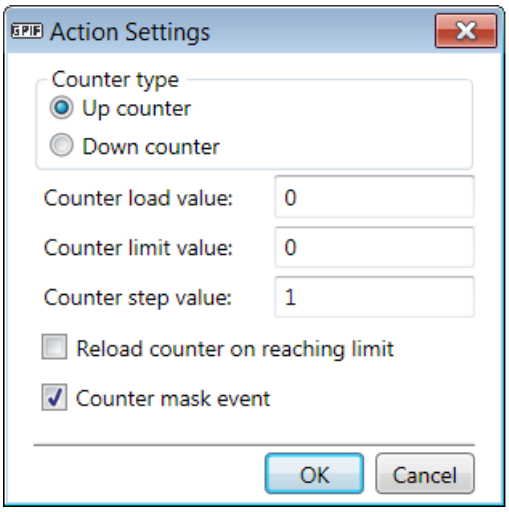
Notes

1. The delayed output and signal mode settings are global for each signal and cannot be changed every time the action is used in a different state. The tool will generate a configuration file that corresponds to the last settings that were made for each output signal.
2. "Repeat actions until next transition" in the state setting has no effect on the behavior of the DR_GPIO action. This action is repeated for every clock cycle (the clock being the interface clock or the FX3 internal clock).

7.4.1.7 Action - LD_ADDR_COUNT

This action loads the counter settings. Settings are loaded when the state machine starts. This value needs to be the same in all states within a given state machine diagram.

Figure 7-10. LD_ADDR_COUNT Action Settings Dialog Box



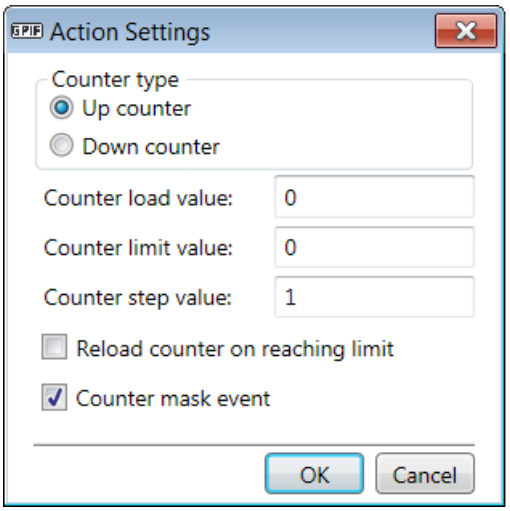
The following parameters are associated with this action:

- Counter type: Up counter/Down counter-The counter can be configured to count in ascending/descending order.
- Counter load value-Initial count loaded when this action is performed.
- Counter limit value-The count value at which the event should be generated.
- Reload counter on reaching limit-The count load value can be reloaded when the counter limit value is hit by selecting this parameter.
- Counter mask event-If this option is not selected, a firmware event is generated when the counter limit is reached.
- Counter step value-The counter step value that should be added/subtracted each time the COUNT_ADDR action is used.

7.4.1.8 Action - LD_DATA_COUNT

This action loads the counter with initial settings. The initial settings are loaded when the state machine starts. This value needs to be the same in all states within a given state machine diagram.

Figure 7-11. LD_DATA_COUNT Action Settings Dialog Box



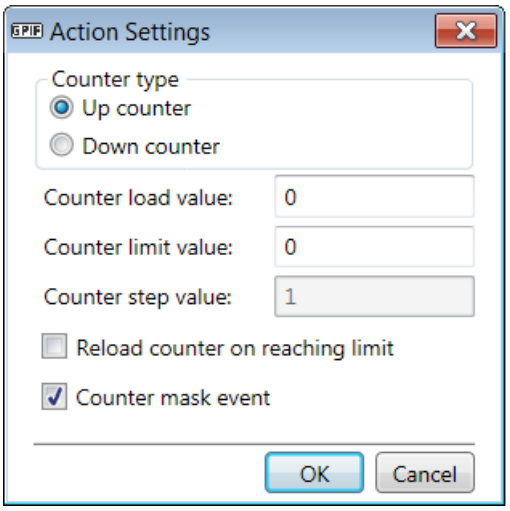
The following parameters are associated with this action:

- Counter type: Up counter/Down counter-The counter can be configured to count in ascending/descending order.
- Counter load value-The initial count loaded when this action is performed.
- Counter limit value-The count value at which the event should be generated.
- Reload counter on reaching limit-The count load value can be reloaded when the count limit value is hit by selecting this parameter.
- Counter mask event-If this option is not selected, a firmware event is generated when the counter limit is reached.
- Counter step value-The counter step value that should be added/subtracted each time the COUNT_DATA action is used.

7.4.1.9 Action - LD_CTRL_COUNT

This action loads the counter with initial settings. The initial settings are loaded when the state machine starts. This value needs to be same in all states within a given state machine diagram. Multiple values are not allowed.

Figure 7-12. LD_CTRL_COUNT Action Settings Dialog Box



The following parameters are associated with this action:

- Counter type: Up counter/Down counter-The counter can be configured to count in ascending/descending order.
- Counter load value-The initial count loaded when this action is performed.
- Counter limit value-The count value at which the event should be generated.
- Reload counter on reaching limit-The count load value can be reloaded when the count limit value is hit by selecting this parameter.
- Counter mask event-If this option is not selected, a firmware event is generated when the counter limit is reached.
- Counter step value-The counter step value that should be added/subtracted each time the COUNT_CTRL action is used.

7.4.1.10 Action - COUNT_ADDR

This action updates the address counter value with the step value configured through the LD_ADDR_COUNT action. The ADDR_CNT_HIT trigger will become true if this update results in the count reaching the specified limit.

Note that there are no parameters associated with this action.

7.4.1.11 Action - COUNT_DATA

This action updates the data counter value with the step value configured through the LD_DATA_COUNT action. The DATA_CNT_HIT trigger will become true if this update results in the count reaching the specified limit.

Note that there are no parameters associated with this action.

7.4.1.12 Action - COUNT_CTRL

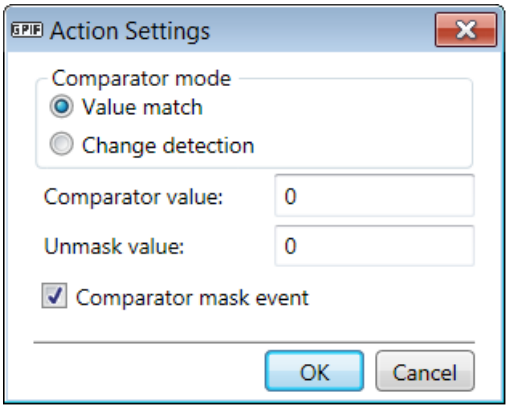
This action updates the control counter value with the step value configured through the LD_CTRL_COUNT action. The CTRL_CNT_HIT trigger will become true if this update results in the count reaching the specified limit.

Note that there are no parameters associated with this action.

7.4.1.13 Action - CMP_ADDR

This action compares the address sampled with the specified comparison value.

Figure 7-13. CMP_ADDR Action Settings Dialog Box



The following parameters are associated with this action:

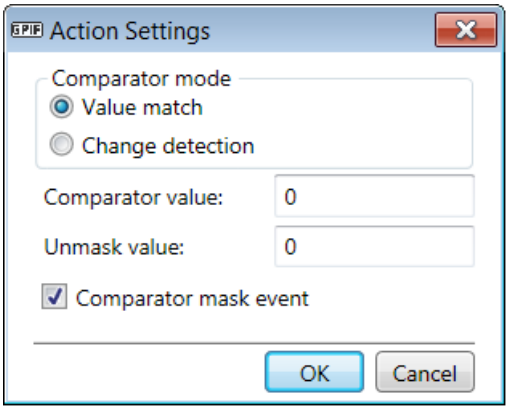
- Comparator mode-In Value match mode, compares the current address value with the comparator value. In Change detection mode, any change in the address value will cause a comparator match.
- Unmask value-Use this value to unmask the bits to be compared.
- Comparison value-Value against which to compare the address.
- Comparator mask event-When you deselect this parameter, it will cause an event to be generated to the firmware application on a comparator match.

Note The "Repeat actions until next transition" in the state setting has no effect on the behavior of the CMP_ADDR action. This action is repeated for every clock cycle (the clock being the interface clock or the FX3 internal clock).

7.4.1.14 Action - CMP_DATA

This action compares the data sampled with the specified comparison value.

Figure 7-14. CMP_DATA Action Settings Dialog Box



The following parameters are associated with this action:

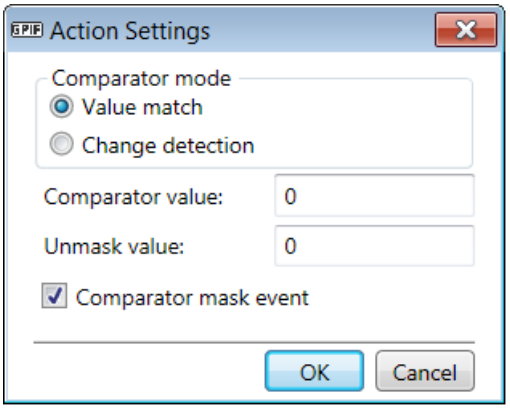
- Comparator mode-In Value match mode, compares the current data value with the comparator value. In Change detection mode, any change in the data value will cause a comparator match.
- Unmask value-Use this value to unmask the bits to be compared.
- Comparator value-Value to compare the data against.
- Comparator mask event-When you deselect this parameter, it will cause an event to be generated to the firmware application on a comparator match.

Note: The "Repeat actions until next transition" in the state setting has no effect on the behavior of the CMP_DATA action. This action is repeated for every clock cycle (the clock being the interface clock or the FX3 internal clock).

7.4.1.15 Action - CMP_CTRL

This action compares the control bits with the specified comparison value.

Figure 7-15. CMP_CTRL Action Settings Dialog Box



The following parameters are associated with this action:

- Comparator mode-In Value match mode, compares the current control signal values with the comparator value. In Change detection mode, any change in the unmasked control signals will cause a comparator match.
- Unmask value-Use this value to unmask the bits to be compared.
- Comparison value-Value against which to compare the control signals.
- Comparator mask event-When you deselect this parameter, it will cause an event to be generated to the firmware application on a comparator match.

Note: "Repeat actions until next transition" in the state setting has no effect on the behavior of the CMP_CTRL action. This action is repeated for every clock cycle (the clock being the interface clock or the FX3 internal clock).

7.4.1.16 Action - INTR_CPU

This action interrupts the on-chip CPU to generate a CYU3P_GPIF_EVT_SM_INTERRUPT event, which is to be handled by the firmware application.

No parameters associated with this action.

7.4.1.17 Action - INTR_HOST

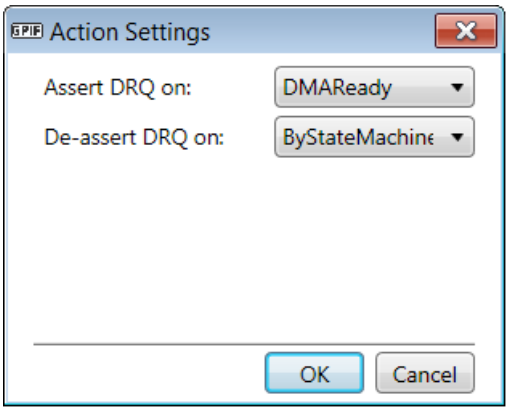
This action interrupts the external processor by driving the INTR pin on the P-port.

No parameters associated with this action.

7.4.1.18 Action - DR_DRQ

This action drives the DRQ pin in the P-port interface to indicate the presence of a data request to the external processor.

Figure 7-16. DR_DRQ Action Settings Dialog Box



The following parameters are associated with this action:

DRQ value-Asserted or deasserted

- Assert DRQ on-The DRQ signal can be asserted using any of these options:
- From the DMA engine
- On assertion from the external processor on the DACK pin of FX3
- On deassertion of DACK from the external processor on the DACK pin of FX3
- From the state machine using the DR_DRQ action
- De-assert DRQ on-The DRQ can be deasserted using any of these options:
- On assertion from the external processor on the DACK pin of FX3
- On deassertion from the external processor on the DACK pin of FX3
- From the state machine using the DR_DRQ action

7.4.2 Triggers

Triggers are signals that cause state transitions to occur. They can be:

External: Control signals driven by the external device

Internal: Internal signals asserted due to hardware or firmware events

Table 7-3 captures events generated as a result of the GPIF II actions.

Table 7-3. Event Generated by GPIF II Actions

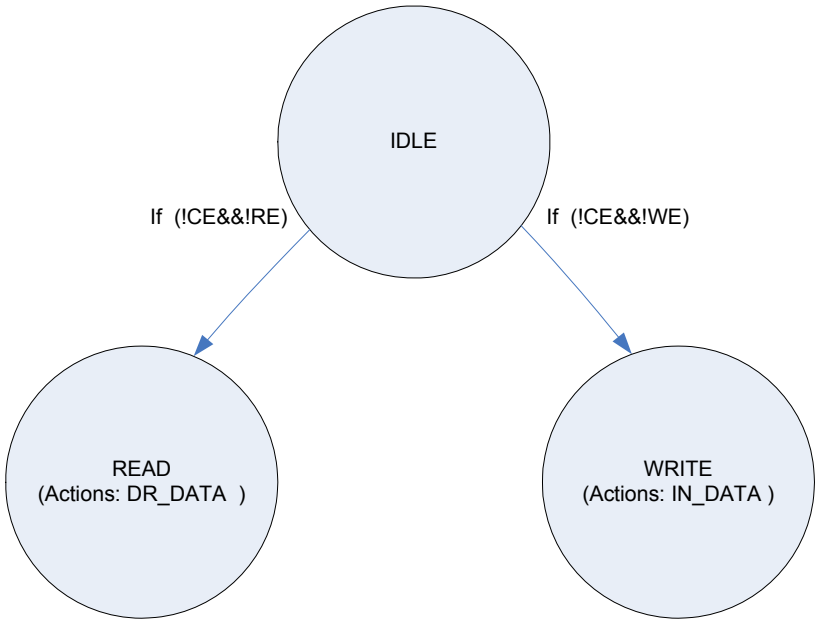
No.	Event	Action Causing the Event	Description
1	Input signal name	None; this is an external event	The name associated with any GPIO configured as an input can be used as a trigger in a transition equation.
2	FW_TRG	Firmware application	This trigger can be generated from the firmware using the firmware API CyU3PGpifControlSWInput().
3	INTR_PENDING	Interrupt to CPU	This is true only when the CPU/host has yet to read the previously raised interrupt.
4	CTRL_CNT_HIT	COUNT_CTRL	This trigger is true when the count specified is reached. It is generated as a result of the COUNT_CTRL action.
5	ADDR_CNT_HIT	COUNT_ADDR	This trigger is true when the count specified is reached. It is generated as a result of COUNT_ADDR action.
7	DATA_CNT_HIT	COUNT_DATA	This trigger is true when the count specified is reached. It is generated as a result of COUNT_DATA action
8	CTRL_CMP_MATCH	CMP_CTRL	This trigger is true when the control pattern and mask match the current GPIO. It is generated only when the CMP_CTRL action is specified in the parent state.
9	DATA_CMP_MATCH	CMP_DATA	This trigger is true when the data pattern and mask match the current data. It is generated only when the CMP_DATA action is specified in the parent state.
10	ADDR_CMP_MATCH	CMP_ADDR	This trigger is true when the address pattern and mask match the current address read. It is generated only when the CMP_ADDR action is specified in the parent state.
11	DMA_RDY_CT, DMA_RDY_TH0, DMA_RDY_TH1, DMA_RDY_TH2, DMA_RDY_TH3	IN_DATA DR_DATA	This trigger is true when the DMA is ready to send or receive data.
12	DMA_WM_CT, DMA_WM_TH0, DMA_WM_TH1, DMA_WM_TH2, DMA_WM_TH3	IN_DATA	This trigger is true when the active DMA thread crosses the transferred data above the watermark.
13	DMA_RDY_ADDR	DR_ADDR	This trigger is true when the DMA is ready to send or receive an address.
14	IN_REG_CR_VALID, IN_REG0_VALID, IN_REG1_VALID, IN_REG2_VALID, IN_REG3_VALID	IN_DATA	This trigger is true when the data in the input register (INGRESS_DATA_REGISTER) is valid. It is set when GPIF II writes data into the INGRESS_DATA_REGISTER using the IN_DATA action. It is cleared by the firmware by setting the IN_DATA_VALID field in the GPIF_DATA_CTRL register.
15	OUT_REG_CT_VALID, OUT_REG0_VALID, OUT_REG1_VALID, OUT_REG2_VALID, OUT_REG3_VALID	DR_DATA	This trigger is true when the data in the output register (EGRESS_DATA_REGISTER) is valid. The firmware needs to enable this trigger by setting the EG_DATA_VALID bit in the GPIF_DATA_CTRL register. This trigger will be cleared when the data in the register has been transmitted by the GPIF II as a result of the DR_DATA action.
16	OUT_ADDR_VALID	DR_ADDR	This trigger is true when the address in the output register (EGRESS_ADDRESS_REGISTER) is valid. It is done when the firmware sets the EG_ADDR_VALID bit in the GPIF_DATA_CTRL register.

7.4.3 Transition Conditions

Transition conditions determine the transition out of a state and into another. The transition conditions are Boolean logic functions of the trigger signals. A simple example of actions, triggers, and transition conditions is shown in Figure 7-17, where the three GPIF II states are IDLE, READ, and WRITE. No actions are asserted in the IDLE state. The CE, RE, and WE signals are the trigger signals. They are the control input signals to GPIF II and are driven by an external device. The transition conditions are formed with these signals. The left transition condition is (!CE&&!RE) (a read operation), and the right

transition condition is (!CE&&!WE) (a write operation). If the left transition condition evaluates to TRUE, then GPIF II transitions from the IDLE state to the READ state. If the left transition condition evaluates to FALSE, then the right transition condition is checked. If the right transition condition evaluates to TRUE, then GPIF II transitions from the IDLE state to the WRITE state. If neither left nor right conditions evaluate to TRUE, then GPIF II remains in IDLE. Note that the first test resolves the contention if both f0 and f1 are true.

Figure 7-17. Example State Transitions with Actions



* f0 and f1 are logical functions of trigger signals

7.4.4 GPIF II Designer Tool

An essential part of working with FX3 GPIF II is the GPIF II Designer tool. The GPIF II Designer tool provides a convenient graphical user interface (GUI) that allows you to define GPIF II state machines in graphical form. The various GPIF II triggers and actions are available in an easy-to-use manner, allowing simple addition to the states in the diagram. The tool converts the user graphical design into a C header file that can be integrated with the firmware application code using the firmware API framework. The tool also generates warnings and error messages during the graphical entry process. You can create your own GPIF II designs, but the GPIF II Designer tool also provides a set of "canned" designs for popular interfaces. Documentation of these interfaces, describing the protocol along with timing diagrams, is available when you open an example project in GPIF II Designer. As part of its initialization, FX3 firmware copies the settings in the .h file into the appropriate GPIFII registers.

You can also make minor customizations to these designs to suit the target environment. A detailed description of the tool and its use is provided in the GPIF II Designer User Guide, which is available when you install the tool with the EZ-USB FX3 Software Development Kit.

7.4.5 GPIF II Hardware Resources

7.4.5.1 Comparators

GPIF II includes an address comparator, a data comparator, and a control comparator. You can set the value of the comparator and a mask that does a bit-by-bit enable. A trigger is available for these comparators that asserts when the comparator meets the set limit.

7.4.5.2 Counters

GPIF II has three counters for control, address, and data. The reset value and count limit value can be programmed for each counter. The counter value is incremented by using the actions provided in the GPIF II Designer tool. A counter reaching its programmed limit provides one of the available GPIF II trigger signals.

7.4.5.3 GPIF II Interrupt

Any GPIF II state can interrupt the CPU by asserting the INTR_CPU action provided in the GPIF II Designer tool. GPIF II state machine execution continues after interrupting the CPU.

7.4.6 Threads and Sockets

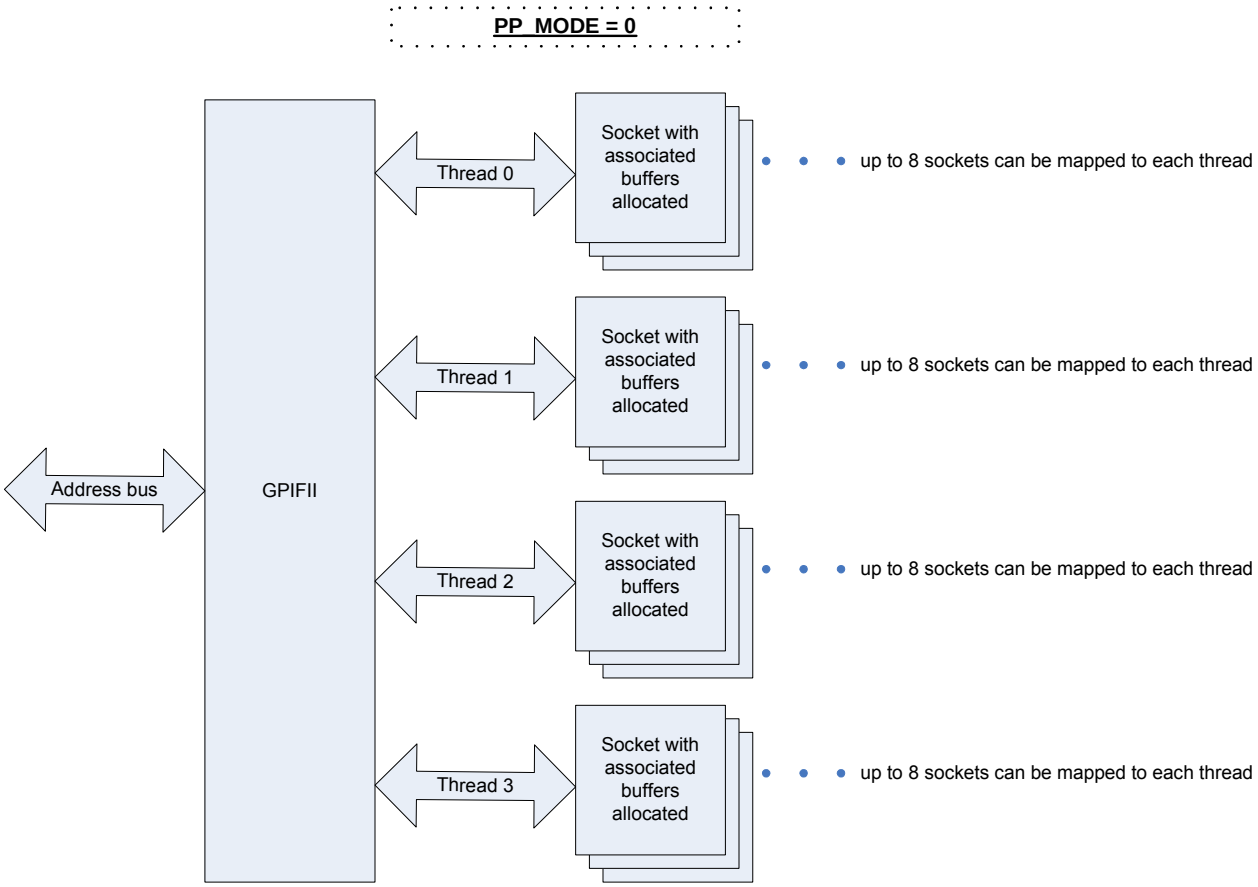
A thread in the GPIF II context is a physical, hardware, data channel. There are four physical threads, each with an independent controller. A socket, on the other hand, is very similar to an endpoint in the USB context. Each socket has buffers assigned to it during initialization. A particular thread is mapped to a particular socket during initialization. This mapping can be changed later during execution. A GPIF II thread is basically a pipe through which the data on the GPIF II interface is connected to the socket (and hence buffer), where it is required to be written to or read from. Though there are four independent threads, GPIF II can access only one of them at a time. Which thread is to be accessed can be specified using the address bus on the GPIF II interface (two address bits select one of the four threads).

7.4.6.1 Difference Between PP_MODE=0 and PP_MODE=1

PP_MODE is a bit field of GPIF II Configuration register (GPIF_CONFIG). This decides the two modes of GPIF II. One mode, PP_MODE=0, is designed for peripherals that use hardware signaling to switch end points or sockets. In this mode, the 32 sockets are mapped to 4 threads in a modulo-4 fashion. The other mode, called PP_MODE=1, is designed for interfacing with chips that have DMA engines that transfer large data and switch end points through the accessing processor port (PP) registers. In this mode, all the sockets are connected to a single thread inside the GPIF II.

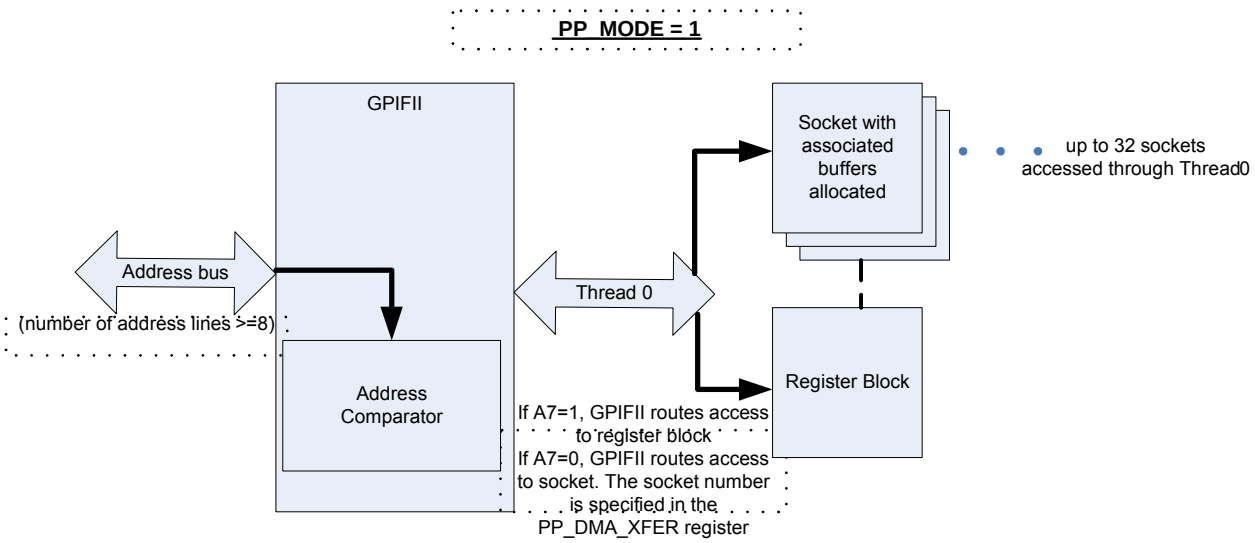
PP_MODE=0 enables all four threads described in the previous section, as shown in Figure 7-18.

Figure 7-18. GPIF II - PP_MODE=0



The second mode is PP_MODE=1. In this mode, while the four independent threads still exist, only one thread (Thread0) is used, through which all sockets are accessed. In this mode, the external processor must first write to a few FX3 registers over the GPIF II interface. These registers have 8-bit addresses; hence, in PP_MODE=1, having at least an 8-bit-wide address bus on the interface is a requirement. By writing certain FX3 registers (PP_DMA_XFER and PP_DMA_SIZE), the socket number and amount of data to be transferred are specified. Then when data access begins, the data is automatically routed through thread 0, to and from whichever socket number was specified earlier in the register, as shown in Figure 7-19.

Figure 7-19. GPIF II - PP_MODE=1



7.4.7 Addressing

7.4.7.1 Number of Address Lines

The number of address lines is an important factor in determining the addressing scheme. When the number is eight or more, the GPIF II Designer tool automatically sets a field, "PP_MODE," which enables an external processor to access some PIB registers. An example of an interface with eight address lines is the SRAM interface. In this case, the external processor can directly program the socket and access direction and byte count through the PP_DMA_XFER and PP_DMA_SIZE registers. When the interface has fewer than eight address lines, access to P-port registers is not possible, and the GPIF II Designer tool clears the PP_MODE field to 0. For example, this is the case for the Slave FIFO interface, in which, the mapping of sockets to threads is important, as explained in the following section.

7.4.7.2 Assigning Sockets to Threads

FX3 provides up to four physical threads for data transfer over GPIF II. At a time, any one socket may be mapped to a thread. For example, in the Slave FIFO implementation, the address signals A0:A1 on the interface indicate the thread to be accessed. The FX3 DMA fabric will then route the data to the socket mapped to that thread. So if socket 2 is mapped to thread 2, when A0:A1 = 2, thread 2 is accessed, and any data that is transferred over thread 2 will be routed to socket 2.

Note: In PP_MODE=1, only thread 0 is used. The socket number programmed in the DMA_XFER register gets mapped to thread 0. In PP_MODE=0, all four threads are accessible. Even though only four physical threads are available, an address mapping mechanism can be used to access more than four socket addresses. Refer to application note AN68829 - Slave FIFO Interface for EZ-USB FX3: 5-Bit Address Mode.

7.4.7.3 Addressing Methods

A socket may be addressed directly in PP_MODE =1 by the external processor programming the PP_DMA_XFER register. Note that when PP_MODE =1, the GPIF II hardware decodes the address based on address bit A7. If A7=1, GPIF II interprets the access as a register access and performs a read or write to the register address specified by A[7:0]. If A7=0, then GPIF II interprets the access to be a socket access and performs a read or write operation to the socket number specified in the PP_DMA_XFER register.

In PP_MODE =0, when the external processor/device does not have access to the PP registers, the thread to be accessed may be addressed in the following ways. Note that in this case, the corresponding GPIF_THREAD_CONFIG(x) register must be programmed using the API with the active socket to be accessed.

- Thread_in_state: Thread address from GPIF II state

- A0, A1: Thread address specified by A0:A1
- Data_thread: Thread address specified by the data_thread field of the GPIF_AD_CONFIG register

7.4.8 Async/Sync

GPIF II supports both asynchronous and synchronous interfaces, configured by programming the "sync" field of the GPIF_CONFIG register. For synchronous interfaces, the interface clock may be provided by an external source or the FX3 internal clock.

7.4.9 Configuration of Flags

To enable flow control on the interface, flags may be configured as empty, full, partially empty, or partially full signals. These are not controlled by GPIF II states; rather, they are controlled directly by the DMA hardware engine internal to FX3. Flags are associated with specific threads and hence indicate the status of the socket currently mapped to that thread. Flags indicate empty or full, based on the direction of the socket (configured during socket initialization). So, the flag indicates an empty or not empty status if data is being read out of the socket. It indicates a full or not full status if data is being written into the socket. The GPIF II Designer tool allows for the configuration of flags, which are typically used in Slave FIFO mode. For a partial flag, the watermark level must be programmed in the corresponding GPIF_THREAD_CONFIG register. For information about the Slave FIFO interface and flag usage, refer to the application note AN65974 - Designing with the EZ-USB FX3 Slave FIFO Interface.

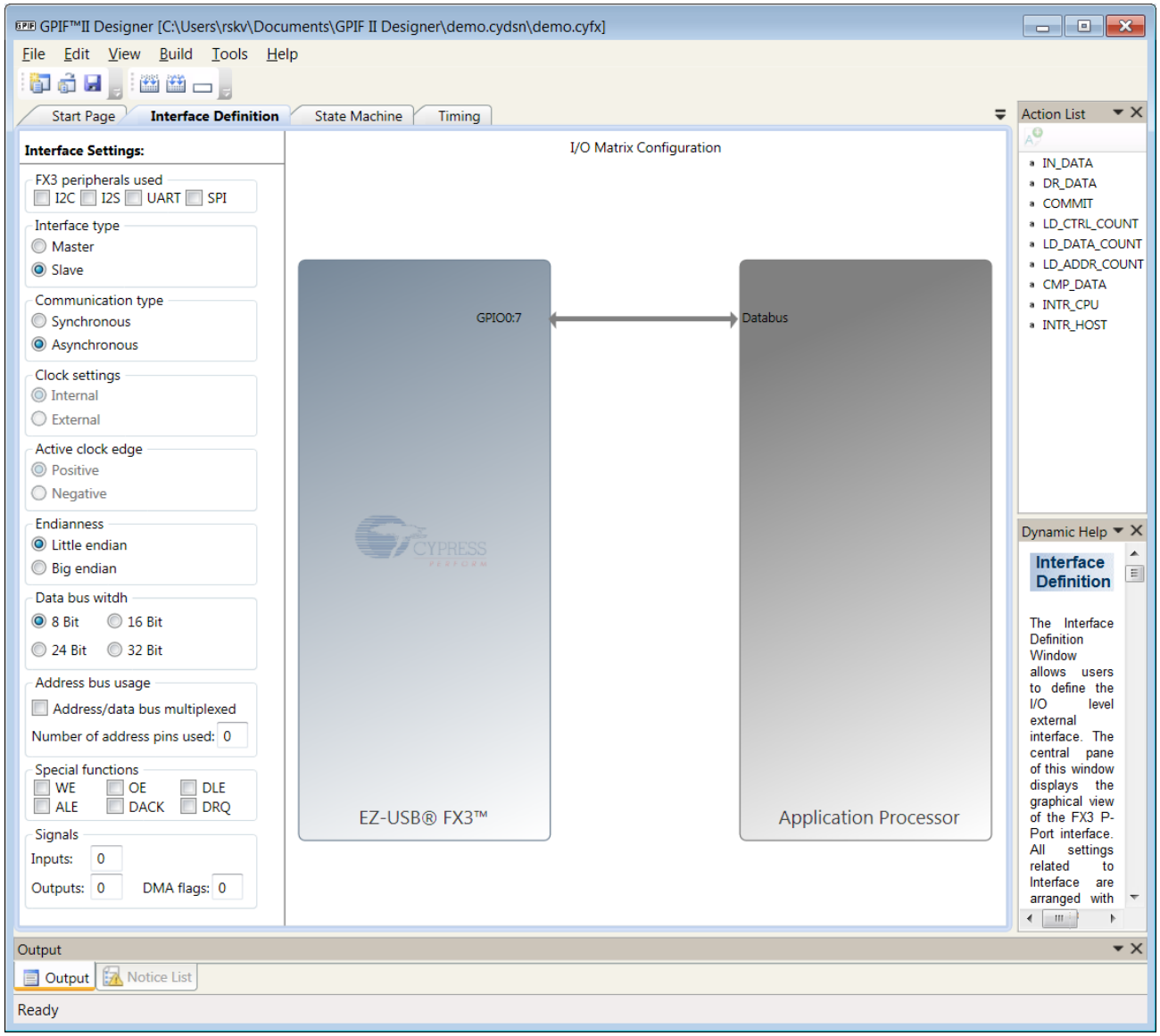
7.4.10 Developing the GPIF II State Machine

This section describes the steps involved in developing the GPIF II state machine.

7.5 Designing a GPIF II Interface

The first step is to configure the GPIF II outside-world interface by filling out the entries in the Interface Settings section of the Interface Definition tab (Figure 7-20). As you select and deselect options, the center panel changes to reflect a "living schematic" of the interface. This saves you the trouble of figuring out the FX3 pin mapping because the FX3 signals are labeled in the FX3 block.

Figure 7-20. GPIF II Interface Definition Tab



Before starting a GPIF II design, consider the following questions:

1. **Which FX3 serial peripherals will be used by your overall application?**

   In addition to the programmable GPIF II interface, the FX3 device implements a set of serial communication blocks to connect to external peripheral devices. The serial communication protocols supported are I2C (master only), I2S (transmitter only), SPI (master only), and UART. Enabling a specific peripheral block on the FX3 affects the number of pins available for use as part of the GPIF II interface.

2. **Will FX3 act as a master or a slave of the interface?**

   A GPIF II interface allows the FX3 to interface with an external processor acting as a master or a slave. If the interface transactions with the external system are initiated by FX3, then FX3 is the master. FX3 is a slave if the interface transactions are initiated by the external processor (connected on the GPIF II port), and the FX3 is only expected to respond to the actions initiated by the external processor. A given GPIF II configuration uses the FX3 as a master or a slave device. It is not possible to implement both modes in a single configuration.

   If an address bus is part of the electrical interface, this will serve as an input for slave mode designs and as an output for master mode designs.

When you select Master or Slave, the Action List (right panel) automatically updates to show the available choices for the selection.

3. **Does this interface use a clock?**

   A GPIF II-based interface can be synchronous or asynchronous in nature. In the case of a synchronous project, the GPIF II state machine operates using the same clock that is used on the interface. A given GPIF II configuration is asynchronous or synchronous; it is not possible to have both coexisting in a given configuration.

   An example of an asynchronous interface is a static RAM that has an address and data bus, read and write strobes, but no clock. A read operation, for example, occurs by asserting the address, and then asserting chip enable and read strobes. The data comes out a propagation time after the read strobe, with no reference to a clock.

4. **Do you want FX3 GPIF II to drive the interface clock?**

   If yes, then select Internal; otherwise, select External. This section is dimmed out if you do not select Synchronous in question 3.

   This interface clock can be generated and driven out by the FX3 device or provided as input to the FX3 device. The maximum frequency supported for the GPIF II interface clock is 100 MHz.

5. **When do you want GPIF II to sample interface signals? On the positive edge or the negative edge?**

   This section is dimmed out if you do not select Synchronous in question 3.

6. **Endian-ness of your device: Little endian or big endian?.**

   "Endian-ness" refers to byte ordering in multibyte integers-most significant (big) or least significant (little) byte first.

7. **Data bus width of the external device connected to FX3: 8-bit, 16-bit, 24-bit or 32-bit?**

   A GPIF II interface offers a maximum of 32 bidirectional data lines, which can be split into, or time multiplexed with, address lines. In multiplexed operation, the address bus has the same width as the data bus. One of the control pins can be used to implement the ADV or ALE signal required to control the functionality of the address/data multiplexed bus. If the address bus is not time multiplexed with the data bus, the number of address lines available will depend on the width of the data bus.

8. **Does the external device require address lines?**

   This would be true if the device needs to select between FX3 resources, such as threads, sockets (configured as Slave in GPIF II) or to select the particular memory region on the external device (configured as Master in GPIF II).

9. **Do you want to enable special functions?**

   Some of the general-purpose I/O signals of FX3 can be associated with special functions that are commonly used in standard protocols. These special functions are only available on specific pins, and control signals that use them cannot be moved to other pins. The enables are:

   OE: Output Enable. Provides direct control of output drivers. The OE signal directly controls whether the data bus is driven or tristated by the FX3, so that the latencies associated with doing this through the GPIF II state machine are removed.

   WE: Write Enable. Provides direct control to disable output drivers on the data bus. The data bus will not be driven by FX3 if the WE special function is selected and the WE input is asserted.

   DLE: Data Latch Enable. Enables the input stage on the FX3 data bus to latch data. This special function is normally combined with the WE function.

   ALE: Address Latch Enable. Enables the input address stage on the FX3 to latch address values.

   DRQ: DMA Request. Provides a DMA request output that is controlled through the GPIF II state machine and that responds to the DACK input signal.

   DACK: DMA Acknowledge. This is not a separate special function, but an input that is used to control the behavior of the DRQ signal.

Table 7-4. GPIO mapping for Special Function signals

Special Function	GPIO
DLE/WE	GPIO_18
OE	GPIO_19
ALE	GPIO_27
DACK	GPIO_20
DRQ	GPIO_21

10. How many signals need to be monitored by GPIF II, and how many need to be driven by GPIF II? Also, how many DMA flags do you need in your application?

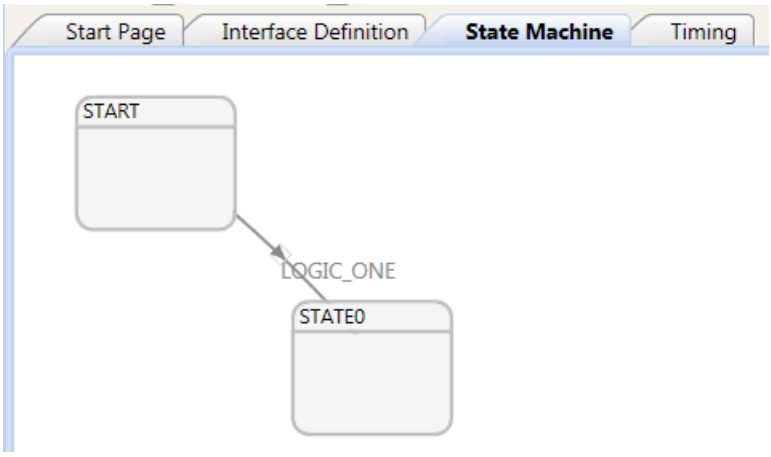
Control pins can be configured as input signals that drive the GPIF II state machine, output signals that are driven by the state machine, or flags that reflect the transfer readiness of various buffers on the FX3 device. You can configure any pins that are unused by the GPIF II block as GPIOs that are controlled by the firmware application. As you add inputs or outputs, the state machine designer automatically adds them as choices to be tested (inputs) or asserted (outputs) in any state.

7.6 GPIF II State Machine Implementation

The state machine canvas provides graphical controls to draw a GPIF II state machine diagram. A GPIF II state machine consists of actions performed in each state and triggers that cause transitions from one state to another. The Action List window displays the list of actions that can be added to states.

Click the State Machine tab to open the canvas. An unedited canvas has two states: START and STATE0. An unconditional transition (LOGIC_ONE) connects the states as shown in the figure below.

Figure 7-21. GPIF II Designer Interface

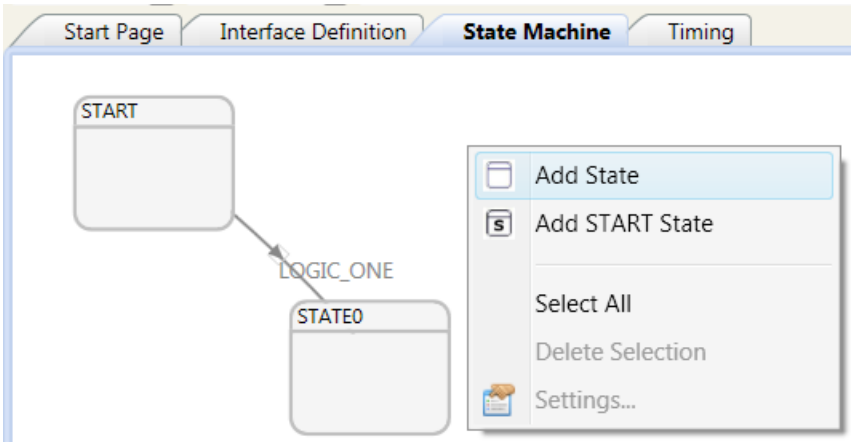


The state machine canvas supports the following operations.

7.6.1 Add a State

A right-click anywhere on the canvas displays the State Machine menu. Select Add State to add a state to the canvas as shown in the figure below.

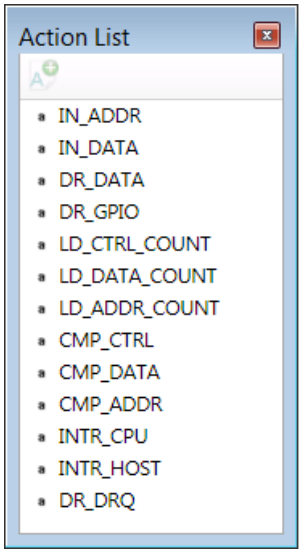
Figure 7-22. GPIF II Designer (Adding a State)



7.6.2 Add Actions to a State

To add an action to a state, select the state by clicking the mouse inside the state graphic. Right-click the action that you want to add from the Action List window. Select Add to Selected State from the menu. You can also open the Action List window by choosing View > Action List. The list of actions is shown in the figure below.

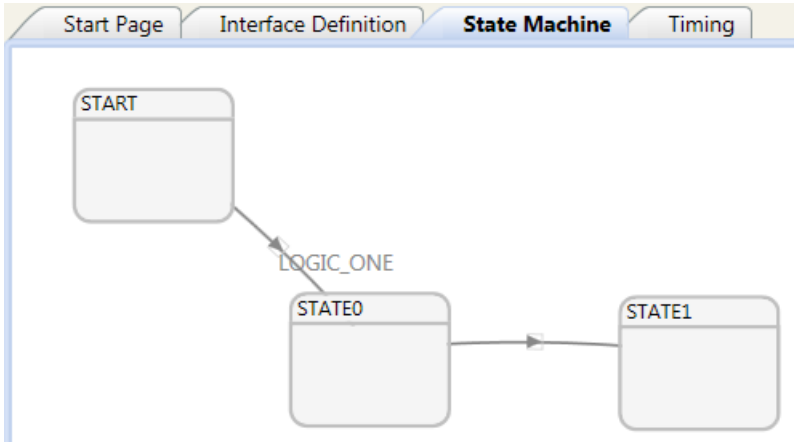
Figure 7-23. GPIF II Designer (Adding Actions to a State)



7.6.3 Draw Transitions Between Actions

Position the mouse cursor inside the source state, from which the transition will originate. The cursor changes its shape to “+.” Press the left mouse button, drag the mouse cursor to the destination state, and release it. The result of this step is shown in the figure below.

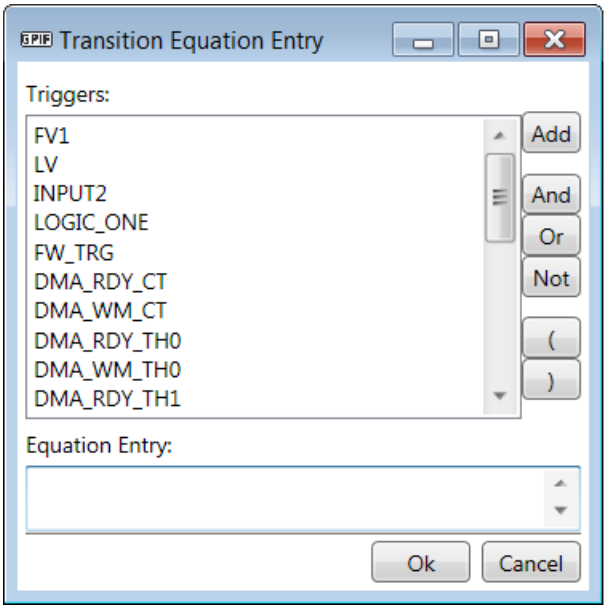
Figure 7-24. GPIF II Designer (Transitions Between Actions)



7.6.4 Add a Transition Equation

Bring the cursor to point to the transition line. Double-click on the line to open the Transition Equation Entry dialog box. All available triggers to form a transition equation are displayed as a selectable list. Select the trigger and add to the Equation Entry using the Add button. Use the necessary Boolean expression notations from the buttons on the left of the dialog box. Alternatively, you can type the equation directly in the Equation Entry box to enter the Boolean expression. Click OK after entering the equation. The window that pops up to enter the transition equation is shown in the figure below.

Figure 7-25. GPIF II Designer (Adding a Transition Equation)



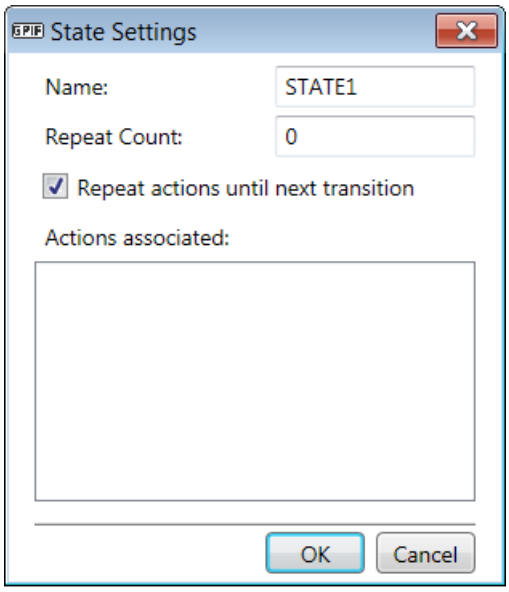
7.6.5 Set State Properties

Position the mouse cursor inside a state graphic. Right-click to display the State menu. Select Settings from the menu to open the State Settings dialog box. Every state can be associated with a name using an alphanumeric string. Macros that map the state names to the state IDs generated by the tool are generated by the tool as part of the header generation.

The Repeat Count property indicates the number of clock cycles to continue inside the state before evaluating any outgoing transition equation.

You can opt to have the actions associated with the state repeated until the transition to the next state by selecting the Repeat actions until next transition option as shown in the figure below.

Figure 7-26. GPIF II Designer (Set State Properties)



7.6.6 Analyzing the Signal Timing of the GPIF II Interface

After entering the state machine corresponding to the P-port interface, you can use the Timing window to simulate the relative timing of the input and output signals. The GPIF II project should be built without any errors before using the timing window for simulation.

To perform the timing simulation, select a specific state machine path in the state machine. Save the selected path for further viewing in a file. You can load a saved timing scenario from the menu provided on the top strip of the Timing window.

You can create a new timing scenario using the Create Scenario dialog box. A toolbar icon to create a scenario appears on the top strip of the Timing window. You can enter a unique name to identify the scenario being created in the Scenario Name. Use the Scenario Entry dialog box to select for the simulation a path consisting of a set of states that you can traverse sequentially.

The Timing window provides the following features:

7.6.6.1 Selection of Time Frame

The display frame of the timing diagram can be selected by specifying the start and end of the time frame. The start and end of time frame can be specified on the left and right bottom corner of the timing display respectively.

7.6.6.2 Automatic Timing Scale Selection

The timing window will be adjusted to the optimum scale according to the screen resolution and viewable area.

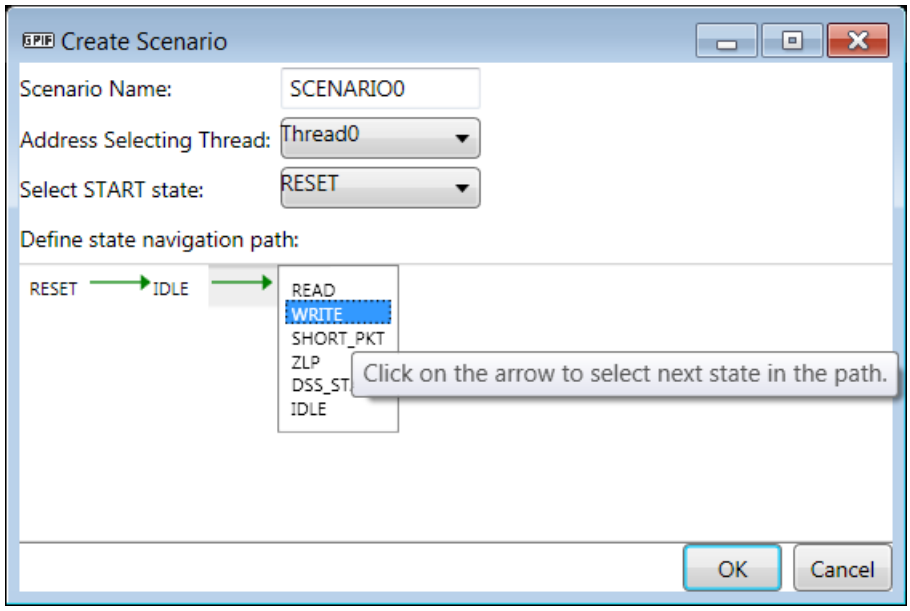
Note The timing simulation is performed based on a model of the GPIF II hardware. The GPIF II hardware execution is synchronous even if the interface is operating asynchronously. A typical operating (internal) clock frequency of 200 MHz (5-ns cycle time) is used in the simulation of asynchronous protocols.

7.6.7 Scenario Entry

You can simulate the state machine to view the relative timing and value of the signals in the form of a timing diagram. Select the state machine path whose behavior is to be simulated.

To select the path (state sequence) to be simulated, go to Timing Simulation > New Scenario or use the toolbar icon. The Create Scenario dialog box is displayed.

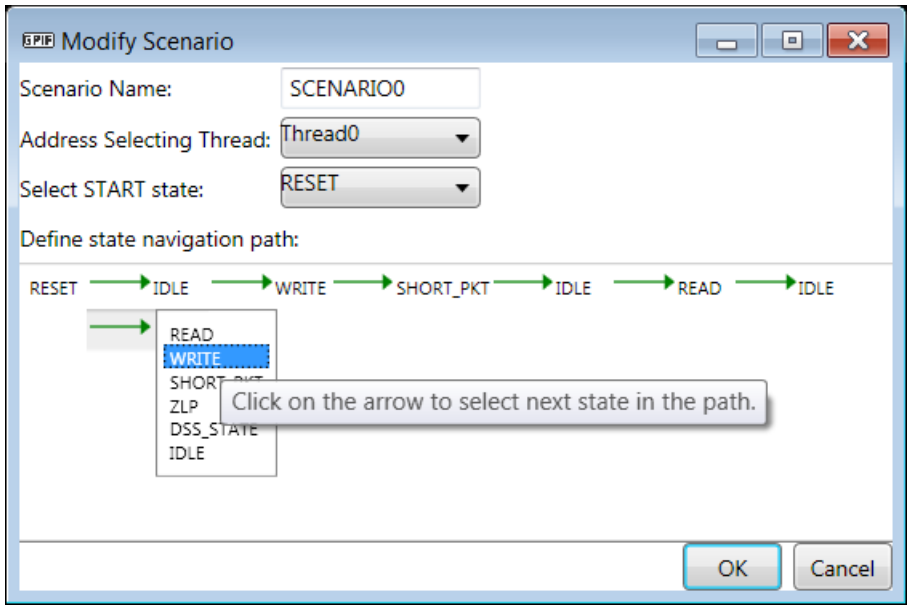
Figure 7-27. Create Scenario Dialog Box



Enter a name for the scenario being created. The name can also be used to view the timing scenario later. Select the start state of the sequence for states to be traversed. The tool then continuously provides a drop-down menu showing all the possible next states for the currently selected state, and you can define the path by repeatedly selecting the desired states. The current state is available as the next state option, unless a compulsory transition is specified in the state machine.

A timing scenario entered is saved with the name specified by the user during creation. All saved timing scenarios will be available in a drop-down menu on the top strip of the window. A saved scenario can be selected from this menu and modified or deleted. The Timing Simulation menu allows you to delete or modify the scenario.

Figure 7-28. Modify Scenario Dialog Box

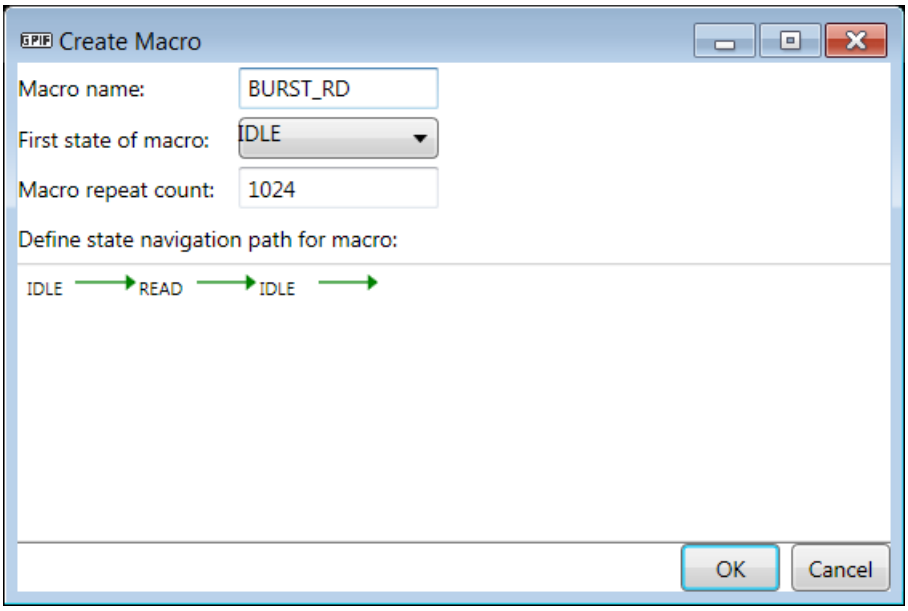


7.6.8 Macro

A macro is a reusable scenario corresponding to a circular timing path with repeat count. A macro is created and saved similar to a Scenario, except that the macro always has the same end state as the start state and has a repeat count.

Typically a macro is used to analyze repeating circular state paths. Consider a burst operation. The state paths of a burst read or burst write are repeated a number of times. An analysis of such an operation can be simplified by specifying it as a macro.

Figure 7-29. Create Macro Dialog Box



7.7 GPIF II Constraints

The GPIF II hardware imposes some limitations on the state machines that can it can create, as follows:

- Native support is limited to state machines that are limited to two (or fewer) outgoing transitions from each state. Such state machines are called "binary state machines" in the rest of this document.
- Each transition equation is limited to the use of four or fewer trigger variables.

Note that many GPIF II protocols involve simple decisions, such as sampling ready signals or other inputs. The following discussion applies to situations where the decision logic is less simple.

Two techniques are available to surmount these limitations:

- Mirror States
- Intermediate States

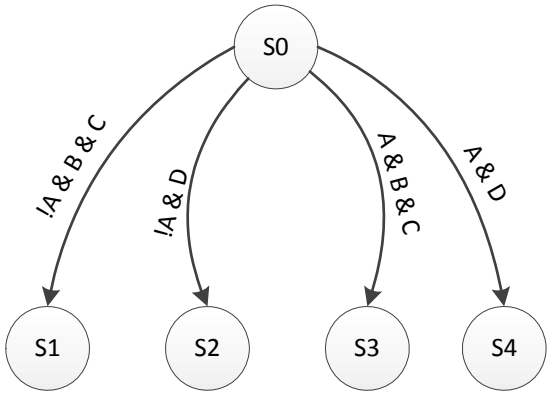
The GPIF II Designer tool applies the first technique automatically on state machines that exceed two exit triggers. However, it is possible that the tool may be unable to identify a set of transformations that allow the state machine to be mapped to the GPIF II hardware. In such a case, the tool outputs the error message "Unable to synthesize state machine." The user can follow Guidelines for Transition Equation Entry on page 154 to try and resolve these errors. If the error persists despite making these changes, contact Cypress Support for assistance.

7.7.1 Mirror States

The mirror state machine technique uses a GPIF II feature that facilitates state machine designs that do not conform to the previously described rules.

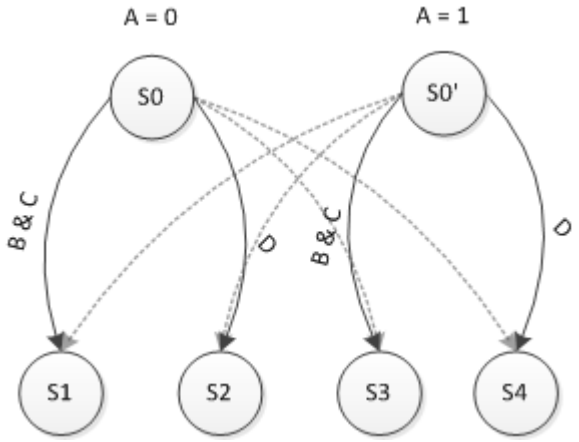
The GPIF II hardware supports working with multiple copies of a binary state machine, where the active state machine copy is determined using the current value of a trigger variable. For example, consider the state machine fragment shown in Figure 7-30. This state machine has a state, S0, which has four outgoing transitions based on the transition equations !ABC, !AD, ABC, and AD.

Figure 7-30. GPIF II State Machine Example Requiring Mirror States



The "A" can be removed from these transition equations to obtain the two modified state machines shown in Figure 7-31.

Figure 7-31. A GPIF II State Machine Split as Mirror States



In this case, S0' is inserted as a mirror state of S0. S0 has valid transitions pointing to S1 and S2, and S0' has valid transitions pointing to S3 and S4. The state machine {S0, S1, S2} is applicable when the trigger variable A has a value of 0, and the state machine {S0', S3, S4} is applicable when the trigger variable A has a value of 1. The GPIF II hardware automatically selects the correct state between S1 and S3 or between S2 and S4 by looking at the current value of A.

GPIF II Designer tries to apply this feature to implement state machines that have more than two out transitions from a state as well as state machines that use transition equations with excessive triggers. The derived state machines using this procedure need to follow a set of GPIF II mirroring rules (described in the next section) in order to be implemented without side effects.

A real example of this method is provided in 7.7.3 Mirror State Example on page 153.

7.7.2 Mirror State Rules

The GPIF hardware allows up to eight mirror state machines to be created, so that the active mirror is selected based on the current value of up to three input signals. The input signals that are used to select the active mirror state machine are called "global triggers."

A set of rules needs to be satisfied by all the mirror state machines. GPIF II Designer tries to identify a set of global triggers that reduce the state machine to one that satisfies the following rules.

- The transitions can be split into groups of two each, such that each group applies to a specific combination of trigger values.
- The transitions in each group have the same transition equations after the global trigger terms have been removed.
- The target states that share the same transition equation do not use conflicting actions. See Table 7-2 for details on incompatible actions.

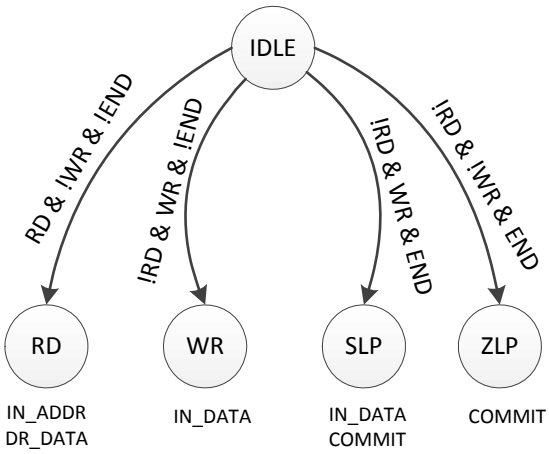
The tool first attempts to find a solution with one global trigger, then with two, and finally three global triggers. If the tool is unable to find a state machine reduction that satisfies the previous conditions with any of these levels, it will issue an error message saying that the state machine input cannot be synthesized.

In many cases, such state machine mapping errors are a consequence of an incompletely specified input. See 7.7.4 Guidelines for Transition Equation Entry on page 154 for state machine definition guidelines to aid the tool in finding a solution.

7.7.3 Mirror State Example

Figure 7-32 shows a part of the synchronous Slave FIFO protocol implementation that is included with GPIF II Designer as a Cypress supplied interface project.

Figure 7-32. Slave FIFO Interface Example State Machine Requiring Mirror States



The IDLE state is where the state machine waits for control signals to start performing a data transfer. Depending on the input signals, it may transition to one of four states:

The RD state, where a read operation is handled. Transition to the RD state is triggered by the RD input signal asserting while the WR and the END input signals are deasserted. The IN_ADDR and DR_DATA actions are performed in the RD state.

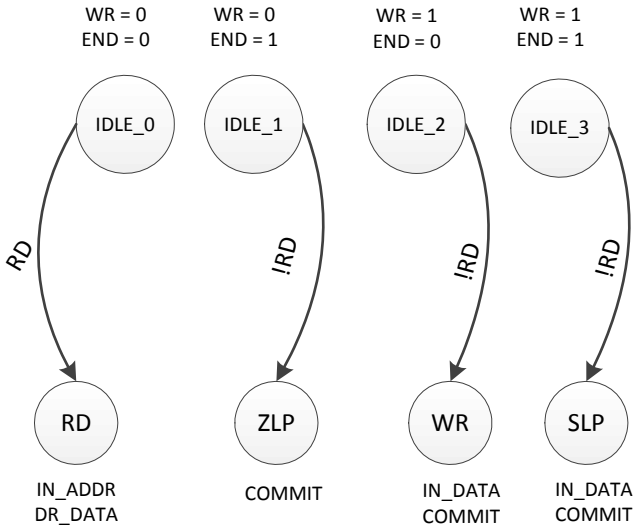
The WR state, where a general write operation (not end of packet) is handled. Transition to the WR state is triggered by the WR input asserting while the RD and END signals are deasserted. The IN_DATA action is associated performed with this state.

The SLP state, where a single word write operation is handled (data along with end-of-packet signaling). Transition to the SLP state is triggered by the WR and END inputs asserting while RD is deasserted. The IN_DATA and COMMIT actions are associated with this state.

The ZLP state, where a zero-length write operation is handled (no data, only end-of-packet signaling). This transition is triggered by END asserting while RD and WR are both deasserted. The COMMIT action is associated with this state.

The GPIF II Designer tool converts this state machine fragment into the implementation shown in Figure 7-33, which has four mirror state machines. The conditions under which each of the four mirrors is active are listed on top. All the transitions that are shown going to the right share the same transition equation, and the actions specified in these states are nonconflicting.

Figure 7-33. Slave FIFO Interface Example Implementation with Mirror States

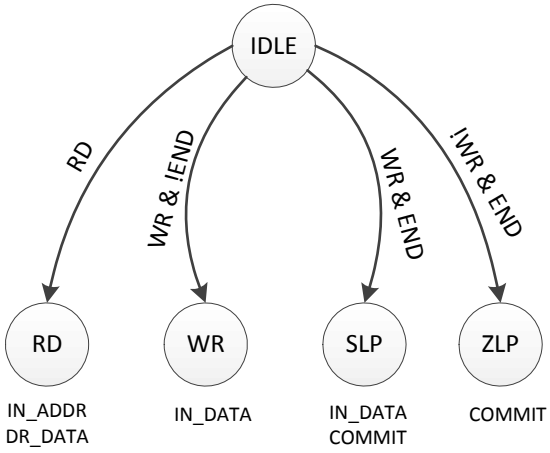


7.7.4 Guidelines for Transition Equation Entry

In some cases, GPIF II Designer is unable to reduce the input state machines to an implementable form because of insufficient input information.

For example, consider another form of the state machine fragment shown in Mirror State Example on page 153. This version of the state machine cannot be converted into multiple mirrored state machines that satisfy the mirroring rules using any global trigger combination.

Figure 7-34. State Machine Example with Mirror States



The problem is that the transition equations are not fully defined. For example, the tool needs to assume that the IDLE RD transition can happen independent of the values of the WR and END signals. If all the transition equations are made completely defined by adding the expected values for the other input signals, the state machine is transformed into the version shown in the example, and the tool can reduce the state machine to an implementable form.

In general, specifying each transition equation (particularly for transitions originating from a state that has more than two output transitions) fully by listing the expected value of all trigger inputs helps the tool find a mapping of the state machine to the GPIF II hardware. In the absence of such data, the tool treats the unspecified triggers as don't cares for a particular transition and may fail to find an implementable mapping.

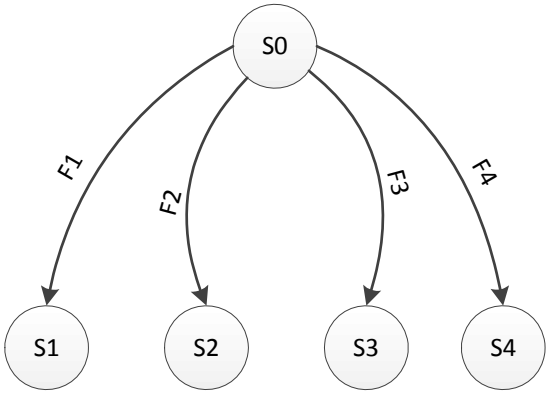
It is also recommended to avoid transition equations that involve multiple OR clauses in states that have more than two output transitions. This is because transition equations that have OR clauses typically cannot be refactored to remove global triggers.

7.7.5 Intermediate States

If the design allows adding one or more clocks to a transition, intermediate states can be inserted into the state machine to reduce the number of state transitions originating at a state.

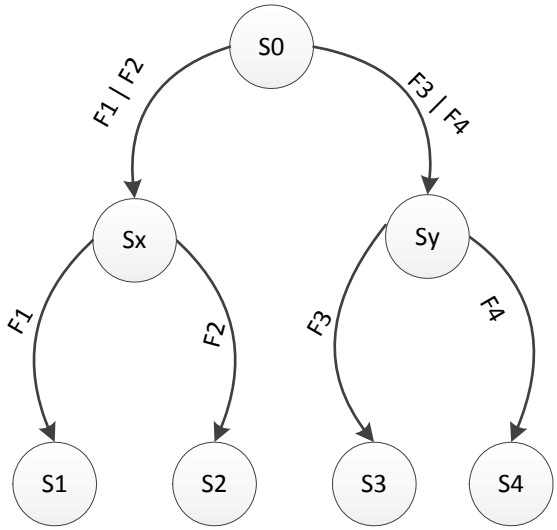
Consider the state machine fragment shown in Figure 7-35. There are four transitions originating at state S0, with the transition equations F1, F2, F3, and F4 respectively.

Figure 7-35. Example State Machine with Multiple Transitions



This state machine can be transformed into the version shown in Figure 7-36. Here Sx and Sy are dummy states that have been inserted to meet the constraint that each state can have only two outgoing transitions.

Figure 7-36. GPIF II Implementation for Multiple Transitions Avoiding Mirror States



This kind of transformation has an impact on the overall latency between states. For example, the actions specified in the state S1 will now be performed with an additional delay of one cycle. Similarly, the input values that trigger the S0 --> S1 transition now need to stay valid for one additional cycle. Since these system-level timing changes cannot be assumed safely, such transformations are not automatically performed by the GPIF II Designer tool. This technique can be used as necessary to reduce state machines to an implementable form.

7.8 Initialization and Configuration of GPIF II Block

The GPIF II block is configured by writing to a set of device registers and configuration memories. The GPIF II Designer utility generates the configuration data corresponding to the communication protocol to be implemented based on the graphical state machine. This tool generates the GPIF II configuration information in the form of a C header file that defines a set of data structures. The FX3 SDK and the GPIF II Designer installation also include a set of header files that have the configuration data for a number of commonly used protocols.

The CyU3PGpifLoad() API is used by the firmware application to load the configuration generated by the GPIF II Designer tool. This API, in turn, internally uses the CyU3PGpifWaveformLoad(), CyU3PGpifInitTransFunctions(), and CyU3PGpifConfigure() APIs to complete the configuration. These APIs can be used directly as a replacement for the CyU3PGpifLoad() call with the constraint that the CyU3PGpifConfigure() call should be made after the CyU3PGpifWaveformLoad() and CyU3PGpifInitTransFunctions() calls.

7.8.1 GPIF II State Machine Control

The firmware application may require the GPIF II state machine to be started and stopped multiple times during run time. In most cases where the FX3 device is functioning as a slave device, there will be a single GPIF II state machine, and the application needs to start it as soon as the configuration has been loaded. The CyU3PGpifSMStart() API can be used to start the state machine operation in this case.

If the application uses FX3 as a GPIF II master, there may be multiple disjointed state machines that implement different parts of the communication protocol. The CyU3PGpifSMSwitch() API can be used to switch between different state machines in this case.

If the GPIF II state machine operation is to be suspended and later resumed, the CyU3PGpifSMControl() API can be used to pause or resume state machine operation.

If the state machine operation is to be stopped at some point and restarted from the reset state later, the CyU3PGpifDisable() API can be used with the forceReload parameter set to CyFalse(0). The CyU3PGpifSMStart() API can then be used to restart the state machine.

If the active GPIF II configuration is to be changed, the CyU3PGpifDisable() API can be used with the forceReload parameter set to CyTrue(1). The CyU3PGpifLoad() API can then be used to load a fresh configuration.

7.9 Performing Read and Write Operations Using GPIF II

This section shows you the steps to program the GPIF II block with the help of a simple example that can be implemented using the FX3 SDK. There is no need to connect any external peripheral to the FX3 GPIF II. In this example, GPIF II performs a read operation when a DMA buffer is available in FX3, or it performs a write operation when data is available in the FX3 DMA buffers.

To design an interface using GPIF II Designer, start by creating a GPIF II Design project. A new project can be created using the New Project command from the File menu, or by using the links provided on the start page. The New Project command pops up the New Project dialog box shown in Figure 7-37. Here the user can enter the project name and choose the location where the project files will be stored.

An existing project can be opened by choosing the File menu item Open Project. The start page also provides links to open the most recently used projects.

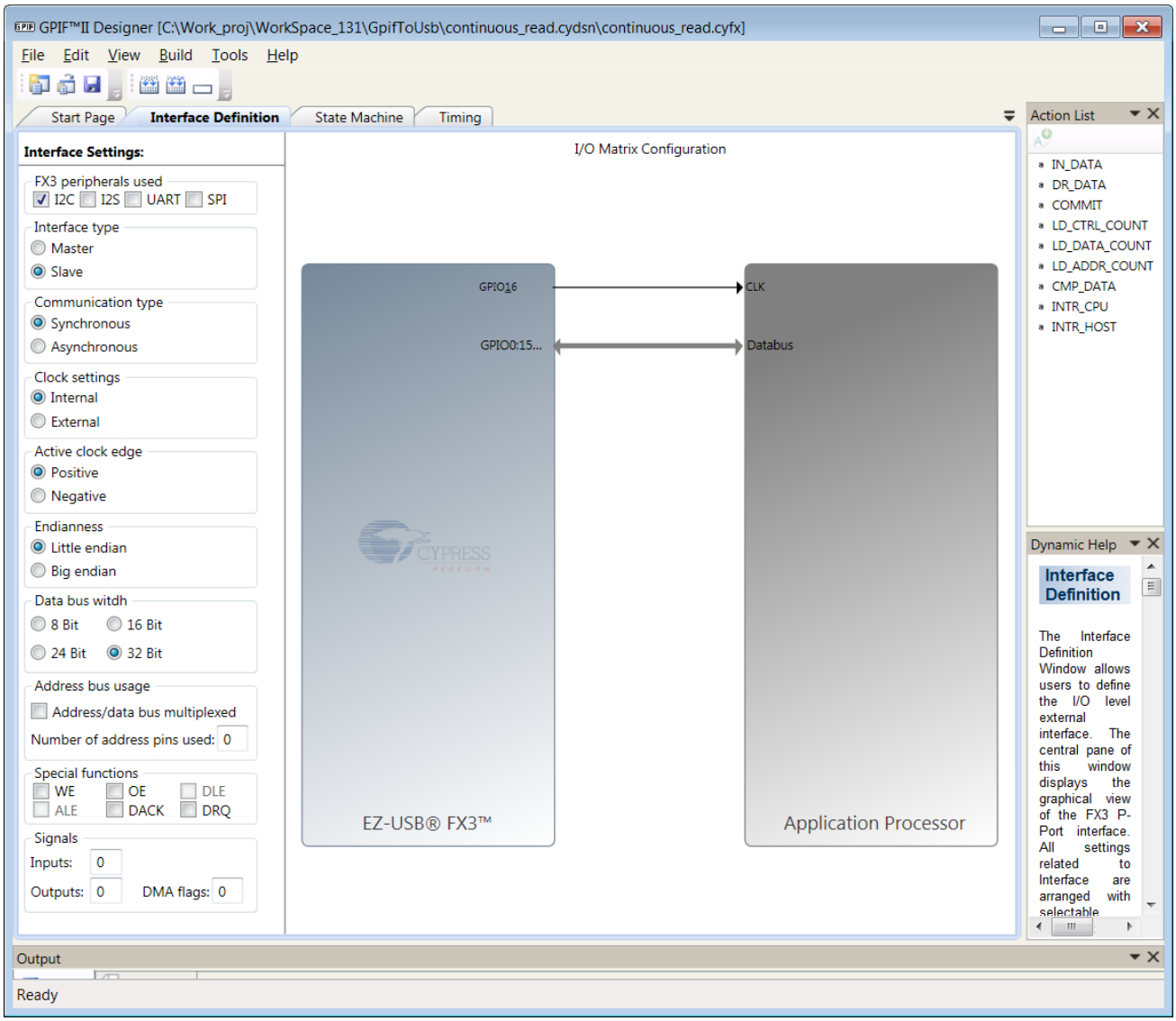
Once a project is opened, the Interface Definition window appears, where the user can enter the electrical interface details. See Designing a GPIF II Interface for details on the Interface Definition window.

Selections applicable for the interface are the following:

- Interface Type: Slave
- Communication type: Synchronous
- Clock settings: Internal
- Active clock edge: Positive
- Endianness: Little endian
- Data bus width: 8 Bit, 16 Bit, 24 Bit, or 32 Bit, as required
- Address/data bus multiplexed: Deselected (nonmultiplexed)
- Number of address pins used: 0
- No special function signals are needed for this example design. No input, output, and flags are used.

Figure 7-37 shows a screen shot of the Interface Definition window with the previous settings.

Figure 7-37. Interface Definition Window for Example Project



7.10 DMA Channel Creation in FX3 Firmware to Perform GPIF II to USB Data Transfers

The code to create a DMA channel in FX3 firmware to perform GPIF II to USB data transfers follows.

```
    dmaCfg.size = 16384;
    dmaCfg.count = 4;
    dmaCfg.prodSckId = CY_U3P_PIB_SOCKET_0;
    dmaCfg.consSckId = CY_U3P_UIB_SOCKET_CONS_1;
    dmaCfg.dmaMode = CY_U3P_DMA_MODE_BYTE;
    dmaCfg.notification = 0;
    dmaCfg.cb = 0;
    dmaCfg.prodHeader = 0;
    dmaCfg.prodFooter = 0;
    dmaCfg.consHeader = 0;
    dmaCfg.prodAvailCount = 0;

   apiRetStatus = CyU3PDmaChannelCreate (&glDmaChHandle, CY_U3P_DMA_TYPE_AUTO, &dmaCfg);
    if (apiRetStatus != CY_U3P_SUCCESS)     {
        CyU3PDebugPrint (4, "CyU3PDmaChannelCreate failed, Error code = %d\n", apiRetStatus);
         CyFxAppErrorHandler(apiRetStatus);
    }
```

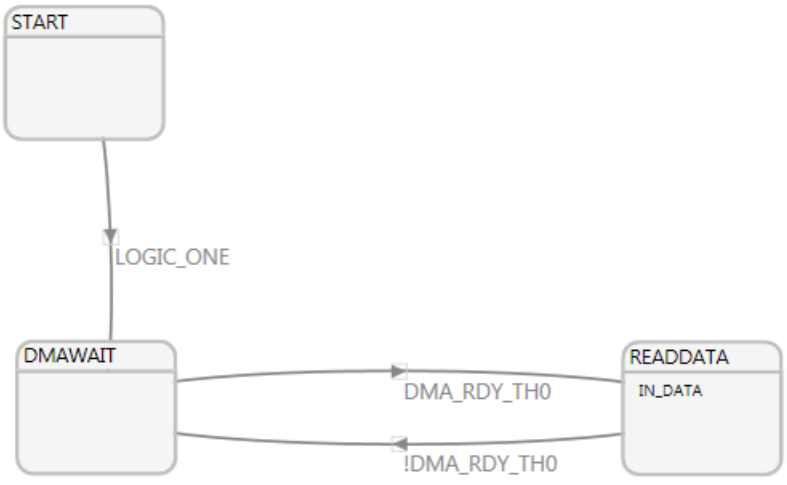
This code creates a DMA channel between socket 0 of the PIB and socket 1 of the UIB. In this example, GPIF II reads the data and sends it to the USB host. So you need to use the producer socket of the PIB and the consumer socket of the UIB.

This project allocates four buffers for this data path, and each buffer is 16384 bytes.

7.11 GPIF II State Machine to Read Data into a Socket

Figure 7-38 shows the GPIF II state machine to read data from the data bus. The GPIF II state machine stays in the DMAWAIT state until DMA_RDY_TH0 becomes HIGH. DMA_RDY_TH0 indicates the readiness of the DMA buffer of GPIF II thread 0. It asserts HIGH when there is at least one empty buffer available for producer socket 0. By default, socket 0 is mapped to thread 0.

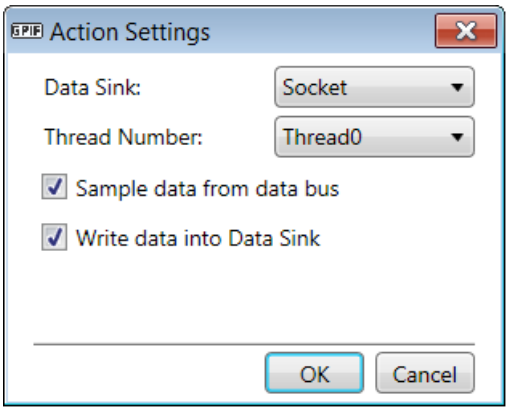
Figure 7-38. GPIF II State Machine to Read Data



When the DMA_RDY_TH0 flag asserts, the GPIF II state machine moves to the READDATA state. It performs an IN_DATA action in this state. The IN_DATA action reads the data available on the data bus and places it in the DMA buffer of socket 0. The IN_DATA action settings are shown in Figure 7-39. As long as the state machine is in the READDATA state, a new byte

is read from the data bus and sent to the DMA buffer on every clock.

Figure 7-39. IN_DATA Action Setting



The GPIF II state machine returns to the DMAWAIT state when the DMA_RDY_TH0 flag goes low. This happens when DMA buffer switching occurs for the created DMA channel or when all the allocated DMA buffers are filled with data and waiting to get cleared from the USB side.

When GPIF II tries to push data to the DMA buffer after it is full, then a PIB overrun error is flagged for the corresponding thread. The bit fields of the PIB error indicator register let you know if a PIB error is flagged.

7.12 DMA Channel Creation in FX3 Firmware to Perform USB to GPIF II Data Transfers

The code to create a DMA channel in FX3 firmware to perform USB to GPIF II data transfers follows.

```
    dmaCfg.size = 16384;
    dmaCfg.count = 4;
    dmaCfg.prodSckId = CY_U3P_UIB_SOCKET_PROD_1;
    dmaCfg.consSckId = CY_U3P_PIB_SOCKET_1;
    dmaCfg.dmaMode = CY_U3P_DMA_MODE_BYTE;
    dmaCfg.notification = 0;
    dmaCfg.cb = 0;
    dmaCfg.prodHeader = 0;
    dmaCfg.prodFooter = 0;
    dmaCfg.consHeader = 0;
    dmaCfg.prodAvailCount = 0;
   apiRetStatus = CyU3PDmaChannelCreate (&glDmaChHandle, CY_U3P_DMA_TYPE_AUTO, &dmaCfg);
    if (apiRetStatus != CY_U3P_SUCCESS)     {
        CyU3PDebugPrint (4, "CyU3PDmaChannelCreate failed, Error code = %d\n", apiRetStatus);
         CyFxAppErrorHandler(apiRetStatus);
    }
```

This code creates a DMA channel between socket 1 of the PIB and socket 1 of the UIB. In this example, GPIF II is driving data onto the data bus that is coming from the USB host. So you need to use the consumer socket of the PIB and the producer socket of the UIB.

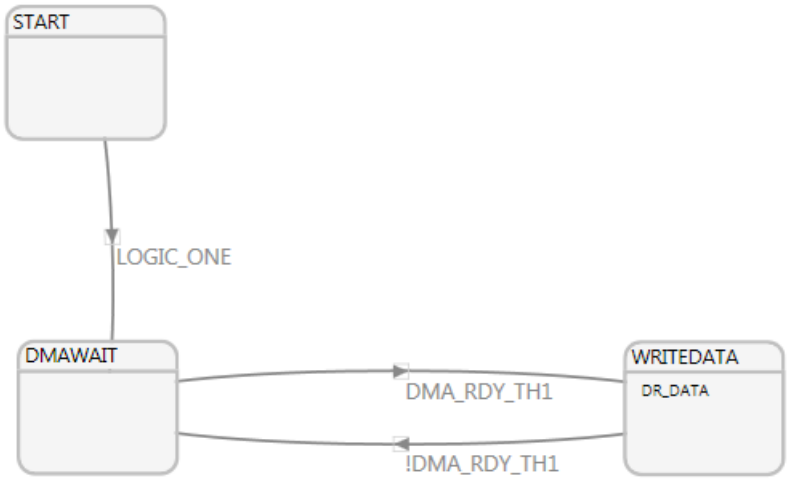
This project allocates four buffers for this data path, and each buffer is 16384 bytes.

7.13 GPIF II State Machine to Drive Data from Socket as Data Source

The GPIF II state machine to drive data onto the data bus is shown in Figure 7-40. The GPIF II state machine stays in the DMAWAIT state until DMA_RDY_TH1 becomes HIGH. DMA_RDY_TH1 indicates the readiness of the DMA buffer of GPIF II

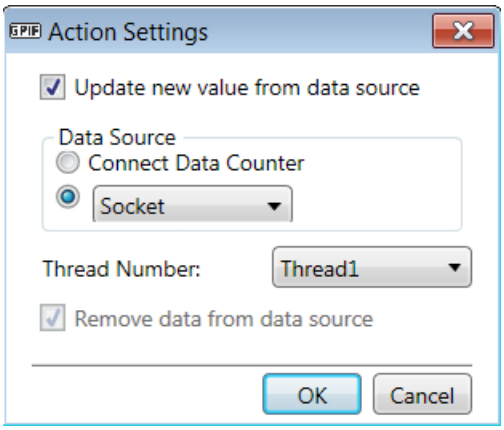
thread 1. It asserts HIGH when at least one DMA buffer of consumer socket 1 is available with data. By default, socket 1 is mapped to thread 1.

Figure 7-40. GPIF II State Machine to Drive Data on the Data Bus



The GPIF II state machine moves to the WRITEDATA state when the DMA_RDY_TH1 flag goes HIGH. It performs the DR_DATA action in this state. The DR_DATA action drives the data available in the DMA buffer of socket 1 onto the data bus, one word per clock. The DR_DATA action settings are shown in Figure 7-41.

Figure 7-41. DR_DATA Action Setting



The GPIF II state machine returns to the DMAWAIT state when the DMA_RDY_TH1 flag goes low. This happens when DMA buffer switching occurs for the created DMA channel or when all the allocated DMA buffers are empty and waiting to get data from the USB side.

When GPIF II tries to read data from the DMA buffer after it is emptied, then a PIB underrun error is flagged for the corresponding thread. The bit fields of the PIB error indicator register let you know if a PIB error is flagged.

7.13.1 Alpha Values

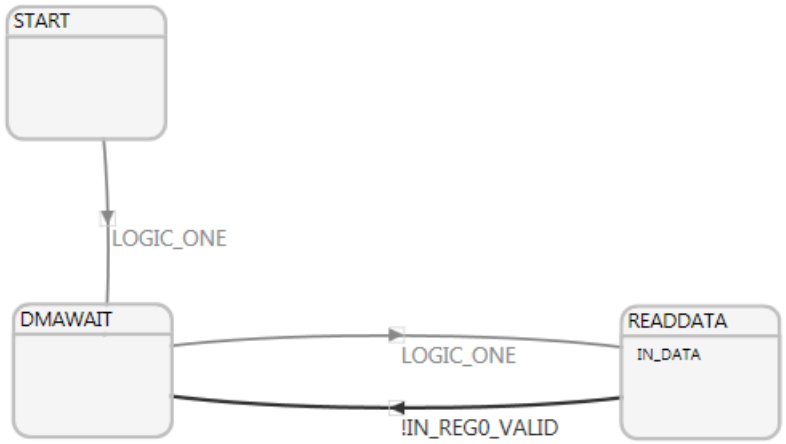
Control outputs from the GPIF II state machine can be classified into early outputs and delayed/normal outputs. The early outputs have a shorter output latency than the delayed outputs. This is achieved by making their values available to the GPIF II hardware one cycle earlier than all the other state information. The early output signals from GPIF II are called "Alphas," and their total number is limited to eight signals.

As the values for the Alpha class outputs are specified and interpreted differently by the GPIF II hardware, the initial values for these signals also need to be specified outside the state machine description. The initial Alpha values that a GPIF II design needs to have are generated in the form of a <PROJECT_NAME>_ALPHA_START macro in the GPIF II configuration header file. This value is expected to be passed as the initial Alpha parameter to the CyU3PGpifSMStart() API after the CyU3PGpifLoad() API has been called.

7.14 GPIF II Read and Write over Registers

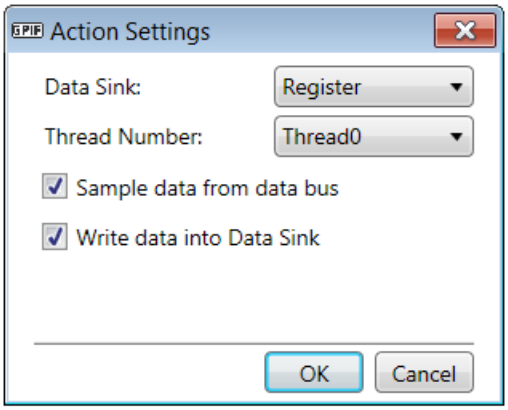
IN_DATAx_VALID and OUT_DATAx_VALID (where "x" varies from 0 to 3) are the bit fields of the GPIF_DATA_CTRL register. IN_DATAx_VALID is set high by the GPIF II hardware when data is written to the GPIF_INGRESS_DATA register corresponding to socket x using the IN_DATA action in the GPIF II state machine. This will be cleared by the FX3 firmware when the data in the GPIF_INGRESS_DATA register is read using the CyU3PGpifReadDataWords function. Figure 7-42 shows the GPIF II state machine that reads data from the data bus into the FX3 over the GPIF II register. It will read all ones if no device is connected to the FX3 GPIF II.

Figure 7-42. GPIF II State Machine to Read Data from the Data Bus



The IN_DATA action reads the data available on the data bus and places it in the GPIF INGRESS register corresponding to socket 0. IN_DATA action settings are shown in Figure 7-43.

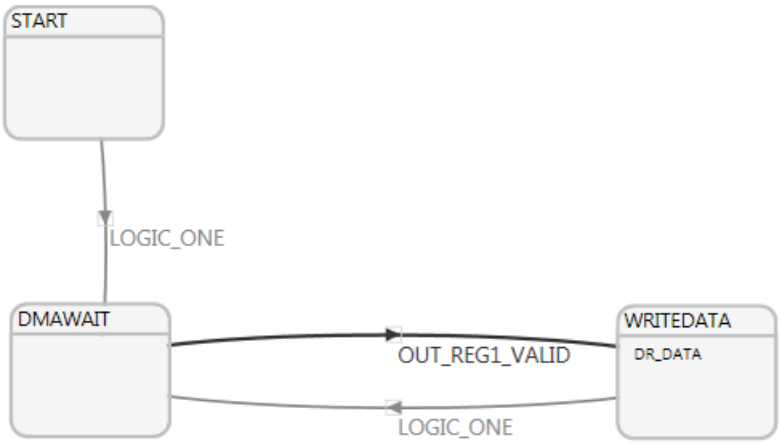
Figure 7-43. IN_DATA Action Settings



FX3 firmware writes '1' to the OUT_DATAx_VALID field to indicate a valid word is present in the GPIF_EGRESS_DATA register corresponding to socket x. The FX3 SDK provides the CyU3PGpifWriteDataWords function for doing so. The GPIF II hardware writes' 0' to indicate that the data is used and a new word can be written when the DR_DATA action executes in a

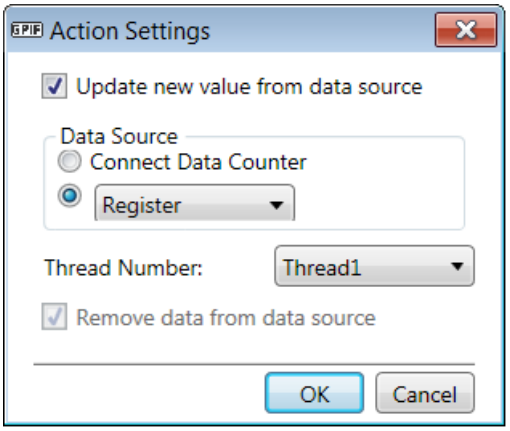
GPIF II state. The GPIF II state machine that drives data from the register is shown in Figure 7-44.

Figure 7-44. GPIF II State Machine Driving Data from the Register



The DR_DATA action drives the data available in the GPIF EGRESS register that corresponds to thread 1 onto the GPIF II data bus. The OUT_DATA action settings are shown in Figure 7-45.

Figure 7-45. OUT_DATA Action Settings



Selecting the option Remove data from data source will clear the OUT_REGx_VALID field of the GPIF_DATA_CTRL register.

7.15 Implementing Synchronous Slave FIFO Interface

This section explains the steps involved in designing a Slave FIFO interface using the FX3 GPIF II block.

The first step is to configure the GPIF II outside-world interface by filling out the entries in the Interface Settings tab. Selections applicable to the synchronous Slave FIFO interface are as follows.

- Interface type: Slave
- Communication type: Synchronous
- Clock settings: External
- Active clock edge: Positive
- Endianness: Little endian
- Data bus width: 8 Bit, 16 Bit, 24 Bit, or 32 Bit, as required

- Address/data bus multiplexed: Deselected (nonmultiplexed)
- Number of address pins used: 2 for selecting among the four threads available
- Special functions: OE: This feature is available only with GPIO_19. With this function, the assertion of GPIO_19 can directly change the data bus direction for the egress path (out of the FX3 device).
- There are five input signals: SLCS#, SLWR#, SLRD#, SLOE#, and PKTEND#. All are active low signals.

Table 7-5. Slave FIFO Interface signal description

Signal Name	Signal Description
SLCS#	The chip select signal for the Slave FIFO interface. It must be asserted to perform any access to the Slave FIFO interface.
SLWR#	The write strobe for the Slave FIFO interface. It must be asserted to perform write transfers to the Slave FIFO.
SLRD#	The read strobe for the Slave FIFO interface. It must be asserted to perform read transfers from the Slave FIFO.
SLOE#	The output enable signal. It causes the data bus of the Slave FIFO interface to be driven by FX3. It must be asserted to perform read transfers from the Slave FIFO.
PKTEND#	The packet end signal. It must be asserted to send a short packet to the USB host.

Four DMA flags (FLAGA, FLAGB, FLAGC, and FLAGD) are provided to the external peripheral to manage the data flow.

Figure 7-46 shows the FLAGA settings.

Figure 7-46. FLAGA Settings

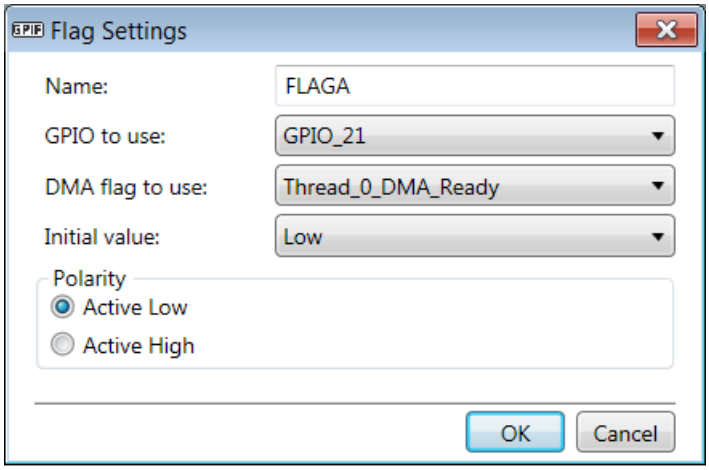
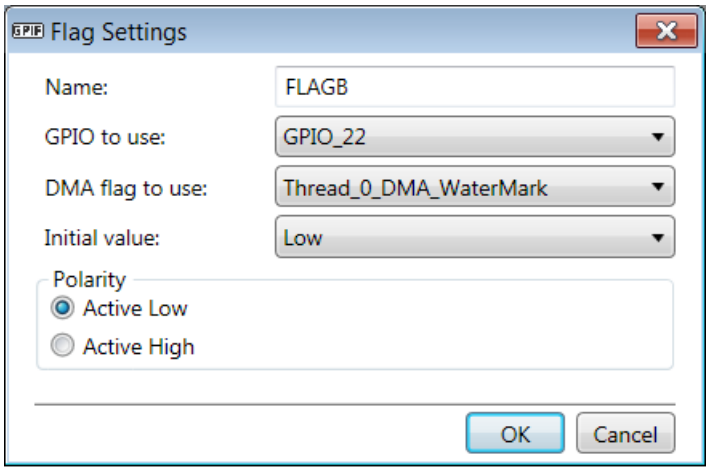


Figure 7-47 shows the FLAGB settings.

Figure 7-47. FLAGB Settings



The WaterMark value needs to be set using the CyU3PGpifSocketConfigure API. With this API, the active socket for each thread and its properties can be selected by the user at run time.

Figure 7-48 shows the FLAGC settings.

Figure 7-48. FLAGC Settings

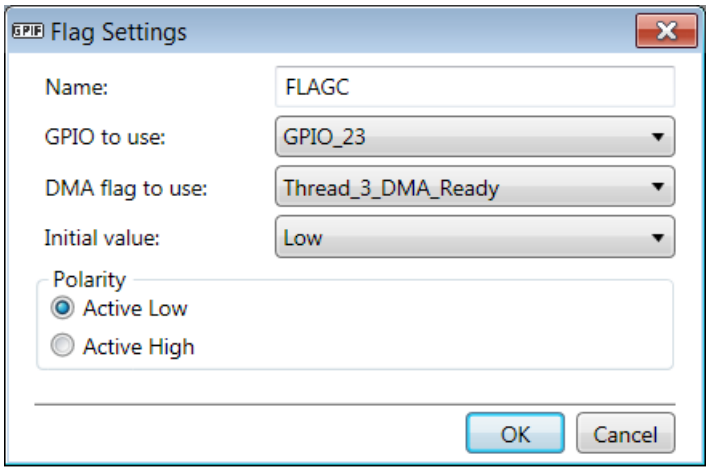
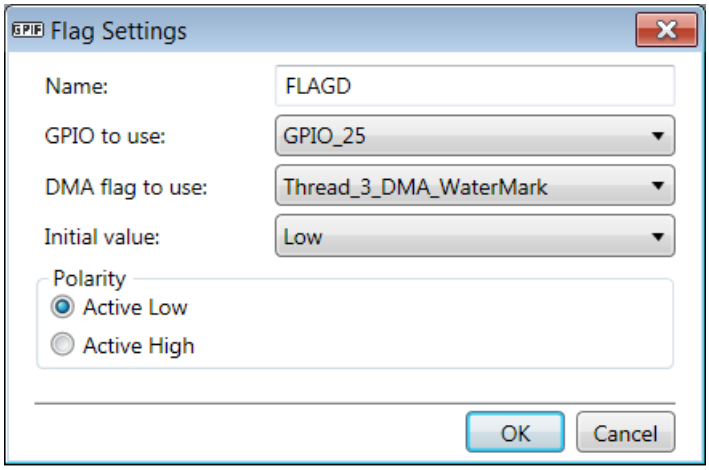


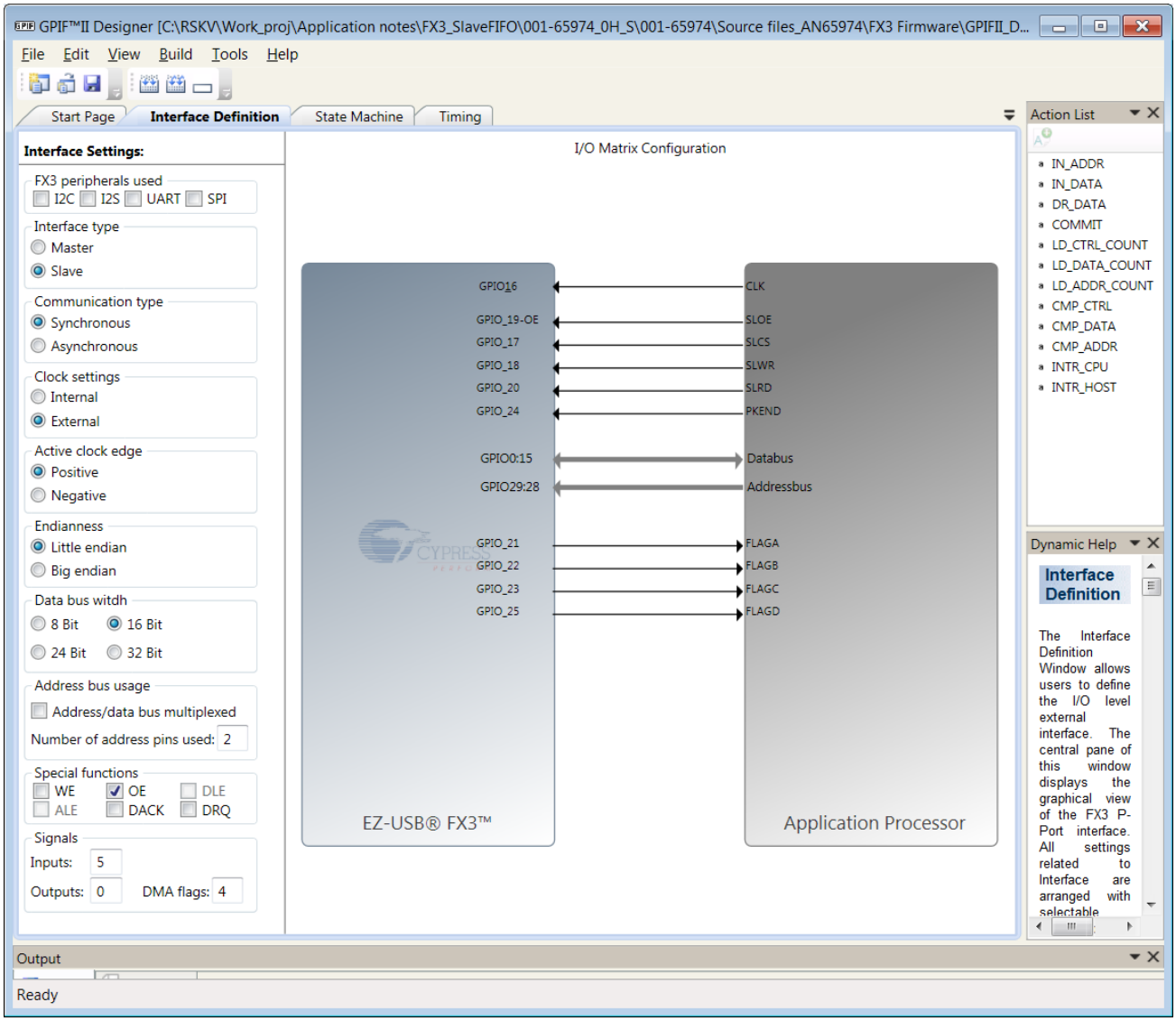
Figure 7-49 shows the FLAGD settings.

Figure 7-49. FLAGD Settings



A screen shot of the Interface Definition window with the previous settings is shown in Figure 7-50.

Figure 7-50. Interface Definition Window

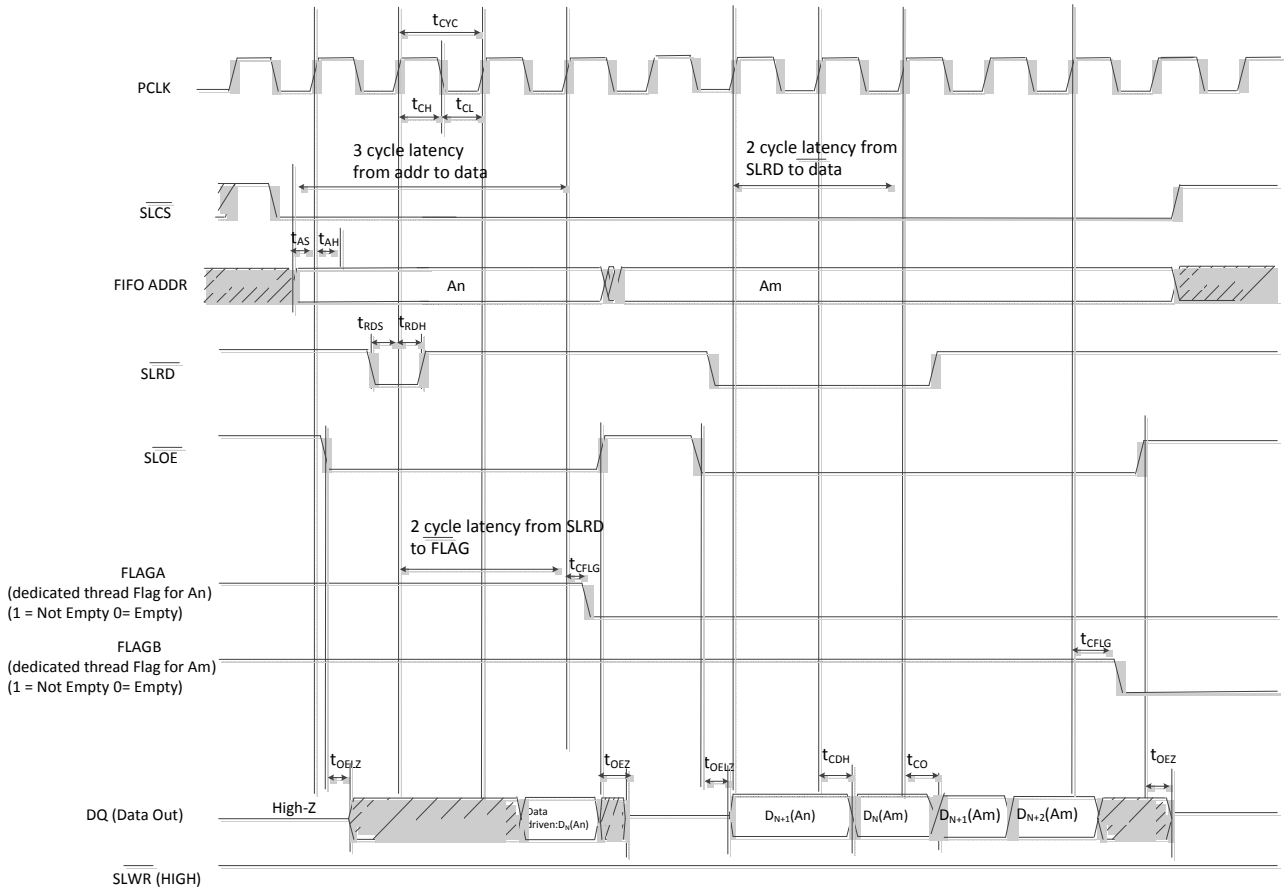


7.16 Synchronous Slave FIFO Access Sequence and Interface Timing

This section describes the access sequence and timing of the synchronous Slave FIFO interface.

An external processor or device (functioning as interface master) may perform single-cycle or burst data accesses to the FX3 internal FIFO buffers. The external master drives the 2-bit address on the ADDR lines and asserts the read or write strobes. FX3 asserts the FLAG signals to indicate the empty or full condition of the buffer.

Figure 7-51. Synchronous Slave FIFO Read Sequence



7.16.1 Synchronous Slave FIFO Read Sequence Description

The sequence for performing reads from the synchronous Slave FIFO interface is as follows.

1. The FIFO address is stable and SLCS# is asserted.
2. SLOE# is asserted. SLOE# is an output enable that signals FX3 to drive the data bus.
3. SLRD# is asserted.

The FX3 FIFO pointer is updated on the rising edge of the PCLK, while SLRD# is asserted. This action starts the propagation of data from the newly addressed FIFO to the data bus. After a propagation delay of tCO (measured from the rising edge of PCLK), the new data value is present. N is the first data value read from the FX3 FIFO. To drive the data bus, SLOE# must also be asserted.

The same sequence of events applies to a burst read.

Note: For burst mode, SLRD# and SLOE# are left asserted during the entire duration of the read. When SLOE# is asserted, the data bus is driven with data from the previously addressed FIFO. For each subsequent rising edge of PCLK, while SLRD# is asserted, the FIFO pointer is incremented, and the next data value is placed on the data bus.

FLAG Usage: FLAG signals are monitored by the external processor for flow control. FLAG signals are FX3 outputs that may be configured to show empty/full/partial status for a dedicated thread or the current thread being addressed.

Figure 7-52. Synchronous Slave FIFO Write Sequence

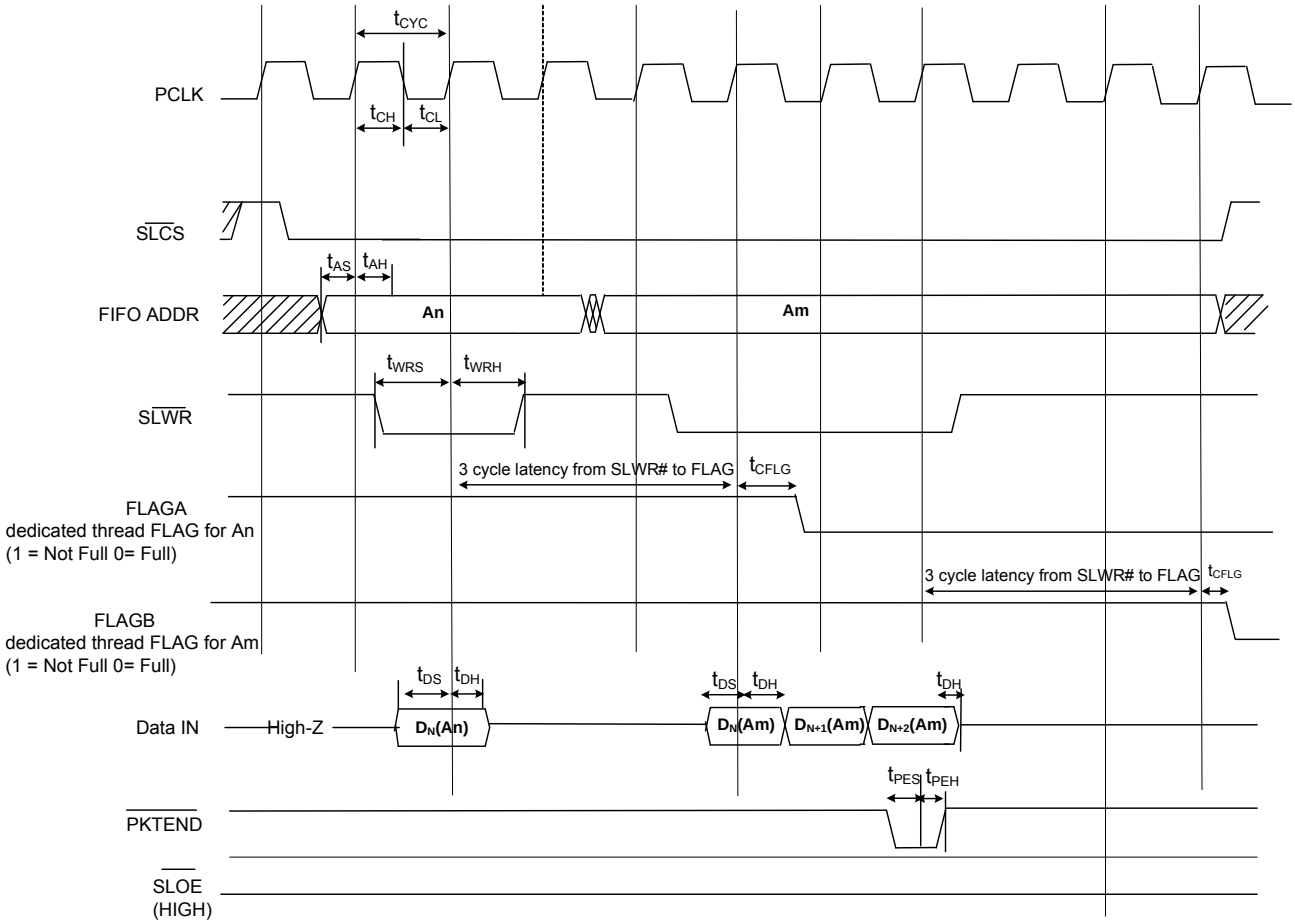
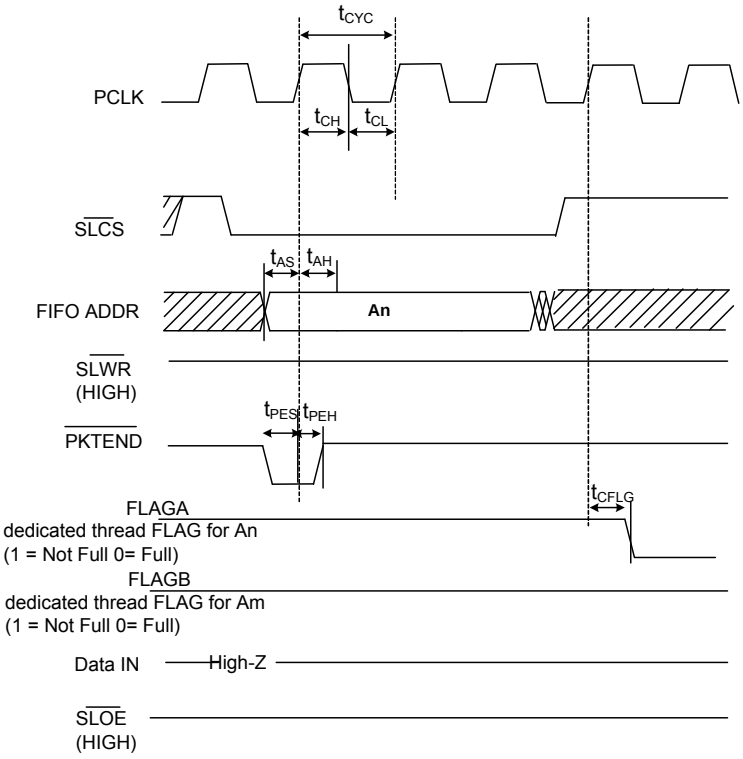


Figure 7-53. Synchronous ZLP Write Cycle Timing



7.16.2 Synchronous Slave FIFO Write Sequence Description

The sequence for performing writes to the synchronous Slave FIFO interface is as follows.

1. The FIFO address is stable, and the signal SLCS# is asserted.
2. The external master/peripheral outputs the data onto the data bus.
3. SLWR# is asserted.
4. While SLWR# is asserted, data is written to the FIFO, and on the rising edge of PCLK, the FIFO pointer is incremented.
5. The FIFO flag is updated after a delay of tCFLG from the rising edge of the clock.

The same sequence of events applies to a burst write.

Note: In the burst mode, SLWR# and SLCS# are left asserted for the entire duration of the burst write. In the burst write mode, after SLWR# is asserted, the value on the data bus is written into the FIFO on every rising edge of PCLK. The FIFO pointer is updated on each rising edge of PCLK.

Short Packet: A short packet can be committed to the USB host by using the PKTEND# signal. The external device/processor should be designed to assert the PKTEND# along with the last word of data and the SLWR# pulse corresponding to the last word. The FIFOADDR lines must be held constant during the PKTEND# assertion. On assertion of PKTEND# with SLWR#, the GPIF II state machine interprets the packet to be a short packet and commits it to the USB interface. If the protocol does not require any short packets to be transferred, the PKTEND# signal may be pulled high.

Note that in the read direction, there is no specific signal to indicate that a short packet has been sourced from USB. The empty FLAG must be monitored by the external master to determine when all the data has been read.

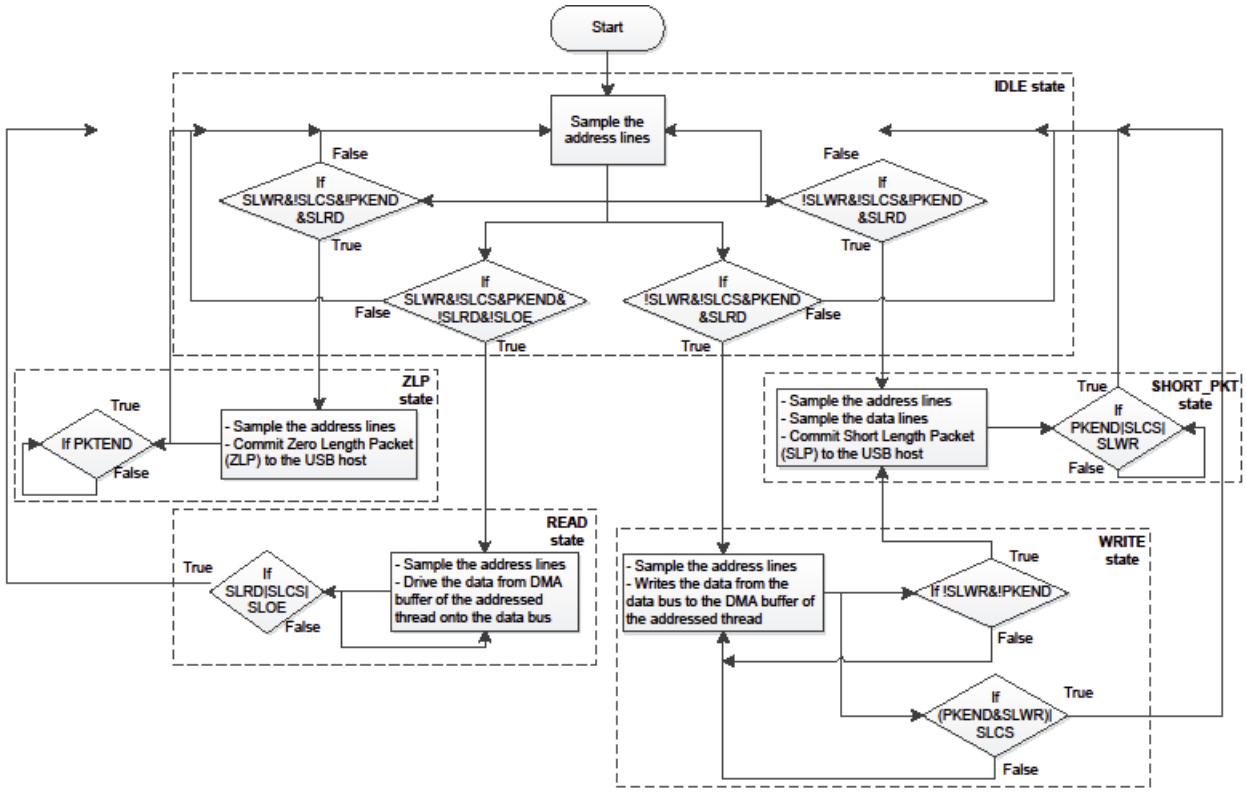
Zero-Length Packet: The external device/processor can signal a ZLP by asserting PKTEND#, without asserting SLWR#. SLCS# and the address must be driven as shown in Figure 7-53.

FLAG Usage: FLAG signals are monitored by the external processor for flow control. FLAG signals are FX3 outputs that may be configured to show empty/full/partial status for a dedicated thread or the current thread being addressed.

7.16.3 Slave FIFO Interface Logical Diagram

Figure 7-54 illustrates the logical flow chart of the Slave FIFO interface. For more information about the synchronous Slave FIFO interface, refer to the application note AN65974 - Designing with the EZ-USB FX3 Slave FIFO Interface.

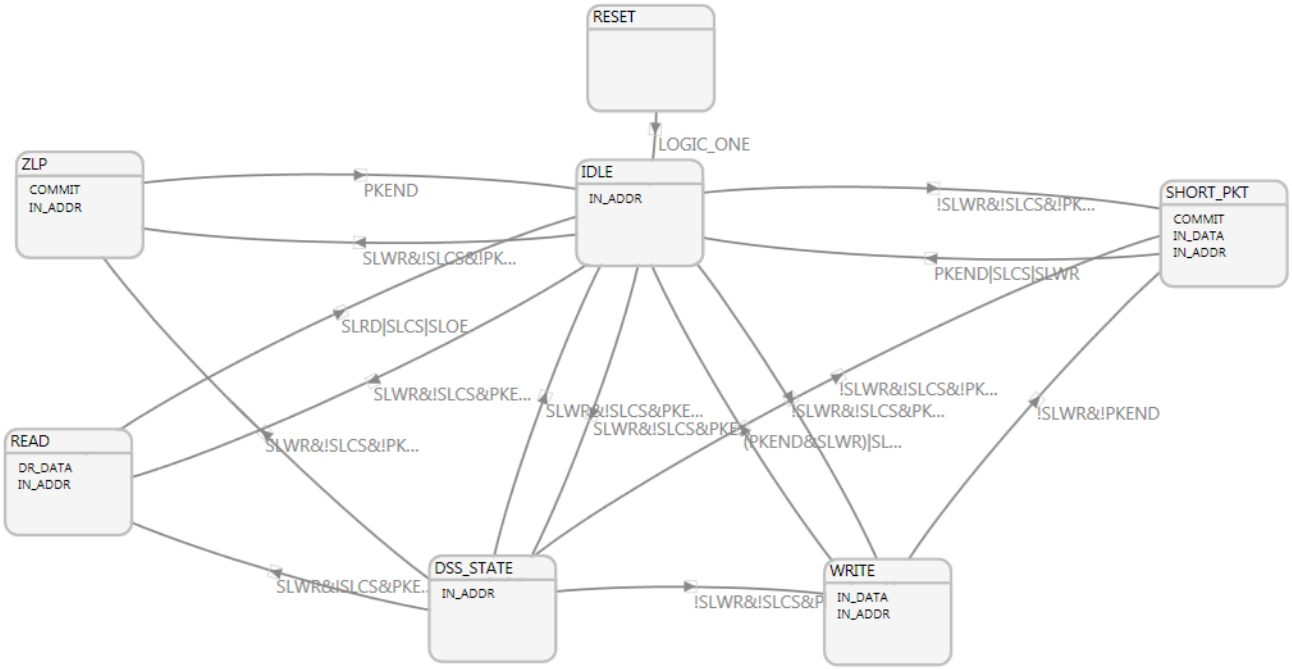
Figure 7-54. Slave FIFO Interface Flowchart



7.16.4 GPIF II State Machine of Slave FIFO Interface

Figure 7-55 shows the GPIF II state machine of the Slave FIFO interface.

Figure 7-55. GPIF II State Machine of the Slave FIFO Interface

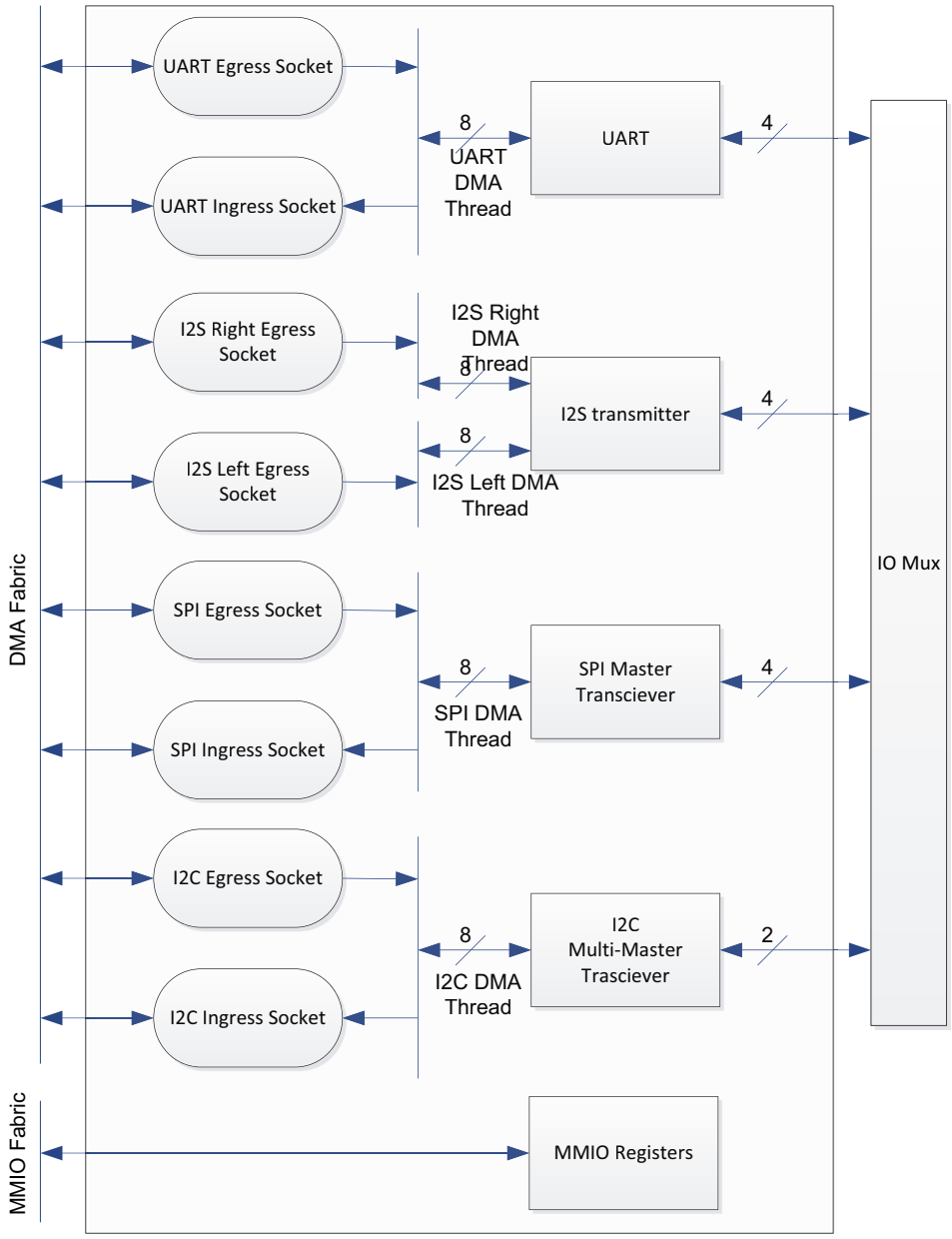


8. Low Performance Peripherals (LPP)

The FX3 serial peripherals (I2C, SPI, I2S and UART) including GPIO form the Low Performance Peripherals (LPP). LPP blocks are accessed using registers and the I2C, SPI, I2S, and UART peripheral blocks are DMA capable. The GPIOs cannot be grouped into a parallel bus and cannot use DMA.

The bandwidth requirement of all the low-performance peripherals is minimal compared to DMA bandwidth. All LPP peripherals interface to the DMA fabric as one entity, which connects to the DMA fabric through a single adapter and multiple threads, as shown in Figure 8-1. Eight DMA sockets are available for four LPP blocks. Any LPP block can use any number of available sockets from the eight DMA sockets, but only two sockets can be active or enabled at a time for a block. All eight DMA sockets can be active together.

Figure 8-1. LLP's Interfaces and DMA Fabric



Upon reset, all the LPP blocks are disabled. All the LPP blocks become operational only after an enable bit in a configuration register is set. Each block has its own block-level soft reset; LPP soft reset and the global reset are synchronized to the core clock. All the resets need to be deasserted for the block to be operational.

8.1 I2C Interface

8.1.1 I2C Block Features

The I2C block offers the following features:

- Operates in master mode

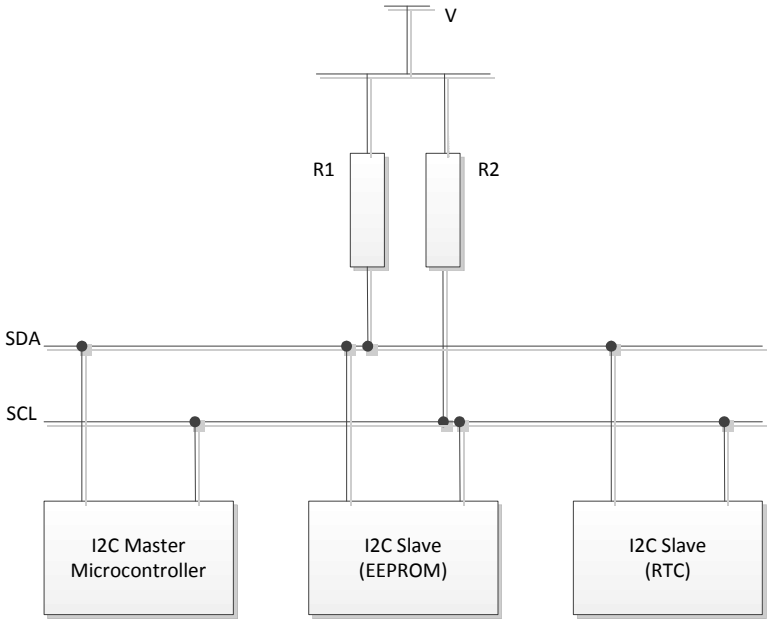
- Capable of multimaster negotiations
- Supports 100-kHz, 400-kHz, and 1-MHz clock
- Supports 7-bit and 10-bit addressing modes
- Supports clock stretching
- Supports register-based and DMA-based transfers
- FX3 can boot from I2C EEPROM

Refer to the application note AN76405 - EZ - USB FX3 Boot Options for more information.

8.1.2 I2C Interface Overview

The I²C protocol, created by Philips Semiconductor, allows data to be communicated between I2C devices over two wires. It sends information serially using one line for data (SDA) and one for clock (SCL), as shown in Figure 8-2.

Figure 8-2. I2C Bus



An I2C master device controls the bus, controlling the clock and generating START/STOP signals that are required for the communication. I2C slave devices simply listen to the bus and act based on the data sent on the I2C bus. The I2C master can send data to a slave or receive data from a slave. Slave devices cannot transfer data among themselves. In a single master system, the master device drives the clock, but slave devices can hold it low to synchronize slow slave devices (clock stretching). The two signals (SCL and SDA) must be pulled high using external resistors, as they are open collector/drain outputs and doing so implements a wired AND function. Any device pulling the signal low causes all the devices to see a low logic value, and for logic high, all devices must stop driving the signal. A slow slave device may need to stop the bus while it gathers data or services an interrupt. It can do so while holding the clock line (SCL) low, forcing the master into the wait state. The master must then wait until SCL is released before proceeding. Multimaster mode is used when there is more than one master on the I2C bus. This mode involves arbitration of the bus and clock synchronization.

FX3 is capable of functioning as a master transceiver and supports 100-kHz, 400-kHz, and 1-MHz operation. The I2C block operates in big endian mode (most significant bit first) and supports both 7-bit and 10-bit slave addressing. It supports single and burst (DMA) data transfers. Slow devices on its I2C bus can work with the FX3 I2C master by using clock stretching-based flow control.

FX3 can function in multimaster I2C bus environments, as it is capable of carrying out negotiations with other masters on the bus using SDA-based arbitration. Additionally, FX3 supports the repeated START feature to communicate with multiple slave

devices on the bus without losing ownership of the bus in between. Also, combined format communication is supported, which allows you to load multiple bytes of data (including slave chip address phases) into special registers called PREAMBLE. You can choose to place START, Repeated START, or STOP bits in between the data and also define the master's behavior on receiving either a NAK or ACK for the data in the preamble. In applications such as EEPROM reads, this greatly reduces the firmware complexity and execution time by packing the initial communication data into a transaction header with the ability to abort the header transaction on receiving the NAK indications in the middle of the transactions. In addition, the FX3 preamble repeat feature simplifies the firmware and saves time in situations that require time-consuming polling. For example, after programming (writing) an I2C EEPROM, the device must be polled to check for completion of the write operation. In this situation, the FX3 I2C block can be programmed automatically to repeat a single-byte preamble containing the EEPROM's I2C address until the device responds with an ACK.

By programming a burst read count value for this block, burst reads from the slave (EEPROM, for example) can be performed without byte-by-byte firmware intervention. In this case, the FX3 master receiver sends ACK response for all bytes received as long as the burst read counter does not expire. When the last byte of the burst is received, the FX3 I2C block automatically signals a NAK followed by a STOP bit, forcing the device to stop sending data.

8.2 FX3 I2C Operations Overview

The FX3 architecture supports register-based I2C operations for small transfers and offers DMA-based I2C operations for larger transfers. This section explains the I2C operations in detail:

8.2.1 Reset and Initialization

- On reset, all the blocks of the I2C core are placed in a disabled state. The core becomes operational only after the ENABLE bit in the configuration registers of this block (I2C_CONFIG) is set by the firmware.
- I2C clock speed is set in the GCTL block, as is the case for all FX3 blocks. GCTL_I2C_CORE_CLK is used to set the I2C clock speed. The I2C core clock must run at 10 times the bit rate on the external I2C interface (for example, for 100-kHz I2C, the I2C core clock must be set to 1 MHz). This requirement is necessary to generate the external clock with the proper setup and hold with respect to SDA. At 100 kHz, a 50 percent duty cycle is required, and at 400 kHz and 1 MHz, a 40/60 (high/low) duty cycle is required to meet the timing parameters of the I2C specification.
- The I2C clock must be enabled from GCTL before resetting the I2C block from the I2C_POWER register. Once the block is reset, the ACTIVE bit of the I2C_POWER register is monitored by the firmware before accessing any of the I2C block registers. Also, the block ENABLE bit of I2C_CONFIG can be set only after the block is in the active state.

8.2.2 Preamble

At the beginning of an operation (DMA or register mode), the master generates the start condition and transmits a 7-bit or 10-bit slave address, requiring 1 or 2 bytes. The last bit in these bytes indicates R/W#. The host can begin writing to and reading from the default address in the slave at this point. However, slave selection is typically followed by address specification in the slave's internal address space, which can comprise multiple bytes. After sending the address, the FX3 controller resends the start condition and slave address if the operation requires reading from the host, this time setting the R/W# bit high to indicate a read operation. This part of the transaction is called the “preamble” in FX3. The preamble is typically followed by data transfer, which can be one or several bytes..

8.2.3 Data Transfer

At the end of the preamble, the master transfers data that is derived from the connected DMA sockets or registers. The DMA_MODE bit of the I2C_CONFIG register determines whether the I2C core is configured for DMA mode or register mode transfers.

8.2.3.1 Programming Model

The FX3 I2C controller divides I2C transactions into two phases.

1. Control Phase: This phase consists of the addresses being transmitted before the actual read or write data starts. It is mainly handled by the I2C_PREAMBLE_CTRL and I2C_PREAMBLE_DATA registers. The I2C_PREAMBLE_DATA register is 8 bytes long. Start and stop conditions after each byte (0: before first byte) are specified in the 16-bit I2C_PREAMBLE_CTRL register.
2. Data Phase: This phase is handled by the I2C_COMMAND and I2C_BYTE_COUNT registers. The data phase can be connected to either DMA- or register-based data paths. DMA programming is the same as for other blocks and is not explained here.

8.2.3.2 Register-Based I2C Transfers

The FX3 firmware uses the I2C_PREAMBLE_CTRL, I2C_PREAMBLE_DATA, and I2C_COMMAND registers to initiate an operation. It then performs writes or reads from the TX and RX registers. ARM registers that transmit data from the ARM core to an external interface (in this case, I2C) are called EGRESS_DATA registers. For the receive function, that is, when READ is set in the I2C_COMMAND register, the data is received from the external interface (in this case, I2C) and pushed into the registers, called INGRESS_DATA registers. The firmware can be notified of completion by interrupt or by polling through the status register. A register I2C_BYTE_COUNT is used to specify the number of bytes to be written out and the same register is also used to determine the number of bytes received.

4-byte-deep FIFOs hold egress and ingress data in its register space. Based on the status of these FIFOs, the TX_SPACE, TX_HALF, TX_DONE and RX_DATA, RX_HALF flags are set in the I2C_STATUS and I2C_INTR registers.

The firmware can clear the FIFOs by asserting TX_CLEAR and RX_CLEAR from the I2C_CONFIG register. Table 8-1 shows the conditions for the assertion of flags:

Table 8-1. Conditions for Assertion of Flags

Flag Asserted	Egress FIFO State	Ingress FIFO State
TX_SPACE	Not full	-
TX_HALF	At least half empty	-
TX_DONE	Full	-
RX_DATA	-	Not empty
RX_HALF	-	At least half full

8.2.3.3 DMA-Based I2C Transfers

DMA-based transfers use the I2C sockets. The DMA interface supports sockets that are used for moving the data between the peripheral and the USB 3.0 interconnect. DMA sockets that transmit I2C data are called egress sockets, and DMA sockets that receive I2C data are called ingress sockets. I2C data is pushed into the DMA socket, when I2C interface is configured for READ in I2C_COMMAND register. The FX3 firmware uses the I2C_PREAMBLE_CTRL, I2C_PREAMBLE_DATA, and I2C_COMMAND registers to initiate an operation. The I2C transceiver reads the preamble and command and then initiates the operation after consulting the availability of ingress or egress sockets. Ingress and egress sockets can be programmed to interrupt the CPU upon transaction completion. The I2C controller always raises interrupts in error conditions.

Once the expected number of data bytes has been received (indicated by BYTE_COUNT), then an end of transfer is indicated to the DMA adapter by setting the flag RX_DONE (of I2C_STATUS and I2C_INTR) in both the register- and DMA-based transfers.

At any point, the status of the I2C block can be read from I2C_COMMAND.I2C_STAT bits to indicate the status of the block and I2C lines.

8.2.3.4 Starting a Transaction

1. Select the DMA mode and enable the block through the I2C_CONFIG register. This step needs to be done only once after booting, unless you want to change the data-path used in the data phase.
2. Program the I2C Byte count register to indicate the number of bytes to be transferred in the data phase. If you want the data phase to continue without any limit on the number of bytes, then program 0xFFFFFFFF. In this case, the data phase

can be terminated only by the FX3 firmware. The operation of the I2C_BYTE_COUNT register is identical in DMA and register mode data paths. The SCK_SIZE register of the chosen socket can be set to zero to indicate an indefinite transfer size in DMA.

3. Program the preamble and I2C_COMMAD registers. This step is explained in the next section through examples.
4. The transaction starts when I2C controller hardware detects the I2C_COMMAND.PREAMBLE_VALID bit.

8.2.3.5 Terminating Transactions: Software and Hardware Aborts

1. Transactions with a finite I2C_BYTE_COUNT are usually terminated by the hardware after the required number of bytes is exchanged. The end of transfer is indicated through the TX_DONE and RX_DONE interrupts.
2. The firmware can disable the block using the I2C_CONFIG register to abort a transfer in progress at the end of the current byte. In this case, the firmware must reset the I2C block before initiating a new transfer, and the new transfer should start with a START condition. It is advisable to read the I2C_INGRESS_DATA register during read transactions before aborting a transfer to avoid loss of data. The hardware may not generate a STOP in such cases depending upon when the abort happened. The firmware must ensure that the bus is freed by issuing a transaction with a STOP condition, such as a slave address or a general call address with a STOP condition, so that the bus is not deemed busy by other masters.
3. The hardware can terminate a transfer upon detecting a NACK from a slave or upon long periods of inactivity (SCK held low by the slave) as defined in the I2C_TIMEOUT register. In this case, I2C_BYTES_TRANSFERRED in the data phase indicates when the abort happened. For example, if a data NACK error happens at the end of the 5th byte, the value in this register will be 5. If a timeout occurs while transmitting the 1st bit of the 11th byte, then this value will be 11 and so on. If the hardware abort conditions occur during the preamble, the error codes indicate where in the preamble abort happened. NACK hardware aborts are indicated by an ERROR interrupt and the associated error codes.

8.2.3.6 Multimaster Arbitration

When multiple I2C master devices are present on the I2C bus, including FX3, then if FX3 sends a START condition and the competing master sends a STOP condition, the SDA line will be low. In this situation, FX3 will consider itself to have won arbitration since there is no way for it to detect a STOP sent by the competing master. On the other hand, if FX3 sent a STOP and the other master sends a START, then FX3 will detect the START condition and FX3 loses control of the I2C bus, since the SDA status does not match the expected value. Whenever FX3 loses control of the bus, a Lost Arbitration interrupt will be raised. Upon receiving this interrupt, the FX3 firmware must reset the block, which will clear any data in the core pipeline.

8.2.3.7 Error Conditions

Error conditions are indicated in the I2C_STATUS register. All errors marked as nonsticky do not require firmware intervention, while the errors marked sticky require the firmware to reset the I2C DMA sockets and reissue the command.

Note: If an ERROR bit is set in I2C_Status, it is recommended to disable and re-enable the I2C block.

8.2.4 Examples

This section shows you the example codes to write and read to an I2C slave device (EEPROM) using register-based and DMA-based transfers from FX3. APIs to perform read and write accesses to an I2C device are provided with the FX3 SDK. Refer to the Cyu3i2c.c file, which is located at C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\firmware\lpp_source (after FX3 SDK installation) for the source code of I2C-related APIs. Refer to FX3APIGuide.pdf located at C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\doc for more details on FX3 APIs.

8.2.4.1 Initialize I2C Block

The CyU3PI2cInit API initializes the FX3 I2C block. The I2C block is initialized to operate at the default frequency of 100 kHz. The CyU3PI2cInit function definition follows.

```
CyU3PI2cInit (void)
{
    /* Set the clock frequency. This should precede the I2C power up */
    CyU3PI2cSetClock (100000);
```

```
    /* Initialize the LPP block if not already started. */
    CyU3PLppInit (CY_U3P_LPP_I2C, CyU3PI2cInt_ThreadHandler);
    CyU3PMutexCreate (&glI2cLock, CYU3P_NO_INHERIT);

    /* Power on the I2C module */
    I2C->lpp_i2c_power = 0;
    CyU3PBusyWait (10);
    I2C->lpp_i2c_power |= 1 << 31;
    while (!(I2C->lpp_i2c_power & CY_U3P_LPP_I2C_ACTIVE));

    /* Clear the error status */
    glI2cStatus = 0;

    /* Mark the module as active. */
    glIsI2cActive = CyTrue;
}
```

8.2.4.2 Configure I2C Block

The CyU3PI2cSetConfig API configures the FX3 I2C block. Following is the code for setting the I2C bus frequency to 400 kHz. DMA transfers are not enabled in the following code. i2cConfig.isDma would need to be set CyTrue(1) to enable DMA transfers.

```
/* Start the I2C master block. The bit rate is set at 100 kHz. */
    CyU3PMemSet ((uint8_t *)&i2cConfig, 0, sizeof(i2cConfig));
    i2cConfig.bitRate    = 400000;
    i2cConfig.busTimeout = 0xFFFFFFFF;
    i2cConfig.dmaTimeout = 0xFFFF;
    i2cConfig.isDma      = CyFalse;

    CyU3PI2cSetConfig (&i2cConfig, NULL);
```

8.2.4.3 Reads and Writes Using Register Transfers

The CyU3PI2cReceiveBytes API reads data from the I2C slave in register mode, and the CyU3PI2cTransmitBytes API writes data to an I2C slave in register mode. The following code performs reads and writes to an I2C slave using register transfers.

```
CyFxUsbI2cTransfer (uint16_t  byteAddress, uint8_t   devAddr, uint16_t  byteCount, uint8_t
*buffer, CyBool_t  isRead)
{
    CyU3PI2cPreamble_t preamble;
    uint16_t pageCount = (byteCount / glI2cPageSize);
    uint16_t resCount = glI2cPageSize;
    CyU3PReturnStatus_t status = CY_U3P_SUCCESS;
    if ((byteCount % glI2cPageSize) != 0)
    {
        pageCount ++;
        resCount = byteCount % glI2cPageSize;
    }

    while (pageCount != 0)
    {
        if (isRead) /* Read */
        {
            /* Update the preamble information. */
            preamble.length    = 4;
            preamble.buffer[0] = devAddr;
```

```
            preamble.buffer[1] = (uint8_t)(byteAddress >> 8);
            preamble.buffer[2] = (uint8_t)(byteAddress & 0xFF);
            preamble.buffer[3] = (devAddr | 0x01);
            preamble.ctrlMask  = 0x0004;

            status = CyU3PI2cReceiveBytes (&preamble, buffer, (pageCount == 1) ? resCount :
glI2cPageSize, 0);

        }
        else /* Write */
        {
            /* Update the preamble information. */
            preamble.length    = 3;
            preamble.buffer[0] = devAddr;
            preamble.buffer[1] = (uint8_t)(byteAddress >> 8);
            preamble.buffer[2] = (uint8_t)(byteAddress & 0xFF);
            preamble.ctrlMask  = 0x0000;

           status = CyU3PI2cTransmitBytes (&preamble, buffer, (pageCount == 1) ? resCount :
glI2cPageSize, 0);

            /* Wait for the write to complete. */
            preamble.length = 1;
            status = CyU3PI2cWaitForAck(&preamble, 200);
        }

        /* An additional delay seems to be required after receiving an ACK. */
        CyU3PThreadSleep (1);

        /* Update the parameters */
        byteAddress  += glI2cPageSize;
        buffer += glI2cPageSize;
        pageCount --;
    }
}
```

8.2.4.4 Reads and Writes Using DMA Transfers

A DMA channel (glI2cTxHandle) is created to transfer data to an I2C slave device, and another DMA channel (glI2cRxHandle) is created to read data from the I2C slave device. The following code reads and writes to an I2C slave using DMA transfers.

```
CyFxUsbI2cTransfer (uint16_t  byteAddress, uint8_t   devAddr, uint16_t  byteCount, uint8_t
*buffer,  CyBool_t  isRead)
{
    CyU3PDmaBuffer_t buf_p;
    CyU3PI2cPreamble_t preamble;
    uint16_t pageCount = (byteCount / glI2cPageSize);
    CyU3PReturnStatus_t status = CY_U3P_SUCCESS;
    if ((byteCount % glI2cPageSize) != 0)
    {
        pageCount ++;
    }
    /* Update the buffer address. */
    buf_p.buffer = buffer;
    buf_p.status = 0;
```

```
    while (pageCount != 0)
    {
        if (isRead) /* Read */
        {
            /* Update the preamble information. */
            preamble.length    = 4;
            preamble.buffer[0] = devAddr;
            preamble.buffer[1] = (uint8_t )(byteAddress >> 8);
            preamble.buffer[2] = (uint8_t)(byteAddress & 0xFF);
            preamble.buffer[3] = (devAddr | 0x01);
            preamble.ctrlMask  = 0x0004;

            buf_p.size = glI2cPageSize;
            buf_p.count = glI2cPageSize;

            status = CyU3PI2cSendCommand (&preamble, glI2cPageSize, isRead);
            status = CyU3PDmaChannelSetupRecvBuffer (&glI2cRxHandle, &buf_p);
            status = CyU3PDmaChannelWaitForCompletion(&glI2cRxHandle, 0x7FFF);
        }
        else /* Write */
        {
            /* Update the preamble information. */
            preamble.length    = 3;
            preamble.buffer[0] = devAddr;
            preamble.buffer[1] = (uint8_t)(byteAddress >> 8);
            preamble.buffer[2] = (uint8_t)(byteAddress & 0xFF);
            preamble.ctrlMask  = 0x0000;

            buf_p.size = glI2cPageSize;
            buf_p.count = glI2cPageSize;

            status = CyU3PDmaChannelSetupSendBuffer (&glI2cTxHandle, &buf_p);
            status = CyU3PI2cSendCommand (&preamble, glI2cPageSize, isRead);
            status = CyU3PDmaChannelWaitForCompletion(&glI2cTxHandle, 0x7FFF);
        }

        /* Update the parameters */
        byteAddress  += glI2cPageSize;
        buf_p.buffer += glI2cPageSize;
        pageCount --;

        /* Need a delay between write operations. */
        CyU3PThreadSleep (10);
    }
}
```

8.3 Serial Peripheral Interface

8.3.1 SPI Block Features

- Supports master mode only
- Supports all four SPI transfer modes (programmable SPI clock polarity and phase)
- Comlies with the Motorola SPI specification in chapter 8 of the MC68HC11 reference manual
- Supports register-based and DMA-based transfers
- Supports programmable data unit length of 4 bit, 8 bit,16 bit, and 32 bit, MSB or LSB first

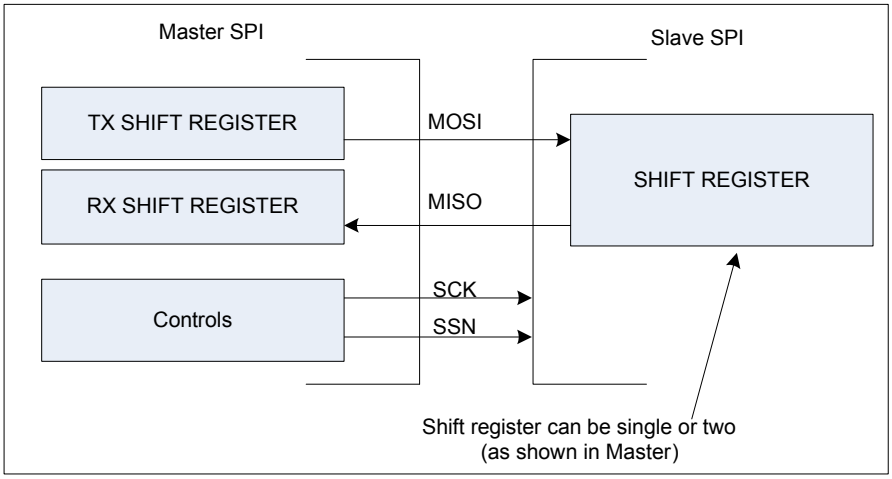
- Provides the Chip Select (CS#) signal
- Provides SPI clock up to 33 MHz
- FX3 can boot from SPI flash/ EEPROM (refer to AN76405 - EZ-USB FX3 Boot Options for more information)

8.3.2 SPI Interface Overview

The SPI bus is a synchronous serial data link interface, named by Motorola, which operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame and provides the clock. Multiple slave devices are allowed with individual slave select (chip select) lines. Sometimes SPI is called a four-wire serial bus, in contrast to three-, two-, and one-wire serial buses.

During an SPI transmission, data is transmitted (shifted out serially) and received (shifted in serially) simultaneously. The serial clock (SCK) synchronizes the shifting and sampling of information on the two serial data lines. A slave select line allows an individual slave SPI device to be selected. Slave devices that are not selected do not interfere with SPI bus activities.

Figure 8-3. SPI Bus



The FX3 SPI block operates in master mode and facilitates standard full-duplex synchronous transfers using the master out, slave in (MOSI), master in, slave out (MISO), serial interface clock (SCK), and slave select (SSN) pins. The roles of the four SPI interface pins are as follows:

- SCK: As the SPI master, FX3 provides this clock. The FX3 SPI interface clock can run up to 33 MHz. This clock is gated so that start and stop of transaction can be signaled in conjunction with the SS line. Eight clock cycles are generated on this line per transfer.
- SSN: As the SPI master, FX3 drives the slave select line in a mode-dependent fashion along with SCK to signal the start of the transaction. Depending upon mode, a slave can remain selected for multiple 8-bit transactions (based on Clock Phase (CPHA) configuration), has to toggle between transactions, or can be firmware controlled. FX3 also supports firmware control operation of the SSN line. It has a provision to drive SSN signal with a programmable lead and lag with respect to SCK at the start and end of transmission.
- MOSI: FX3 drives serial data on the MOSI line. Data can be driven on the positive or negative edge of SCK selected by a configuration bit. The selected slave samples the data on the negative or positive edges respectively.
- MISO: FX3 samples serial data on the MISO line. Data can be driven by the selected slave on the positive or negative edge of SCK governed by a configuration bit. FX3 samples the data on the negative or positive edge respectively.

The SPI block supports single and burst (DMA) data transfers. The SPI transmit and SPI receive blocks can be enabled independently using the TX_ENABLE/RX_ENABLE register settings. Independent shift registers are used for the transmit and receive paths. The shift register length can be set to values between 4 and 32 bits. By default, the TX and RX registers shift data to the left. This can be reversed, if necessary.

The FX3 SPI block can share its MOSI, MISO, and SCLK pins with more than one slave connected to the SPI bus. In this case, the SSN signal of the block cannot be used, and the multiple slave select signals need to be managed using GPIOs.

8.3.3 FX3 SPI Operations Overview

This section explains SPI operations.

8.3.3.1 Reset and Initialization

- On reset, the SPI core is placed in a disabled state. The core becomes operational only after the firmware sets the ENABLE bit in the configuration register of this block (SPI_CONFIG).
- SPI clock speed is set in the GCTL block as is the case for all FX3 blocks. GCTL_SPI_CORE_CLK is used for setting the SPI clock speed.
- Once the block is reset, the firmware monitors the ACTIVE bit of the SPI_POWER register before accessing any of the SPI block registers. The block ENABLE bit of SPI_CONFIG can be set only after the block is in the active state.

8.3.3.2 Modes Governing Transfers

The CPHA bit in the configuration register determines how a transfer starts.

1. CPHA=0: When CPHA=0, the slave starts monitoring the SCK line. The master first drives the data on the bus and then toggles the clock to the active state.. The slave samples the data as soon as it detects the idle-to-active clock edge. The master drives new data on the active-to-idle clock edge. Transfer ends when 8 bytes are transferred or aborts early by deassertion of SSN signal from the master. When the SSN toggle mode is CPHA controlled as indicated by the SSNCTRL[1:0] bits, SSN returns to idle at the end of the transfer and asserts to begin a transfer.
2. CPHA=1: When CPHA=1, the master drives data on the idle-to-active clock edge, and the slave samples data on the active-to-idle clock edge. Transfer ends when the word is transferred or aborts early by deassertion of SSN signal from the master. When the SSN toggle mode is CPHA controlled as indicated by the SSNCTRL[1:0] bits, SSN can remain asserted for multiple transfers when CPHA=1.

   The CPOL bit defines the clock polarity. CPOL=0 means that the SCK is idle low. So the idle-to-active edge in this case is the positive edge. The CPOL and CHPA bits need to be identical in all the devices connected to the SPI bus to ensure that the data is sampled half a clock cycle after it is driven on the SPI bus.

8.3.4 SSN Control Configurations

At the beginning of a transfer, SCK is in idle and SSN is deasserted. SSN needs to be asserted (SSN=0) for the slave to begin monitoring the SCK signal. SCK is driven by the SPI master only when data is being transferred. Conversely, when SCK toggles, the status of the MISO line is interpreted as data when the RX mode is enabled. Once the TX mode is enabled, it is the responsibility of the DMA producer to have data available, or SCK will go to idle. If the SPI block is configured to operate in register mode, the SPI block expects to receive data if there is space in the RX register and transmits data if it exists in the TX register. If any of these conditions are met, the SCK will not go to idle. If both are not met, then the SCK will go to idle.

The FX3 SPI supports a firmware-controlled SSN bit. When this mode is enabled, the value will be transmitted as is, except when the DESELECT bit is set high, forcing slave deselecting. The firmware is entirely responsible for managing the polarity and timing of this bit. When SSN is firmware controlled, there is no restriction on the DMA size. The firmware asserts the SSN_BIT of SPI_CONFIG, and the SSN_BIT value will hold for the entire duration of the DMA transfer (as is the case with the SPI EEPROM, where SSN remains asserted throughout). Upon completion of the transfer, the firmware can choose to deassert SSN_BIT or leave it asserted. SSN is asserted for the entire multiword transaction.

The SPI block supports various lead and lag times between SSN and SCK as specified in the SPI_CONFIG register. the lead of 0 is not supported. When the block is disabled, it must finish RX/TX of the current word.

When multiple slaves are connected to the SPI block, the SPI block handles SSN assertion for the default slave, and GPIO lines handle SSN assertion for the other slaves under firmware control (The “default slave” is the one whose chip select is connected to the FX3 SS# output pin). The DESELCT bit in SPI_CONFIG has to be set while accessing nondefault slaves, logically disconnecting the default slave. The firmware can assert and deassert the GPIO line to select alternate slaves.

8.3.5 Data Transfers

The FX3 architecture supports register-based SPI operations for small transfers and DMA-based SPI operations for larger transfers.

The data from the core side can be received either through the DMA interface or the register interface for transmitting it over the SPI bus. The registers used on the ARM side to read the data from the core and convey it to the bus are called EGRESS_DATA registers. The registers used to get the data from the external bus are called INGRESS_DATA registers. The DMA interface supports sockets that are used for moving the data between the peripheral and the USB 3.0 interconnect. The sockets used for transmitting the data to the bus are called EGRESS sockets. The DMA sockets that are used for receiving the data from the bus are called INGRESS sockets.

Apart from the SCK and SSN configurations, SPI endianness also needs to be configured; this can be done by the ENDIAN bit of the SPI_ CONFIG register.

8.4 Programming Model

SPI is a simple block that does not interpret data. The SPI programming model is similar to I2C except for the control phase. No control phase is required in SPI, so only the data phase exists.

8.4.1 Register-Based Transfers

4-byte-deep FIFOs hold egress and ingress data in its register space. Based on the status of FIFOs TX_SPACE, TX_HALF, TX_DONE, and RX_DATA, the RX_HALF flags are asserted in the SPI_STATUS and SPI_INTR registers to indicate to the firmware.

The firmware can clear the FIFOs by asserting TX_CLEAR and RX_CLEAR from the SPI_CONFIG register. Table 8-2 shows the conditions for the assertion of flags.

Table 8-2. Conditions for Assertion of Flags

FLAG Asserted	Egress FIFO State	Ingress FIFO State
TX_SPACE	Not full	-
TX_HALF	At least half empty	-
TX_DONE	Full	-
RX_DATA	-	Not empty
RX_HALF	-	At least half full

8.4.2 DMA-Based Transfers

Ingress and egress sockets can be programmed to interrupt the CPU upon transaction completion. The transceiver always raises interrupts in error conditions.

There are two separate BYTE_COUNT registers for RX and TX paths. These two registers can be used only for DMA mode and cannot be used for register mode transfers. The SPI_TX_BYTE_COUNT register specifies the number of bytes to be written out during DMA transfer. The SPI_RX_BYTE_COUNT register indicates the number of bytes received. Register mode transfers are always treated as infinite-length transfers.

Once the expected number of data bytes has been received or transmitted (indicated by SPI_TX_BYTE_COUNT or SPI_RX_BYTE_COUNT), then an end of transfer is indicated to the DMA adapter by setting the RX_DONE or TX_DONE flag to 1 respectively (of SPI_STATUS and SPI_INTR) in DMA-based transfers.

As SPI protocol is very simple, no special error handling is required apart from handling FIFO overflow and underflow.

8.5 Examples

This section shows example codes to write and read to an SPI slave device (flash memory) using register-based and DMA-based transfers from FX3. APIs to perform read and write accesses to an SPI device are provided with the FX3 SDK. Refer to the Cyu3spi.c file located at C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\firmware\lpp_source (after FX3 SDK installation) for the source code of SPI-related APIs. Refer to FX3APIGuide.pdf located at C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\doc for more details on FX3 APIs.

8.5.1 Initialize SPI Block

The CyU3PSpiInit API initializes the FX3 SPI block. The SPI block is initialized to operate at the default frequency of 1 MHz. Following is the CyU3PSPIInit function definition.

```
CyU3PSpiInit (void)
{
    /* Set the clock freqency. This should precede the SPI power up */
    CyU3PSpiSetClock (1000000);

    CyU3PMutexCreate (&glSpiLock, CYU3P_NO_INHERIT);
    /* Identify if the LPP block has been initialized. */
    CyU3PLppInit (CY_U3P_LPP_SPI, CyU3PSpiInt_ThreadHandler);

    /* Power on the SPI module */
    SPI->lpp_spi_power &= ~(1 << 31);
    CyU3PBusyWait (10);
    SPI->lpp_spi_power |= 1 << 31;
    /* Wait till the active bit is set */
    while (!(SPI->lpp_spi_power & CY_U3P_LPP_SPI_ACTIVE));

    /* Mark the module active. */
    glIsSpiActive = CyTrue;
}
```

8.5.2 Configure SPI Block

The CyU3PSpiSetConfig API configures the FX3 SPI block. Following is the code for setting the SPI bus frequency to 8 MHz. The word length is configured to 8 bits, and slave select (SSN) is configured to be controlled by the FX3 firmware.

```
CyU3PMemSet ((uint8_t *)&spiConfig, 0, sizeof(spiConfig));
    spiConfig.isLsbFirst = CyFalse;
    spiConfig.cpol       = CyTrue;
    spiConfig.ssnPol     = CyFalse;
    spiConfig.cpha       = CyTrue;
    spiConfig.leadTime   = CY_U3P_SPI_SSN_LAG_LEAD_HALF_CLK;
    spiConfig.lagTime    = CY_U3P_SPI_SSN_LAG_LEAD_HALF_CLK;
    spiConfig.ssnCtrl    = CY_U3P_SPI_SSN_CTRL_FW;
    spiConfig.clock      = 8000000;
    spiConfig.wordLen    = 8;

   CyU3PSpiSetConfig (&spiConfig, NULL);
```

8.5.3 Reads and Writes Using Register Transfers

The CyU3PSpiReceiveWords API reads data from the SPI slave device in register mode, and the CyU3PSpiTransmitBytes API writes data to an SPI slave device in register mode. Following is the code for performing reads and writes to an SPI slave using register transfers.

```
CyFxSpiTransfer (uint16_t  pageAddress, uint16_t  byteCount, uint8_t  *buffer, CyBool_t
isRead)
{
    uint8_t location[4];
    uint32_t byteAddress = 0;
    uint16_t pageCount = (byteCount / glSpiPageSize);

    if ((byteCount % glSpiPageSize) != 0)
    {
        pageCount ++;
    }

    byteAddress  = pageAddress * glSpiPageSize;
    while (pageCount != 0)
    {
        location[1] = (byteAddress >> 16) & 0xFF;       /* MS byte */
        location[2] = (byteAddress >> 8) & 0xFF;
        location[3] = byteAddress & 0xFF;               /* LS byte */

        if (isRead) /*Read*/
        {
            location[0] = 0x03; /* Read command. */

            status = CyFxSpiWaitForStatus ();
            CyU3PSpiSetSsnLine (CyFalse);
            status = CyU3PSpiTransmitWords (location, 4);
            if (status != CY_U3P_SUCCESS)
            {
                CyU3PDebugPrint (2, "SPI READ command failed\r\n");
                CyU3PSpiSetSsnLine (CyTrue);
            }

            status = CyU3PSpiReceiveWords (buffer, glSpiPageSize);
            if (status != CY_U3P_SUCCESS)
            {
                CyU3PSpiSetSsnLine (CyTrue);
            }

            CyU3PSpiSetSsnLine (CyTrue);
        }
        else /* Write */
        {
            location[0] = 0x02; /* Write command */

            status = CyFxSpiWaitForStatus ();
            CyU3PSpiSetSsnLine (CyFalse);
            status = CyU3PSpiTransmitWords (location, 4);
            if (status != CY_U3P_SUCCESS)
            {
                CyU3PDebugPrint (2, "SPI WRITE command failed\r\n");
                CyU3PSpiSetSsnLine (CyTrue);
            }
```

```
            status = CyU3PSpiTransmitWords (buffer, glSpiPageSize);
            if (status != CY_U3P_SUCCESS)
            {
                CyU3PSpiSetSsnLine (CyTrue);
            }

            CyU3PSpiSetSsnLine (CyTrue);
        }

        /* Update the parameters */
        byteAddress  += glSpiPageSize;
        buffer += glSpiPageSize;
        pageCount --;

        CyU3PThreadSleep (10);
    }
}
```

8.5.4 Reads and Writes Using DMA Transfers

A DMA channel (glSpiTxHandle) is created to transfer data to an SPI slave device, and another DMA channel (glSpiRxHandle) is created to read data from the SPI slave device. Following is the code for performing reads and writes to an SPI slave using DMA transfers.

```
CyFxSpiTransfer (uint16_t  pageAddress, uint16_t  byteCount, uint8_t  *buffer, CyBool_t
isRead)
{
    CyU3PDmaBuffer_t buf_p;
    uint8_t location[4];
    uint32_t byteAddress = 0;
    uint16_t pageCount = (byteCount / glSpiPageSize);
    if ((byteCount % glSpiPageSize) != 0)
    {
        pageCount ++;
    }

    buf_p.buffer = buffer;
    buf_p.status = 0;

    byteAddress  = pageAddress * glSpiPageSize;
    while (pageCount != 0)
    {
        location[1] = (byteAddress >> 16) & 0xFF;       /* MS byte */
        location[2] = (byteAddress >> 8) & 0xFF;
        location[3] = byteAddress & 0xFF;               /* LS byte */

        if (isRead) /* Read */
        {
            location[0] = 0x03; /* Read command. */

            buf_p.size  = glSpiPageSize;
            buf_p.count = glSpiPageSize;

            status = CyFxSpiWaitForStatus ();
            CyU3PSpiSetSsnLine (CyFalse);
            status = CyU3PSpiTransmitWords (location, 4);
            if (status != CY_U3P_SUCCESS)
            {
```

```
            CyU3PDebugPrint (2, "SPI READ command failed\r\n");
            CyU3PSpiSetSsnLine (CyTrue);
        }

        CyU3PSpiSetBlockXfer (0, glSpiPageSize);

        status = CyU3PDmaChannelSetupRecvBuffer (&glSpiRxHandle, &buf_p);
        if (status != CY_U3P_SUCCESS)
        {
            CyU3PSpiSetSsnLine (CyTrue);
        }
        status = CyU3PDmaChannelWaitForCompletion (&glSpiRxHandle, 5000);
        if (status != CY_U3P_SUCCESS)
        {
            CyU3PSpiSetSsnLine (CyTrue);
        }

        CyU3PSpiSetSsnLine (CyTrue);
        CyU3PSpiDisableBlockXfer (CyFalse, CyTrue);
    }
    else /* Write */
    {
        location[0] = 0x02; /* Write command */

        buf_p.size  = glSpiPageSize;
        buf_p.count = glSpiPageSize;

        status = CyFxSpiWaitForStatus ();
        CyU3PSpiSetSsnLine (CyFalse);
        status = CyU3PSpiTransmitWords (location, 4);
        if (status != CY_U3P_SUCCESS)
        {
            CyU3PDebugPrint (2, "SPI WRITE command failed\r\n");
            CyU3PSpiSetSsnLine (CyTrue);
        }

        CyU3PSpiSetBlockXfer (glSpiPageSize, 0);

        status = CyU3PDmaChannelSetupSendBuffer (&glSpiTxHandle, &buf_p);
        if (status != CY_U3P_SUCCESS)
        {
            CyU3PSpiSetSsnLine (CyTrue);
        }
        status = CyU3PDmaChannelWaitForCompletion(&glSpiTxHandle, 5000);
        if (status != CY_U3P_SUCCESS)
        {
            CyU3PSpiSetSsnLine (CyTrue);
        }

        CyU3PSpiSetSsnLine (CyTrue);
        CyU3PSpiDisableBlockXfer (CyTrue, CyFalse);
    }

    /* Update the parameters */
    byteAddress  += glSpiPageSize;
    buf_p.buffer += glSpiPageSize;
    pageCount --;
```

```
        CyU3PThreadSleep (10);
    }
}
```

8.6 Universal Asynchronous Receiver Transmitter

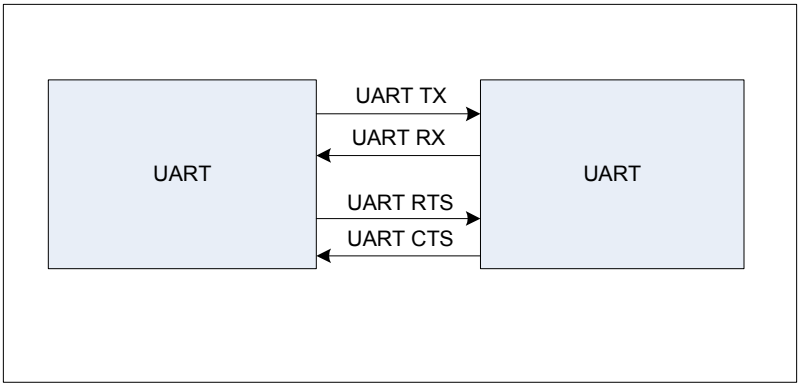
8.6.1 UART block features

- Programmable baud rate up to 4096 kbps
- Supports 10-bit and 11-bit modes.
- Supports asynchronous serial mode of communication
- Uses 16 times oversampling and uses RX bit polling to improve the error rate in transmit and receive operations
- Supports flow control by adding two hardware pins (RTS and CTS) to the UART

8.6.2 UART Overview

The UART allows asynchronous full-duplex transfer using the UART_TX and UART_RX pins. These pins are held high in the absence of transmission. Transmission starts when a 0-level start bit is transmitted and ends when a 1-level stop bit is transmitted. This represents the default 10-bit transmission mode. A 9th parity bit can be appended to data in the 11-bit transmission mode.

Figure 8-4. UART



The FX3 UART block can operate at numerous baud rates, ranging from 300 bps to 4608 Kbps and supports both one and two stop bits. Flow control pins, RTS (Request To Send) and CTS (Clear To Send) are supported in hardware. This block supports both single and burst (DMA) data transfers.

The transmitter and receiver components of the UART block can be individually enabled. Both hardware and software flow control are supported and they can be set individually on the transmitter and receiver components.

External interface devices must be used to convert the logic level signals of the UART to and from the external voltage signaling standards, such as RS-232, RS-422, and RS-485.

8.7 FX3 UART Operations Overview

The FX3 architecture supports register-based UART operations for small transfers and DMA-based UART operations for larger transfers. This section explains the UART operations.

8.7.1 Reset and Initialization

- Upon reset, the UART block is placed in a disabled state. The UART block becomes operational when firmware sets the ENABLE bit in the configuration register (UART_CONFIG).
- UART clock speed is set in the GCTL block as is the case for all FX3 blocks. GCTL_UART_CORE_CLK is used to set the UART clock speed. The UART core clock must be running at 16 times the frequency at the external interface (for example, for a 100-kHz UART clock, the UART core clock is set to 1.6 MHz). This is required to generate the external clock with the proper setup and hold with respect to the data.
- Once the block is reset, firmware monitors the ACTIVE bit in the UART_POWER register before accessing any of the UART block registers. Also, the block ENABLE bit of the UART_CONFIG register can be set only after the block is in the active state.

8.7.2 Programming Model

The UART programming model is similar to I2C except for the control phase. No control phase is required in a UART, so only the data phase exists. Data transfers can be either register based or DMA based.

8.7.3 Register-Based Transfers

ARM registers that convey core data to the data from the core and transmit it onto the bus are called EGRESS_DATA registers. The set of registers used for receiving the data from the bus are called INGRESS_DATA registers. The firmware can be notified of completion by interrupt or by polling through the status register.

4-byte-deep FIFOs hold egress and ingress data in their register spaces. Based on the status of FIFOs TX_SPACE, TX_HALF, TX_DONE and RX_DATA, RX_HALF flags are asserted in the UART_STATUS and UART_INTR registers. Firmware monitors these flags to perform data transfers.

The firmware can clear the FIFOs by asserting TX_CLEAR and RX_CLEAR bits in the UART_CONFIG register. Table 8-3 shows the conditions for the assertion of flags:

Table 8-3. Conditions for Assertion of Flags

FLAG Asserted	Egress FIFO State	Ingress FIFO State
TX_SPACE	Not full	-
TX_HALF	At least half empty	-
TX_DONE	Full	-
RX_DATA	-	Not empty
RX_HALF	-	At least half full

8.7.3.1 DMA-Based Transfers

DMA based transfers use the UART sockets. The DMA interface supports sockets that are used to move the data between the peripheral and the USB 3.0 interconnect. The sockets used for transmitting the data to the bus are called egress sockets. The DMA sockets that are used for receiving the data from the bus (over the RX line) are called ingress sockets. Ingress and egress sockets can be programmed to interrupt the CPU upon transaction completion. The transceiver always raises interrupts in error conditions.

There are two separate BYTE_COUNT registers for the RX and TX paths. These two registers can be used only for DMA mode and not for register mode transfers. The UART_TX_BYTE_COUNT register specifies the number of bytes to be written out during DMA transfer. The UART_RX_BYTE_COUNT register is used to read the number of bytes received. Register mode transfers are always treated as infinite-length transfers.

Once the expected number of data bytes has been received or transmitted (indicated by UART_TX_BYTE_COUNT or UART_RX_BYTE_COUNT), an end of transfer is indicated to the DMA adapter by setting the flag RX_DONE or TX_DONE to 1 respectively (of UART_STATUS and UART_INTR) in DMA-based transfers.

8.7.3.2 Error Conditions

Error conditions are indicated in the UART_STATUS register. All errors marked as nonsticky do not require firmware intervention while the errors marked as sticky require the firmware to reset the UART DMA sockets and reissue the command.

8.7.4 Examples

This section shows the example codes to receive and transfer data from the FX3 UART block APIs to access the UART block are provided with the FX3 SDK. Refer to the Cyu3uart.c file located at C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\firmware\lpp_source (after FX3 SDK installation) for the source code of UART-related APIs. Refer to FX3APIGuide.pdf located at C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\doc for more details on FX3 APIs.

8.7.4.1 Initialize UART Block

The CyU3PUartInit API initializes the FX3 UART block. The UART block is initialized to operate at the default baud rate of 9600. The CyU3PUARTInit function definition follows.

```
   CyU3PUartInit (void)
{

    CyU3PMutexCreate (&glUartLock, CYU3P_NO_INHERIT);

    /* Set the clock to a default value.
    * This should prcede the UART power up*/
    CyU3PUartSetClock (9600);
    /* Identify if the LPP block has been initialized. */
    CyU3PLppInit (CY_U3P_LPP_UART,CyU3PUartInt_ThreadHandler);
    /* Hold the UART block in reset. */
    UART->lpp_uart_power &= ~(1 <<  31);
    CyU3PBusyWait (10);
    UART->lpp_uart_power |= (1 <<  31);

    /* Wait for the active bit to be asserted by the hardware */
    while (!(UART->lpp_uart_power & CY_U3P_LPP_UART_ACTIVE));

    /* Mark the module as active. */
    glIsUartActive = CyTrue;
}
```

8.7.4.2 FX3 Firmware to Send UART Messages and to Receive Fixed Bytes of Text

The following code prints messages on a user interface like HyperTerminal or TeraTerm and receives a fixed number of bytes of text from the same user interface.

- UART block of FX3 is configured for the following settings: Baud rate: 115200, one stop bit, no parity, no flow control, both TX and RX paths enabled, and DMA transfers enabled.
- uartIntrCb callback function to receive UART events is registered using the CyU3PUartSetConfig API.
- Value 0xFFFFFFFFU is passed for the TX path to specify infinite or indefinite data transmission.
- Value 4 is passed for the RX path to specify that only 4 bytes of data transfer is allowed. This means FX3 firmware can process the input UART messages only after 4 characters are entered in TeraTerm. For example, if you enter 3 then the

FX3 firmware waits for you to enter another character. If you enter 5 then the FX3 firmware can read the first 4 characters and wait for 3 more characters to be entered on TeraTerm.

```
    CyU3PUartConfig_t uartConfig;
    CyU3PReturnStatus_t apiRetStatus = CY_U3P_SUCCESS;

    /* Initialize the UART for printing debug messages */
    apiRetStatus = CyU3PUartInit();

    /* Set UART configuration */
    CyU3PMemSet ((uint8_t *)&uartConfig, 0, sizeof (uartConfig));
    uartConfig.baudRate = CY_U3P_UART_BAUDRATE_115200;
    uartConfig.stopBit = CY_U3P_UART_ONE_STOP_BIT;
    uartConfig.parity = CY_U3P_UART_NO_PARITY;
    uartConfig.txEnable = CyTrue;
    uartConfig.rxEnable = CyTrue;
    uartConfig.flowCtrl = CyFalse;
    uartConfig.isDma = CyTrue;

    apiRetStatus = CyU3PUartSetConfig (&uartConfig, uartIntrCb);

    /* Set the UART transfer to a really large value. */
    apiRetStatus = CyU3PUartTxSetBlockXfer (0xFFFFFFFF);

    /* Set the UART transfer to 4 */
    apiRetStatus = CyU3PUartRxSetBlockXfer (4);

    /* Initialize the debug module. */
    apiRetStatus = CyU3PDebugInit (CY_U3P_LPP_SOCKET_UART_CONS, 8);
}
```

Now you can use the CyU3PDebugPrint API to print debug messages on TeraTerm.

A DMA manual channel needs to be created between the UART producer socket and the CPU consumer socket to get data into the FX3 from a user interface like TeraTerm. A minimum-size DMA buffer (16 bytes) is allocated for this data path.

```
        dmaCfg.size  = 16;
        dmaCfg.count = 1;
        dmaCfg.prodSckId = CY_U3P_LPP_SOCKET_UART_PROD;
        dmaCfg.consSckId = CY_U3P_CPU_SOCKET_CONS;
        dmaCfg.dmaMode = CY_U3P_DMA_MODE_BYTE;
        /* Enabling the callback for produce event. */
        dmaCfg.notification = CY_U3P_DMA_CB_PROD_EVENT;
        dmaCfg.cb = 0;
        CyU3PDmaChannelCreate (&glChHandleUARTtoCPU, CY_U3P_DMA_TYPE_MANUAL_IN, &dmaCfg);
        CyU3PDmaChannelSetXfer (&glChHandleUARTtoCPU, 0);
```

The UART callback function definition follows. The firmware needs to check for the RX_DONE event to learn whether the required number of bytes was received. The DMA channel created for the RX path is wrapped up after receiving the RX_DONE event. The received data is printed back on the HyperTerminal and the DMA buffer is cleared for receiving the next set of data from the HyperTerminal.

```
void uartIntrCb(CyU3PUartEvt_t evt, CyU3PUartError_t error)
{

       if (evt == CY_U3P_UART_EVENT_RX_DONE)
```

```
        {
                CyU3PDmaChannelSetWrapUp (&glChHandleUARTtoCPU);
                CyU3PDmaChannelGetBuffer (&glChHandleUARTtoCPU, &inBuf_p, CYU3P_NO_WAIT);
                CyU3PDebugPrint (4, "%d %d %d %d\n\r",inBuf_p.buffer[0],inBuf_p.buf-
fer[1],inBuf_p.buffer[2],
                                                    inBuf_p.buffer[3]);
                CyU3PDmaChannelDiscardBuffer (&glChHandleUARTtoCPU);
                CyU3PUartRxSetBlockXfer (4);
        }
}
```

8.8 Integrated Interchip Sound Interface

8.8.1 I2S Block Features

- Supports I2S transmitter function as master
- Transmits data at 8, 16, 32, 44.1, 48, 96, 192 kHz
- Supplies SCK at 64*WS (Word Select) frequencies
- Transmits 8-, 16-, 24-, 32-bit data values
- Default behavior is:
  - WS = 0 means left channel
  - Data values should be zero padded at LSB to 32 bits
  - MSB is transmitted first
- Nondefault behavior is controlled by bits in the configuration register.
- FX3 is capable of supplying MCLK derived from the PLL when configured as output; however, this clock is not expected to be of high-fidelity quality.

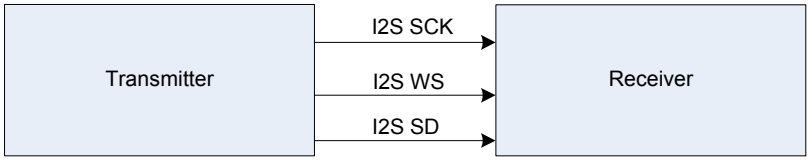
8.8.2 I2S Overview

I2S is a serial bus standard used to connect digital audio devices. It is used to communicate PCM audio data between integrated circuits in an electronic device. The I2S bus separates the clock and serial data signals, resulting in a lower jitter than a normal communications system that recovers the clock from the data stream.

FX3 supports left-justified and right-justified variants of the I2S protocol as defined in the TLV320AIC31 datasheet by Texas Instruments.

The I2S bus handles only audio data, while the other signals such as subcoding and control are transferred separately. To minimize the number of pins required and to keep the interface simple, a 3-line serial bus is used. As shown in Figure 8-5, it consists of a signal for two time-multiplexed data channels (I2S SD), a word select line (I2S WS), and a clock line (I2S SCK). The roles of the three I2S interface pins and MCLK are as follows:

Figure 8-5. I2S Interface



1. WS: The word select line indicates the channel being transmitted:
   WS = 0; channel 1 (left)
   WS = 1; channel 2 (right)

Typical frequencies for WS are 44.1 kHz and 48 kHz, which represent sampling rates for audio data. The I2S master is expected to supply this clock.

2. SCK: Provides the clock for exchanging the one-bit serial data. Audio data can be an 8-, 16-, 24-, or 32-bit value. The I2S standard specifies that the SCK should be a fixed multiple of WS (64X), and if fewer than 32 bits are transmitted, the LSBs should be padded with zeroes. However, some implementations are known to stop SCK, which is controlled by a configuration bit. The I2S master is expected to supply this clock.
3. SD: This line carries the serial data clocked using SCK. Most Significant Bit (MSB) is transmitted first. The standard specifies that the LSBs should be zero padded to transmit a word (32-bit value). Either the master or a slave can drive this line.
4. MCLK: Many codecs expect the master to supply them with a 256X WS master clock. This clock is carried over the MCLK pin. For high-quality audio, MCLK is configured as an input driven by a crystal.

   The I2S block can be configured to support different audio word widths, endianness, number of channels, and data rate. By default, the standard protocol is most significant bit first, but this can be reversed. FX3 also supports the left-justified and right-justified variants of the protocol. When the block is enabled in left-justified mode, the left channel audio sample is sent first on the I2S SD line.

   In mono mode, the "left data" is sent to both channels on the receiver (WordSelect = Left and WordSelect = Right). In the variable SCK mode, WS (WordSelect) toggles every Nth edge of SCK, where N is the word width chosen. In fixed SCK mode, however, WS toggles every 32nd SCK edge. In this mode, the audio sample is zero padded to 32 bits.

8.8.3 FX3 I2S Operations Overview

FX3 supports two types of I2S operations; one word at a time (SGL_LEFT_DATA, SGL_RIGHT_DATA) for small transfers and DMA-based I2S operations for larger transfers. Two special modes of operation, Mute and Pause, are supported. When Mute is held asserted, the DMA data is ignored, and zeros are transmitted instead. When paused, the DMA data flow into the block is stopped, and zeros are transmitted over the interface.

- I2S DMA sockets: I2S exchanges data with the system memory through two 32-bit DMA sockets. The left and right channels have separate sockets as most of the audio algorithms generate data in that manner. The sockets are configured to be a word stream, which means that their readiness signals the availability of a word of data or memory availability for receiving a word of data. However, they are typically configured to have larger buffers, and the producer is expected to generate a larger quantity of data at a time to minimize the latency while switching the DMA sockets. The sockets supply byte-packed data; 8-, 16-, 24-, and 32 bit words are supported.
- I2S transceiver: The I2S transceiver will not begin transmission until both the left and right egress socket have data. It then pulls data in the order determined by the MODE_WS bit, serializes it, and clocks it out using SCK. The block has a clock generation mechanism to generate WS and SCK at the specified rates.

8.8.4 Programming Model

This section explains the I2S operations.

8.8.4.1 Start Transmission

1. The firmware creates a DMA channel between a producer socket and the I2S consumer socket.
2. The firmware enables the I2S block.
3. As soon as the I2S bock is enabled and the socket is ready, the I2S clock starts toggling and data transmission occurs.

8.8.4.2 Mute Condition

1. When Mute is asserted, discard the DMA data, andtransmit zeros instead.
2. When Mute is deasserted, do not discard the DMA data, and transmit it.

8.8.4.3 Pause Condition

1. When Pause is asserted, transmit the current word pair, and then stop the data flow into the I2S block and transmit zeros. However, receive the data and buffer it for the previous request(s). Pause has priority over Mute if both are asserted.
2. Assert the PAUSED bit in the I2S_STATUS register.

8.8.4.4 Buffer Underflow

1. When the DMA buffer runs out of data but end of transfer is not high, treat this as a Mute condition. That is, transmit zeros until both channels have data in stereo mode and the left channel has data in mono mode.
2. Assert the NO_DATA bit in the I2S_STATUS register.
3. Raise an interrupt if interrupts are enabled in the firmware.
4. The clock continues to run when interrupted.
5. Deassert the NO_DATA bit in the I2S_STATUS register upon receiving the socket active signal from the DMA.
6. When only one channel is underflowing, treat it is as an underflow for both the channels to maintain synchronization. This means mute both channels (discard DMA data and transmit zeros from the channel with data) until data for both channels is available.

8.8.4.5 Stop Event

1. The firmware can signal a stop event by disabling the block anytime.
2. Upon receiving this event, finish transmitting the current word pair and shift-in/parallel load zeros in the serializer.
3. Discard any untransmitted data in the pipelines.
4. After aborting or completing an existing transaction, a stop event must be generated before starting a new transaction.

8.8.4.6 Fixed Clock Mode

In fixed clock (FIXED_SCK) mode, SCK is always 64xWS, and left and right data is padded to 32 bits. When FIXED_SCK=0 (called continuous transmission mode),

SCK=2 x (number of bits in sample) x WS.

The I2S master is expected to supply this clock. FX3 derives the clock by dividing MCLK and gating it when transmission ends. FIXED_SCK=0 is not supported when the number of bits per sample is set to 18 and 24, since MCLK is 256xWS and 256 is not divisible by 36 or 48. The main difference in the FIXED and non-FIXED SCK mode is the frequency of the SCK and the padding behavior.

8.8.4.7 Data Shift Mode

The I2S protocol is specified to be MSB first, but the FX3 I2S block supports programmable bit order so that it can interface with different devices.

8.8.4.8 Padding

Padding refers to adding extra zero bits to a sample received from the DMA to make a 32-bit value. This value is then transmitted from left to right. No padding is necessary when FIXED_CLOCK=0, since the exact number of bit clock transitions is available.

8.8.4.9 Error Conditions

Error conditions are indicated in the I2S_STATUS register. Errors denoted as nonsticky do not require firmware intervention. Errors marked sticky require the firmware to reset the I2S DMA sockets and reissue the command.

8.8.4.10 Examples

This section shows an example code to transfer data from the FX3 I2S block. APIs to access the I2S block are provided with the FX3 SDK. Refer to the Cyu3i2s.c file located at C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\firmware\lpp_source (after FX3 SDK installation) for the source code of the UART=related APIs. Refer to FX3APIGuide.pdf located at C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\doc for more details about the FX3 APIs.

8.8.4.11 Initialize I2S Block

The CyU3PI2sInit API initializes the FX3 I2S block. The I2S block is initialized to operate at the default sampling rate of 8 kHz. The CyU3PI2sInit function definition follows.

```
CyU3PI2sInit (void)
{
     CyU3PMutexCreate (&glI2sLock, CYU3P_NO_INHERIT);

    /* Set the clock frequency. This should precede the I2S power up. Setting it to stereo
fixed mode. */
    status = CyU3PI2sSetClock (CY_U3P_I2S_DEFAULT_SAMPLE_RATE * 64);
    /* Identify if the LPP block has been initialized. */
    status = CyU3PLppInit (CY_U3P_LPP_I2S,CyU3PI2sInt_ThreadHandler);

    /* Power on the I2S module */
    I2S->lpp_i2s_power &= ~(1 << 31);
    CyU3PBusyWait (10);
    I2S->lpp_i2s_power |= 1 << 31;
    while (!(I2S->lpp_i2s_power & CY_U3P_LPP_I2S_ACTIVE));

    /* Mark the module active. */
    glIsI2sActive = CyTrue;
}
```

8.8.4.12 Configure I2S Interface

The CyU3PI2sSetConfig API configures the FX3 I2S block. Following is the code for setting the I2S interface sampling frequency of 44.1 kHz. A sample word is configured to 16 bits, and DMA transfers are enabled.

```
/* Configure the I2S interface. */
    CyU3PMemSet ((uint8_t *)&i2sCfg, 0, sizeof (i2sCfg));
    i2sCfg.isMono = CyFalse;
    i2sCfg.isLsbFirst = CyFalse;
    i2sCfg.isDma = CyTrue;
    i2sCfg.padMode = CY_U3P_I2S_PAD_MODE_NORMAL;
    i2sCfg.sampleRate = CY_U3P_I2S_SAMPLE_RATE_44_1KHz;
    i2sCfg.sampleWidth = CY_U3P_I2S_WIDTH_16_BIT;
    status = CyU3PI2sSetConfig (&i2sCfg, NULL);
```

8.8.4.13 Transferring Data from USB Interface to I2S Interface Using DMA Transfers

The FX3 firmware source code to transfer the data from EP1 OUT to the left channel and data received on EP2 OUT to the right channel of an I2S amplifier/speaker follows. The DMA buffer size is defined based on the USB speed: 64 for Full Speed, 512 for High Speed, and 1024 for SuperSpeed. Auto DMA channels are created between two sockets of the USB and the left and right I2S sockets.

```
    /* Identify the usb speed. Once that is identified, create a DMA channel and start the
transfer on this. Based on the Bus Speed configure the endpoint packet size */
    switch (usbSpeed)
    {
        case CY_U3P_FULL_SPEED:
            size = 64;
            break;

        case CY_U3P_HIGH_SPEED:
            size = 512;
```

```
            break;

        case  CY_U3P_SUPER_SPEED:
            size = 1024;
            break;

        default:
            break;
    }

    CyU3PMemSet ((uint8_t *)&epCfg, 0, sizeof (epCfg));
    epCfg.enable = CyTrue;
    epCfg.epType = CY_U3P_USB_EP_BULK;
    epCfg.burstLen = 1;
    epCfg.streams = 0;
    epCfg.pcktSize = size;

    /* Producer 1 endpoint configuration */
    status = CyU3PSetEpConfig(CY_FX_EP_PRODUCER_1, &epCfg);

    /* Producer 2 endpoint configuration */
    status = CyU3PSetEpConfig(CY_FX_EP_PRODUCER_2, &epCfg);

    /* Create a DMA Auto channel between two sockets of the U port and the L and R I2S sock-
ets. DMA size is set based on the USB speed. */
    dmaCfg.size = size;
    dmaCfg.count = 8;
    dmaCfg.prodSckId = CY_FX_EP_PRODUCER_1_SOCKET;
    dmaCfg.consSckId = CY_U3P_LPP_SOCKET_I2S_LEFT;
    dmaCfg.dmaMode = CY_U3P_DMA_MODE_BYTE;
    dmaCfg.notification = 0;
    dmaCfg.cb = NULL;
    dmaCfg.prodHeader = 0;
    dmaCfg.prodFooter = 0;
    dmaCfg.consHeader = 0;
    dmaCfg.prodAvailCount = 0;
    CyU3PDmaChannelCreate (&glI2sLeftCh, CY_U3P_DMA_TYPE_AUTO, &dmaCfg);

    dmaCfg.prodSckId = CY_FX_EP_PRODUCER_2_SOCKET;
    dmaCfg.consSckId = CY_U3P_LPP_SOCKET_I2S_RIGHT;
    CyU3PDmaChannelCreate (&glI2sRightCh, CY_U3P_DMA_TYPE_AUTO, &dmaCfg);

    /* Flush the Endpoint memory */
    CyU3PUsbFlushEp(CY_FX_EP_PRODUCER_1);
    CyU3PUsbFlushEp(CY_FX_EP_PRODUCER_2);

    /* Set DMA Channel transfer size to infinite. */
    status = CyU3PDmaChannelSetXfer (&glI2sLeftCh, 0);
    status = CyU3PDmaChannelSetXfer (&glI2sRightCh, 0);
```

8.9 GPIO

8.9.1 GPIO Features

- All 61 pins can be configured as simple GPIOs.
- Up to 8 pins can be configured as complex GPIOs.
- Programmable drive strength for output I/Os.
- Supports internal weak pull-up or pull-down.

8.9.2 GPIO Overview

General-purpose I/O pins are a special (simple) case of low-performance peripherals that do not need a DMA capability. Several FX3 pins can function as GPIOs. All GPIOs together are considered a single low-performance peripheral. Each pin is multiplexed to support other blocks (such as UART, SPI, and so on). By default, pins are allocated in groups to either one block or the other (Blk I/O) depending on the interface mode in their respective power domain. In a typical application, not all FX3 blocks are used. Also, not all the pins of blocks being used are utilized. Unused pins in each block may be overridden as a simple or complex GPIO pin on a pin-by-pin basis.

Simple GPIOs provide software-controlled and observable input and output capability. They also can raise interrupts. Complex GPIOs support a variety of time-based functions. They work off either a slow or a fast clock. Complex GPIOs can also be used as general-purpose timers by the firmware. Only eight pins can be configured as complex GPIOs at a time.

8.9.3 Programming Model

This section explains the GPIO operations.

8.9.3.1 Reset and Initialization

On reset, all the blocks of the GPIO core are placed in a disabled state. The core becomes operational only after one of the ENABLE bits in the configuration registers of this block (GPIO_SIMPLE or PIN_STATUS) is set by the firmware.

The GPIO core's clock speed is set in the GCTL block as is the case for all FX3 blocks. Three major clock settings are required for GPIOs:

GCTL_ FAST _CORE_CLK: Master GPIO clock derived out of PLL system clock. It can be set to a maximum of 200 MHz.

GCTL_ SLOW _CORE_CLK: Slow clock derived out of GCTL_ FAST _CORE_CLK. It can be set to a maximum of 1 MHz.

GCTL_SIMPLE_CLK: Derived out of GCTL_ FAST _CORE_CLK. Applicable to simple GPIOs only. GCTL_ FAST _CORE_CLK can be divided by 2, 4, 16, and 64 only.

Once the block is reset, the ACTIVE bit of the GPIO_POWER register is monitored by the firmware before accessing any of the GPIO block registers. Also, the block ENABLE bit of GPIO_CONFIG can be set only after the block is in the active state.

Table 8-4 shows the various modes supported by a complex GPIO. It uses the values from three registers. TIMER is the value of the PIN_TIMER register, THRESHOLD is the value of the PIN_THRESHOLD register, and MODE is from the PIN_STATUS register.

Table 8-4. Modes Supported by Complex GPIO

Mode	Name	Description of Functionality Achieved
0	STATIC	Use as simple GPIO, drive output to static value and/or observe input value.
1	TOGGLE	When TIMER=THRESHOLD, change value of output.
2	SAMPLENOW	Set THRESHOLD=TIMER now. This is a method to observe a current value of the running TIMER register, which is not readable.
3	PULSENOW	■ Toggle the pin value immediately. ■ Set TIMER=0. ■ When TIMER=THRESHOLD, toggle output back. ■ Set MODE=STATIC.
4	PULSE	■ Toggle value when TIMER=0. ■ When TIMER=THRESHOLD, toggle output back. ■ Set MODE=STATIC.
5	PWM	Toggle value when TIMER=0. When TIMER=THRESHOLD, toggle output back.
6	MEASURE_LOW	Measure the time a signal is low and continue doing so. Set TIMER=0 on negative edge (negedge) of input. Set THRESHOLD=TIMER on positive edge (posedge) of input.
7	MEASURE_HIGH	Measure the time a signal is high and continue doing so. Set TIMER=0 on posedge of input. Set THRESHOLD=TIMER on negedge of input.
8	MEASURE_LOW_ONCE	Measure the time a signal is low once and then stop. Set TIMER=0 on negedge of input. Set THRESHOLD=TIMER on posedge of input. Set MODE=STATIC.
9	MEASURE_HIGH_ONCE	Measure the time a signal is high once and then stop. Set TIMER=0 on posedge of input. Set THRESHOLD=TIMER on negedge of input. Set MODE=STATIC.
10	MEASURE_NEG	Measure when a signal goes low and continue doing so. Set THRESHOLD=TIMER on negedge.
11	MEASURE_POS	Measure when a signal goes high and continue doing so. Set THRESHOLD=TIMER on posedge.
12	MEASURE_ANY	Measure when a signal changes and continue doing so. Set THRESHOLD=TIMER on posedge or negedge.
13	MEASURE_NEG_ONCE	Measure when a signal goes low once and then stop. Set THRESHOLD=TIMER on negedge. Set MODE=STATIC.
14	MEASURE_POS_ONCE	Measure when a signal goes high once and then stop. Set THRESHOLD=TIMER on posedge. Set MODE=STATIC.
15	MEASURE_ANY_ONCE	Measure when a signal changes and then stop. Set THRESHOLD=TIMER on negedge or posedge. Set MODE=STATIC.

8.9.4 Examples

This section shows an example code to configure and use FX3 GPIOs.. APIs to access the GPIO block are provided with the FX3 SDK. Refer to the Cyu3gpio.c file located at C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\firmware\lpp_source (after FX3 SDK installation) for the source code of GPIO-related APIs. Refer to FX3APIGuide.pdf located at C:\Program Files (x86)\Cypress\EZ-USB FX3 SDK\1.3\doc for more details on the FX3 APIs.

8.9.4.1 Initialize GPIO Block

The CyU3PGpioInit API initializes the FX3 GPIO block, and its definition follows.

```
CyU3PGpioInit (CyU3PGpioClock_t *clk_p, CyU3PGpioIntrCb_t irq)
```

```
{

    /* Store the interrupt handler function pointer. */
    CyU3PRegisterGpioCallBack(irq);

    /* If the boot firmware has left the GPIO block ON, do not reset it. */
    if ((CyU3PLppGpioBlockIsOn () == CyFalse) || ((GPIO->lpp_gpio_power &

CY_U3P_LPP_GPIO_ACTIVE) == 0))
    {
        status = CyU3PGpioSetClock (clk_p);

        /* Register the fact that GPIO has been started with the FX3 API library. */
        status = CyU3PLppInit (CY_U3P_LPP_GPIO, CyU3PGpioInt_Handler);

        /* Power the GPIO block ON, and wait for it to be active. */
        GPIO->lpp_gpio_power &= ~(1 << 31);
        CyU3PBusyWait (10);
        GPIO->lpp_gpio_power |= (1 << 31);
        while (!(GPIO->lpp_gpio_power & CY_U3P_LPP_GPIO_ACTIVE));
    }
    else
    {
        /* GPIO block is already ON. Just register the block with the FX3 API library. */
        status = CyU3PLppInit (CY_U3P_LPP_GPIO, CyU3PGpioInt_Handler);
    }

    /* Update the status flag. */
    glIsGpioActive = CyTrue;
}
```

8.9.4.2 *Configure GPIO[45] as Input Pin and GPIO[21] as Output Pin*

The I/O matrix needs to be configured to use simple and complex GPIOs. I/Os that are not configured for peripheral interfaces can be used as GPIOs. This selection must be explicitly made. Otherwise, these lines will be configured per their default function. This selection is made via four 32-bit bit masks, where each I/O is represented in C by (1 << IO number). In this case, GPIO[45] is used as an input pin and is selected during the I/O matrix configuration. GPIO[21] is also used but cannot be selected here, as it is part of the GPIF II I/Os (CTL4). If this I/O is not used with the GPIF II interface, it can be overridden to become a GPIO by invoking the CyU3PDeviceGpioOverride call.

```
    io_cfg.gpioSimpleEn[0]  = 0; // bit positions 31:0
    io_cfg.gpioSimpleEn[1]  = 0x00002000; /* GPIO 45 */(bit positions 63:32)
    io_cfg.gpioComplexEn[0] = 0;
    io_cfg.gpioComplexEn[1] = 0;
   CyU3PDeviceConfigureIOMatrix (&io_cfg);
```

The clock parameters for the GPIO block are defined using the CyU3PGpioInit API. Three clocks are associated with the FX3 GPIO core clock. The FAST clock is derived from the FX3 system clock, and its frequency is determined by the clkSrc and fastClkDiv parameters. The SLOW clock is derived from the FAST clock, and its frequency is determined by the slowClkDiv value. The operating clock for various complex GPIO timers can be selected from among the FAST clock, the SLOW clock, and a fixed 32-kHz frequency. All simple GPIOs function (are updated or sampled) based on a separate clock, which is also derived from the FAST clock. The frequency of this clock is determined by the simpleDiv value. The FAST clock is configured as half of the system clock, and the SLOW clock is disabled (set to zero). Interrupt callback can be registered using CyU3PGpioInit, but this example does not use it.

```
/* Init the GPIO module */
    gpioClock.fastClkDiv = 2;
    gpioClock.slowClkDiv = 0;
    gpioClock.simpleDiv = CY_U3P_GPIO_SIMPLE_DIV_BY_2;
    gpioClock.clkSrc = CY_U3P_SYS_CLK;
    gpioClock.halfDiv = 0;

    apiRetStatus = CyU3PGpioInit(&gpioClock, NULL);
```

The CyU3PGpioSetSimpleConfig API configures a simple GPIO. The code for configuring GPIO[45] is as follows. GPIO[45] is configured as input, and interrupt is enabled for both edges of GPIO[45].

```
/* Configure GPIO 45 as input with interrupt enabled for both edges */
    gpioConfig.outValue = CyTrue;
    gpioConfig.inputEn = CyTrue;
    gpioConfig.driveLowEn = CyFalse;
    gpioConfig.driveHighEn = CyFalse;
    gpioConfig.intrMode = CY_U3P_GPIO_INTR_BOTH_EDGE;
    CyU3PGpioSetSimpleConfig(45, &gpioConfig);
```

GPIO[21] needs to be overridden, as this pin is associated with the GPIF II control signal. The CyU3PDeviceConfigureIOMatrix call cannot select the I/O as a GPIO, as it is part of the GPIF II I/Os. An override API call must be made with caution, as this will change the functionality of the pin. If the I/O line is used as part of GPIF II and is connected to an external device, then the line will no longer behave as a GPIF II I/O. Here the CTL4 line is not used by the GPIF II block, so it is safe to override.

```
   CyU3PDeviceGpioOverride (21, CyTrue);
```

```
GPIO[21] can be configured as an output using the following code. No interrupt is enabled
corresponding to this I/O change.
    gpioConfig.outValue = CyFalse;
    gpioConfig.driveLowEn = CyTrue;
    gpioConfig.driveHighEn = CyTrue;
    gpioConfig.inputEn = CyFalse;
    gpioConfig.intrMode = CY_U3P_GPIO_NO_INTR;
   CyU3PGpioSetSimpleConfig(21, &gpioConfig);
```

Use the followingcode to get the status of GPIO[45].

```
/* Get the status of the pin */
CyU3PGpioGetValue (45, &gpioValue);
```

Use the following code to set the value of GPIO[21].

```
/* Set the GPIO 21 to high */
CyU3PGpioSetValue (21, CyTrue);
```

GPIO[21] can be driven low by changing the second parameter of the CyU3PGpioSetValue API.

```
        /* Set the GPIO 21 to low */
CyU3PGpioSetValue (21, CyFalse);
```

8.9.4.3 Configure GPIO[50] to Generate PWM Output

The I/O matrix needs to be configured to use simple and complex GPIOs. I/Os that are not configured for peripheral interfaces can be used as GPIOs. This selection must be explicitly made. Otherwise, these lines will be configured per their default function. This selection is made via four 32-bit bit masks, where each I/O is represented by (1 << IO number). In this case, GPIO[50] is used as an input pin and is selected during the I/O matrix configuration.

```
io_cfg.gpioSimpleEn[0]  = 0;
io_cfg.gpioSimpleEn[1]  = 0;
/* GPIO[50] is used as complex GPIO. */
io_cfg.gpioComplexEn[0] = 0;
io_cfg.gpioComplexEn[1] = 0x00040000;
CyU3PDeviceConfigureIOMatrix (&io_cfg);
```

Initialize the GPIO module. The GPIO block runs with a fast clock at SYS_CLK / 2, and a slow clock is not used. In this case, the SYS_CLK runs at 403 MHz

```
gpioClock.fastClkDiv = 2;
gpioClock.slowClkDiv = 0;
gpioClock.simpleDiv = CY_U3P_GPIO_SIMPLE_DIV_BY_2;
gpioClock.clkSrc = CY_U3P_SYS_CLK;
gpioClock.halfDiv = 0;
CyU3PGpioInit(&gpioClock, NULL);
```

CyU3PGpioSetComplexConfig configures a complex GPIO. The code for configuring GPIO[50] as a PWM output with a 25 percnet duty cycle is as follows.

```
/* Configure GPIO 50 as PWM output */
  gpioConfig.outValue = CyFalse;
  gpioConfig.inputEn = CyFalse;
  gpioConfig.driveLowEn = CyTrue;
  gpioConfig.driveHighEn = CyTrue;
  gpioConfig.pinMode = CY_U3P_GPIO_MODE_PWM;
  gpioConfig.intrMode = CY_U3P_GPIO_NO_INTR;
  gpioConfig.timerMode = CY_U3P_GPIO_TIMER_HIGH_FREQ;
  gpioConfig.timer = 0;
  gpioConfig.period = CY_FX_PWM_PERIOD;      /* (201600 - 1) */
  gpioConfig.threshold = CY_FX_PWM_25P_THRESHOLD; /* (50400 - 1) */
  apiRetStatus = CyU3PGpioSetComplexConfig(50, &gpioConfig);
```

The following code changes the PWM duty cycle to 75 percent.

```
/* Change the PWM duty cycle to 75%. */
CyU3PGpioComplexUpdate (50, CY_FX_PWM_75P_THRESHOLD, CY_FX_PWM_PERIOD); /*
CY_FX_PWM_75P_THRESHOLD  = (151200 - 1) */
```

9. Storage Ports

FX3S features an integrated storage controller that supports up to two independent SD/ MMC/SDIO devices on its two storage ports (S0 and S1). Both storage ports support the following specifications:

- MMC system specification, MMCA Technical Committee, version 4.41
- SD specification, version 3.0
- SDIO host controller compliant with SDIO specification version 3.00

9.1 Storage Interface Block Features

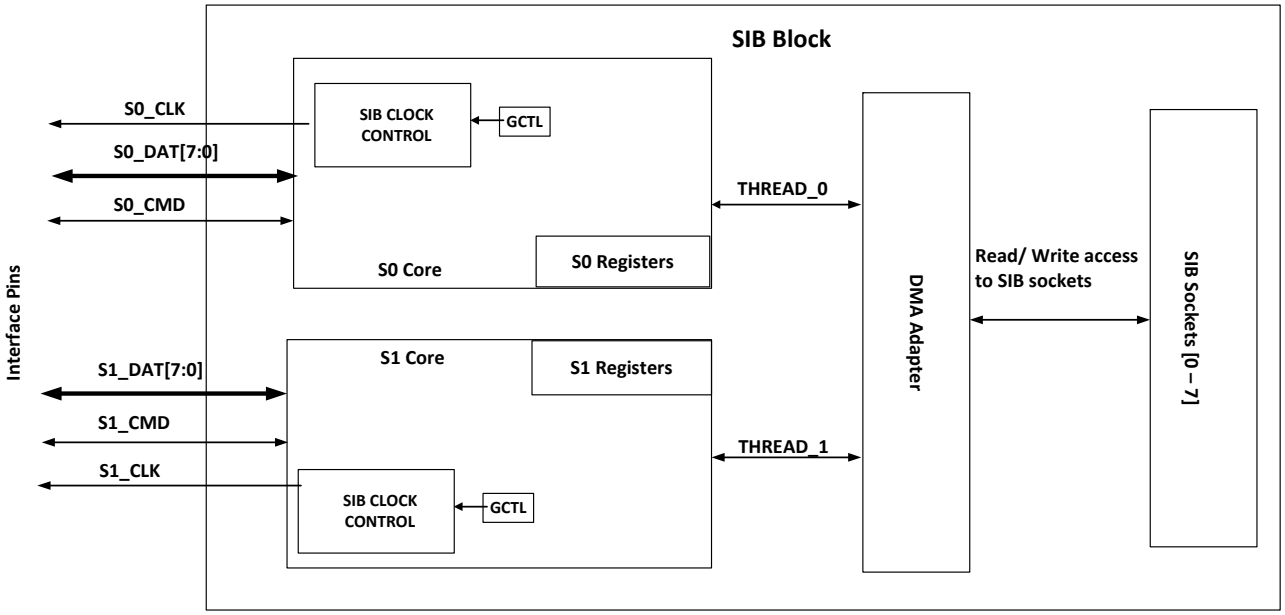
The Storage Interface Block (SIB) contains the interface logic for the storage port. The SIB offers the following features:

- Supports SD3.0/eMMC4.41/SDIO 3.0.
- Supports two independent ports.
- 8 bidirectional sockets.
- 1-, 4-, and 8-bit bus width on the card interface
- Card insertion and removal detection
- Write protection
- SD3.0 voltage switch sequence from 3.3 V to 1.8 V
- SD3.0 host tuning feature
- Normal and alternate eMMC boot as defined in eMMC specification, version 4.41
- Dynamic clock stop to avoid overrun and underrun and power saving
- SDR and DDR mode of operation
- Max operating frequency
  - 104-MHz SDR
  - 52-MHz DDR
  - Throughput of 180 MBps including both ports
  - SDIO interrupt feature as specified in the SDIO specification version 2.00 (January 30, 2007)
  - SDIO read-wait and suspend-resume features as defined in the SDIO specification version 2.00 (January 30, 2007)

9.2 Block Diagram

The SIB is the host controller for external storage devices. It connects storage devices to the ARM processor, USB and low-bandwidth peripherals such as SPI, UART, I2C, I2S, and GPIO. The SIB generates commands and accepts responses at the SD/MMC interface based on the configuration provided by the firmware. SIB contains two storage port controllers (S0 and S1) that can be independently configured to support different protocols.

Figure 9-1. SIB Block Diagram



SIB consists of three major subblocks.

- S0 core
- S1 core
- DMA adapter

The S0 and S1 core are functionally independent, thus can be configured independently. This also allows to clock gate one of the cores based on the active port. All protocol-related translation and error handling is done by the respective cores. The cores provide a simple request/data interface to the DMA backbone.

The S0/S1 core contains a CLOCK CONTROL sub-block. The clock control block accepts the core clock as the input and generates the clocks required for the SIB. This block instantiates the Delay-Locked Loop (DLL) and performs the following functions:

- Gating of clock when sockets cannot accept data
- Gating of clock when sockets do not have data
- Gating of clock when data transfer of all blocks has been completed
- Gating of DLL master clock when the block is IDLE
- Gating of clock only on block boundaries
- Generating pin sample clocks and core clocks used by the core
- Keeping track of when DLL locks and loses lock

Each DLL is used to:

- Internally generate 16 equally spaced phases of input signal. This allows clock control in 22.5º increments. A phase picker circuit at the DLL output stage picks up to a maximum of 4 of these 16 output phases to guarantee no setup and hold time issues on-chip. The input reference frequency range of the signal input clock is 25-208 MHz. This is utilized for SD3.0 Host Tuning Feature, discussed in SD3.0 Host Tuning Feature on page 229.

The DMA adapter is used to interface to the DMA interconnect of the FX3S device. All DMA data through the interface is assembled at the core and then forwarded to the DMA adapter to transfer over the system interconnect. The channel to access the DMA adapter is the thread. A thread controller converts the simple request-data interface from the core to the required transactions on the DMA adapter. The DMA data enters the DMA adapter through two distinct pipes, named Thread 0 and Thread 1 in Figure 9-1. These two pipes are required to allow two simultaneous data streams from/to the two storage

devices. The cores talk to the external world through a 1-, 4-, or 8-bit data bus.

The SIB has eight sockets (sockets 0-7) associated with it. The storage port driver in the FX3 SDK reserves two of these sockets (socket 6 and 7) for internal operations like card initialization. The remaining sockets can be used for data transfers with the storage devices.

9.3 Storage Interface (S-Port)

The two ports (S0 and S1) comprising the FX3S storage interface are configured as follows.

Storage port 0 (S0 port) can be configured as:

- MMC-only interface
- SD/SDIO and GPIO interface
- GPIO-only interface

Storage port 1 (S1 port) can be configured as:

- MMC-only interface
- SD/SDIO and GPIO interface
- SD/SDIO and UART interface
- SD/SDIO and SPI interface
- SD/SDIO and I2S interface
- GPIO-only interface
- GPIO + UART + I2S interface
- UART + SPI + I2S interface.

Figure 9-2 shows the dual-SD/MMC/SDIO configuration, and Table 9-1 shows the S-port mapping in all configurations.

Figure 9-2. Dual-SD/SDIO/MMC Configuration

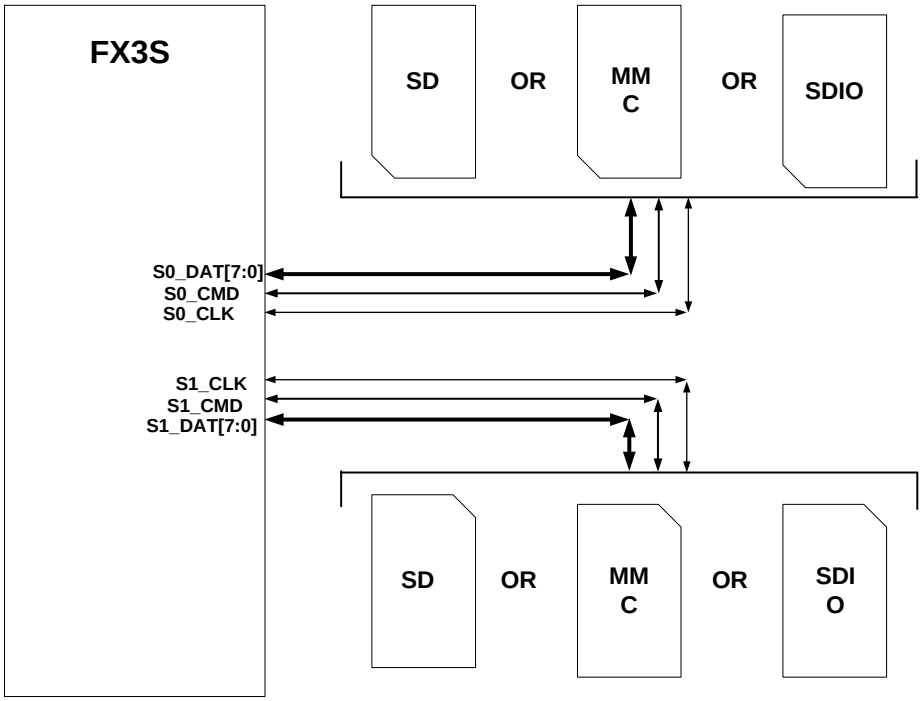


Table 9-1. S-Port Mapping in All Configurations

FX3S Pin Description									
Name	Power Domain	S0-Port							
		8b MMC		SD+GPIO			GPIO		
GPIO[33]	VIO2	S0_SD0		S0_SD0			GPIO		
GPIO[34]	VIO2	S0_SD1		S0_SD1			GPIO		
GPIO[35]	VIO2	S0_SD2		S0_SD2			GPIO		
GPIO[36]	VIO2	S0_SD3		S0_SD3			GPIO		
GPIO[37]	VIO2	S0_SD4		GPIO			GPIO		
GPIO[38]	VIO2	S0_SD5		GPIO			GPIO		
GPIO[39]	VIO2	S0_SD6		GPIO			GPIO		
GPIO[40]	VIO2	S0_SD7		GPIO			GPIO		
GPIO[41]	VIO2	S0_CMD		S0_CMD			GPIO		
GPIO[42]	VIO2	S0_CLK		S0_CLK			GPIO		
GPIO[43]	VIO2	S0_WP		S0_WP			GPIO		
GPIO[44]	VIO2	S0S1_INS		S0S1_INS			GPIO		
GPIO[45]	VIO2	MMC0_RST_OUT		GPIO			GPIO		
		S1-Port							
		8b MMC	SD+UART	SD+SPI	SD+GPIO	GPIO	GPIO+UART+ I2S	SD+I2S	UART+SPI+ I2S
GPIO[46]	VIO3	S1_SD0	S1_SD0	S1_SD0	S1_SD0	GPIO	GPIO	S1_SD0	UART_RTS
GPIO[47]	VIO3	S1_SD1	S1_SD1	S1_SD1	S1_SD1	GPIO	GPIO	S1_SD1	UART_CTS
GPIO[48]	VIO3	S1_SD2	S1_SD2	S1_SD2	S1_SD2	GPIO	GPIO	S1_SD2	UART_TX
GPIO[49]	VIO3	S1_SD3	S1_SD3	S1_SD3	S1_SD3	GPIO	GPIO	S1_SD3	UART_RX
GPIO[50]	VIO3	S1_CMD	S1_CMD	S1_CMD	S1_CMD	GPIO	I2S_CLK	S1_CMD	I2S_CLK
GPIO[51]	VIO3	S1_CLK	S1_CLK	S1_CLK	S1_CLK	GPIO	I2S_SD	S1_CLK	I2S_SD
GPIO[52]	VIO3	S1_WP	S1_WP	S1_WP	S1_WP	GPIO	I2S_WS	S1_WP	I2S_WS
GPIO[53]	VIO4	S1_SD4	UART_RTS	SPI_SCK	GPIO	GPIO	UART_RTS	GPIO	SPI_SCK
GPIO[54]	VIO4	S1_SD5	UART_CTS	SPI_SSN	GPIO	GPIO	UART_CTS	I2S_CLK	SPI_SSN
GPIO[55]	VIO4	S1_SD6	UART_TX	SPI_MISO	GPIO	GPIO	UART_TX	I2S_SD	SPI_MISO
GPIO[56]	VIO4	S1_SD7	UART_RX	SPI_MOSI	GPIO	GPIO	UART_RX	I2S_WS	SPI_MOSI
GPIO[57]	VIO4	MMC1_RST_OUT	GPIO	GPIO	GPIO	GPIO	I2S_MCLK	I2S_MCLK	I2S_MCLK

The S-port interface includes three power domains: VIO2, VIO3, and VIO4. For the system design, the power domain connection depends on the S-port configuration. You must connect the power domains of all signals of a particular interface to the same power source. For example, if the S-port 1 is configured to "8-bit MMC configuration," then VIO2, VIO3, and VIO4 must connect to the same voltage level (for example, 3.3 V).

The storage interface signals can be driven at 3.3 V or 1.8 V (low-voltage operation for UHS mode of operation) based on the voltage provided on the VIO2 or VIO3 (VIO4) input.

The storage ports can be configured with an 8-bit-wide or 4-bit-wide data bus and will support interface clock frequencies up to 104-MHz SDR or 52-MHz DDR.

The storage ports do not support functioning in the legacy SPI mode.

The storage controller block on the FX3S device supports sending any command with custom parameters and receiving arbitrary lengths of device response. The data interface is connected to the DMA fabric, and all data transfers are performed through DMA sockets. Refer to chapter 5 for details on DMA and DMA sockets.

The storage controller can be configured to transfer one or more blocks of data with arbitrary block lengths. However, it is expected that the data block length will be a multiple of 8 bytes and greater than or equal to 16 bytes. The FAST_IO (CMD39) command should be used to transfer small amounts data from/to SDIO devices.

The interface clock frequency is divided down from the master system clock (384 MHz or 416 MHz) through a set of dividers. The block supports clock frequencies ranging from 400-kHz to 104-MHz SDR (or 52-MHz DDR).

The storage controller supports stopping the interface clock to reduce power consumption when the interface is not in use. An auto stop clock feature is supported to prevent data overflow during read operations. The clock automatically stops when the internal buffer is full and restarts when a buffer is made available.

The storage controller supports card insertion and removal detection through one of two mechanisms:

- Voltage change on the DAT[3] pin
- Voltage change on a dedicated GPIO pin connected to a microswitch on the card socket

GPIO pins are used for resetting the eMMC devices and for checking the write protect status of the storage devices. These pins operate under firmware control and do not directly affect the storage port operation.

The storage controller supports SDIO-specific features such as SDIO interrupt detection and the SDIO read-wait and suspend-resume features, as specified in the SDIO specification version 2.00.

9.4 SD/ MMC/ SDIO Interface

9.4.1 SD/MMC Interface Overview

The SD bus includes the following signals:

- CLK: Host to card clock signal
- CMD: Bidirectional command/response signal
- DAT0-DAT3: Four bidirectional data signals
- VDD, VSS1, VSS2: Power and ground signals

The MMC bus includes these signals:

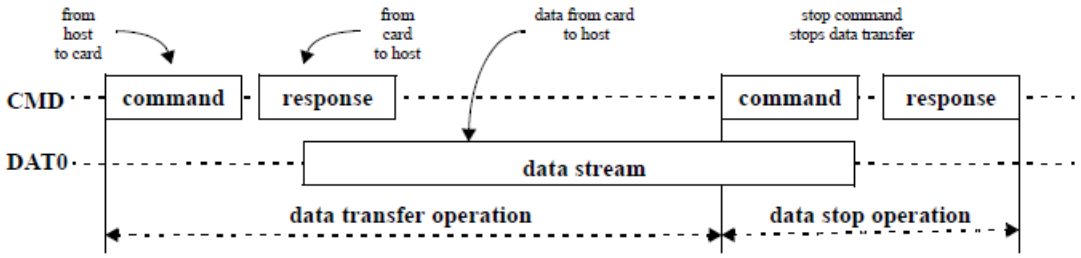
- CLK: Host to card clock signal
- CMD: Bidirectional command/response signal
- DAT0-DAT7: Eight bidirectional data signals
- VDD, VSS1, VSS2: Power and ground signals

The SD/MMC protocol is based on command and data bit streams that are initiated by a start bit and terminated by a stop bit. Additionally, the SD/MMC controller provides a reference clock. Each message/operation is represented by one of the following tokens:

- Command: A token transmitted serially on the CMD pin that starts an operation. A command is sent from the host to a card
- Response: A token from the card transmitted serially on the CMD pin in response to certain commands, from the card to the host.
- Data: Data can be transferred via the data lines from the card to the host or vice versa. The number of data lines used for the data transfer can be one (DAT0) or four (DAT0-DAT3) with the SD protocol and one (DAT0), four (DAT0-DAT3), or eight (DAT0-DAT7) with the MMC protocol.

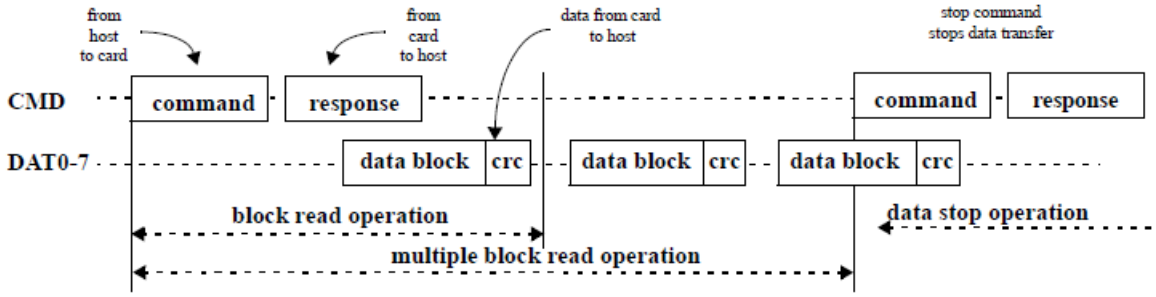
Figure 9-3, Figure 9-4, and Figure 9-5 show the possible bus operations in a normal working case. Please note that the figures are for illustrative purposes and use read operations in the MMC card as an example. Similar operations apply to the SD protocol as well, with a change in data bus width.

Figure 9-3. Sequential Read Operation (Using 1-bit Data Bus)*



* Source: eMMC Specification 4.3

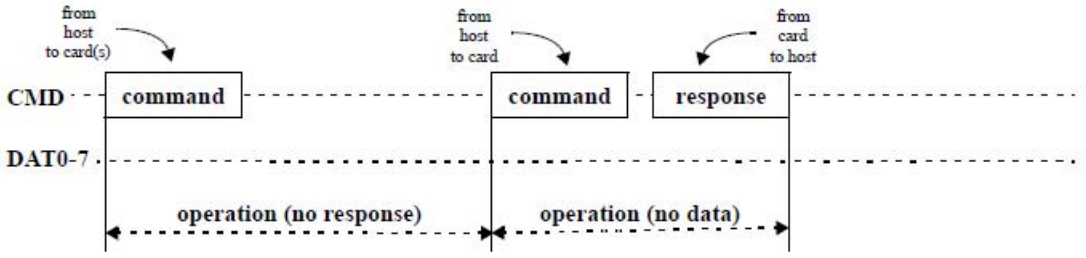
Figure 9-4. (Multiple) Block Read Operation (Using 8-bit MMC Data Bus)*



* Source: eMMC Specification 4.3

*Note: Bus width for the block read operation in the SD card would be 4 bits using the DAT0-3 lines.

Figure 9-5. "No Response" and "No Data" Operations*



* Source: eMMC Specification 4.3

For more details, refer to the SD/MMC specification.

9.4.2 SDIO Interface Overview

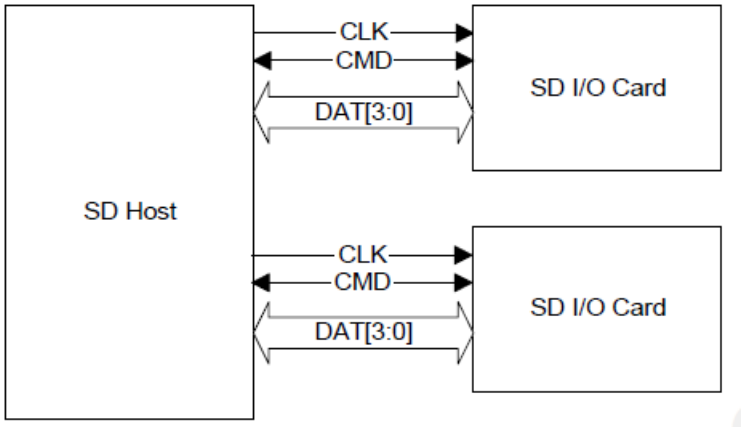
A Secure Digital Input and Output (SDIO) card is an extension of the SD specification to cover I/O functions. The SD standard offers great flexibility, including the ability to use the SD slot for more than memory cards. The SDIO card is an interface that extends the functionality of devices by using a standard SD card slot to give devices that use these new capabilities. A partial list of new capabilities includes the following:

- GPS
- Camera
- Wi-Fi
- FM radio
- Ethernet
- Barcode readers
- Bluetooth®

The SDIO and SD interfaces are mechanically and electrically identical. Thus, the SDIO bus includes the following signals:

- CLK: Host to card clock signal
- CMD: Bidirectional command/response signal
- DAT0-DAT3: Four bidirectional data signals
- VDD, VSS1, VSS2: Power and ground signals

Figure 9-6. Signal Connection to Two 4-Bit SDIO Cards*



* Source: SDIO Specification, version 2.0

The command/response protocol for SDIO is the same as that for the SD protocol described in 9.4.1 SD/MMC Interface Overview on page 207. For more details, refer to the SDIO specification.

9.5 FX3S S-Port Operations Overview

Most of the S-port operations including initialization and other housekeeping tasks are performed by the storage driver, which is provided with the FX3 SDK. The FX3 SDK provides the following:

1. Storage driver that identifies and initializes the storage peripherals connected to FX3S, as well as interrupt services associated with the storage interface
2. A set of APIs (storage API library) that allows users to query peripheral properties and perform data transfers to/from these peripherals

The storage driver uses a dedicated RTOS thread and handles all events arising due to the SIB controller interrupts. The storage driver identifies the type of storage device connected on each of the enabled storage ports and initializes it with the appropriate command sequence.

The storage API provides functions to query the storage device properties like device type, memory capacity, number of SDIO functions, and so on. APIs are provided to perform read/write transfers to SD/MMC memory devices as well as to SDIO cards.

The storage API also provides functions to send individual commands to the storage device and obtain the corresponding responses. The device initialization will be performed by the storage driver in this case as well, and the command mode can be used only after device initialization. The storage APIs are intended only for applications running on the FX3S device.

9.5.1 S-port Initialization and Configuration

9.5.1.1 Configuring the FX3S I/O Matrix

The GCTL_IOMATRIX register must be configured to enable the appropriate mode for the S0 and S1 ports. This is done using the API CyU3PDeviceConfigureIOMatrix. When using eMMC devices, the s0Mode/s1Mode should be set to CY_U3P_SPORT_8BIT to obtain the optimal performance. GPIOs (unused pins that are not configured for any other peripheral interfaces) to be used as voltage switching GPIOs or reset GPIOs also need to be configured for use as simple GPIOs. Refer to 4.1.1 I/O Matrix Configuration on page 53 for details on IO matrix configuration.

```
/* Configure the IO matrix for the device.
     * S0 port is enabled in 8 bit mode.
     * S1 port is enabled in 4 bit mode.
     * UART is enabled on remaining pins of the S1 port.
     */
    io_cfg.isDQ32Bit        = CyFalse;
    io_cfg.s0Mode           = CY_U3P_SPORT_8BIT;
    io_cfg.s1Mode           = CY_U3P_SPORT_4BIT;
    io_cfg.gpioSimpleEn[0]  = 0;
    io_cfg.gpioSimpleEn[1]  = 0x02002000;  /*IOs 45 and 57 are chosen as GPIO
   for voltage switching GPIOs */
    io_cfg.gpioComplexEn[0] = 0;
    io_cfg.gpioComplexEn[1] = 0;
    io_cfg.useUart          = CyTrue;
    io_cfg.useI2C           = CyFalse;
    io_cfg.useI2S           = CyFalse;
    io_cfg.useSpi           = CyFalse;
    io_cfg.lppMode          = CY_U3P_IO_MATRIX_LPP_UART_ONLY;
    status = CyU3PDeviceConfigureIOMatrix (&io_cfg);
    if (status != CY_U3P_SUCCESS)
    {
        goto handle_fatal_error;
    }
```

9.5.1.2 Setting S-Port Interface Parameters

Interface parameters such as low-voltage operation, DDR clocking support, card detection and write protection support need to be set independently for both the storage ports using the API CyU3PSibSetIntfParams.

```
intfParams.resetGpio = 0xFF; /* No GPIO control on SD/MMC power. */
    intfParams.rstActHigh = CyTrue;  /* Don't care as no GPIO is selected. */
    intfParams.cardDetType = CY_U3P_SIB_DETECT_DAT_3;  /* Card detect based
                           on SD_DAT[3]. */
    intfParams.writeProtEnable = CyTrue; /* Write protect handling enabled.*/
    intfParams.lowVoltage = CyTrue;     /* Low voltage operation enabled. */
    intfParams.voltageSwGpio = 45;     /* Use GPIO_45 for voltage switch on
```

```
                S0 port. */
    intfParams.lvGpioState = CyFalse;  /* Driving GPIO low selects 1.8 V on
                SxVDDQ. */
    intfParams.useDdr = CyTrue;        /* DDR clocking enabled. */
    intfParams.maxFreq = CY_U3P_SIB_FREQ_104MHZ; /* No S port clock
                        limitation. */
    intfParams.cardInitDelay = 0; /* No delay required between SD card    insertion before
initialization. */

    status = CyU3PSibSetIntfParams (0, &intfParams);
    if (status == CY_U3P_SUCCESS)
    {
        intfParams.voltageSwGpio = 57;
        /* Use GPIO_57 for voltage switch on S1 port. */
        status = CyU3PSibSetIntfParams (1, &intfParams);
    }

    if (status != CY_U3P_SUCCESS)
    {
        CyU3PDebugPrint (4, "Set SIB interface parameters failed,
code=%d\r\n", status);
        CyFxAppErrorHandler (status);
    }
    status = CyU3PSibStart ();
    if (status != CY_U3P_SUCCESS)
    {
        CyU3PDebugPrint (4, "SIB start failed, code=%d\r\n", status);
        CyFxAppErrorHandler (status);
    }
```

The structure passed as a parameter to CyU3PSibSetIntfParams defines the configuration parameters for the storage ports. These parameter values are used internally by the storage driver to configure the GPIOs (like resetGPIO, voltageswGPIO, and so on, if enabled) appropriately. All parameter values, except for the "cardDetType," are used internally by the storage driver only and do not have a register-level impact. cardDetType enables the appropriate interrupt for card detection. Refer to Card Insertion and Removal Detection Mechanism on page 226 for more details on card detection.

9.5.1.3 Starting the Storage Driver

The CyU3PSibStart API is called to start the storage driver and initialize any storage devices that are connected on the two storage ports. Depending on the type of device (SD/MMC/SDIO), the device is initialized appropriately by the driver. The storage device types supported by FX3S are as follows.

```
/** \brief List the Storage Device types supported by FX3S.

    **Description**\n
    This enumeration lists the various storage device types supported by the
    storage module in FX3S firmware.
*/
typedef enum CyU3PSibDevType
{
    CY_U3P_SIB_DEV_NONE = 0,            /**< No device */
    CY_U3P_SIB_DEV_MMC,                 /**< MMC/eMMC device */
    CY_U3P_SIB_DEV_SD,                  /**< SD Memory card */
    CY_U3P_SIB_DEV_SDIO,                /**< SDIO IO only Device.*/
    CY_U3P_SIB_DEV_SDIO_COMBO           /**< SDIO Combo Device. Data transfers to the combo
card are not supported. */
} CyU3PSibDevType;
```

9.5.1.4 Setting the S-Port Clock

The S0/S1 interface clock is enabled or disabled by the bit SDMMC_CLK_DIS in the register SDMMC_CS (SD/MMC/SDIO card command and status register). For bit definitions, refer to SDMMC_CS register on page 656.

The clock values are set using the register GCTL_SIB0_CORE_CLK and GCTL_SIB1_CORE_CLK (SIB Port 0/1 core clock configuration register). GCTL_SIBx_CORE_CLK.DIV determines the clock divider value for dividing the PLL system clock. GCTL_SIBx_CORE_CLK.SRC selects the clock source, and GCTL_SIBx_CORE_CLK.EN enables it.

The clock choice for the divisor is user-configurable. For example, the following frequencies can be configured through these registers:

- 400 kHz: For the SD/MMC card initialization
- 20 MHz: For a card with 0- to 20-MHz frequency
- 24 MHz: For a card with 0- to 26-MHz frequency
- 48 MHz: For a card with 0- to 52-MHz frequency (48-MHz frequency on SD_CLK is supported when the clock input to FX3S is 19.2 MHz or 38.4 MHz.)
- 52 MHz: For a card with 0- to 52-MHz frequency (52-MHz frequency on SD_CLK is supported when the clock input to FX3S is 26 MHz or 52 MHz)
- 100 MHz: For a card with 0- to 100-MHz frequency

If the DDR mode is selected, data is clocked on both the rising and falling edge of the SD clock. DDR clocks run up to 52 MHz.

9.5.1.4.1 Powering On the SIB

The SIB block is power cycled using the SIB_POWER.RESETN bit and then polled for the SIB_POWER.ACTIVE bit, which remains deasserted until initialization is complete. SIB_POWER.ACTIVE = 1 indicates block is initialized and ready for operation. For bit definitions, refer to SIB_POWER register on page 640.

9.5.1.4.2 Initializing SIB Sockets

Initially, all sockets are disabled, and socket interrupts are cleared using the registers SCK_STATUS, SCK_INTR and SCK_INTR_MASK. Refer to 5.5.5 Sockets on page 68 for more details on sockets.

Later, the active SIB socket number for read/write is set using SDMMC_CS.SOCKET before initiating reads/writes from/to the SD/MMC/SDIO device.

9.5.1.4.3 S-Port I/O Drive Strength

The S-port I/O drive strength is programmable like any other I/O pin as discussed in 4.1.2 I/O Drive Strength on page 55.

9.5.1.5 Sending SD/MMC/SDIO Commands

The SD/SDIO/MMC command to be sent on the command bus is specified through the register SDMMC_CMD_IDX. The cmd field specifies the command index that is to be included in the command sent. When the command is sent, the hardware sends out start and transmission bits followed by (SDMMC_CMD_RESP_FMT.CMDFRMT+1 bits) a 7-bit CRC code and an end bit. The (SDMMC_CMD_RESP_FMT.CMDFRMT+1) bits include as many bits as necessary in the following order: command index (6 bits), argument (starting from bit 31 of SDMMC_CMD_ARG0), SDMMC_CMD_ARG1. The SDMMC_CMD_ARG0 register holds the 32-bit SD/MMC/SDIO command argument. Any additional argument bits for the expanded command may be placed in the SDMMC_CMD_ARG1 register. There is no need for the software to specify start, transmission and end bits-they are to be generated by the hardware.

For bit definitions, refer to SDMMC_CMD_IDX register on page 641, SDMMC_CMD_RESP_FMT register on page 650, SDMMC_CMD_ARG0 register on page 642 and SDMMC_CMD_ARG1 register on page 643.

After writing to these registers, the command transmission is initiated by setting the SDMMC_CS. SNDCMD bit and then polled for the SDMMC_INTR.CMDSENT bit that indicates command send completion. Refer to 9.5.1.6 Handling SIB Events on page 214 for the SDMMC_INTR register field descriptions.

The SDMMC_INTR.RCVDRES bit is polled to get the response from the card. The first 8 bits of response are captured in the RESP_IDX register. These eight bits include the start, transmission, and command index fields. The response data received from the card on the response bus, including crc7 and end bits, is placed in registers SDMMC_RESP_REG.

For bit definitions, refer to SDMMC_RESP_IDX register on page 644 and SDMMC_RESP_REG0 register on page 645.

Command/response timing is specified through the registers, SDMMC_NCC_NWR, SDMMC_NAC, and SDMMC_NCR.

For bit definitions, refer to SDMMC_NCC_NWR register on page 665, SDMMC_NAC register on page 666, SDMMC_NCR register on page 664.

A block size of 512 (0x0200) is set as the default using the register SDMMC_BLOCKLEN.

Following successful initialization, the SIB mode (SD/ MMC/ SDIO) is set and SIB is configured for the data bus width and clock speed using the SDMMC_MODE_CFG register. For bit definitions, refer to SDMMC_MODE_CFG register on page 653.

The devices detected on the storage ports can be queried using the API CyU3PSibQueryDevice. If the device on the storage port has more than one partition, then the API CyU3PSibQueryUnit can be used to access the storage partition (LUN) information for the specified port and unit number. This applies only to SD and eMMC devices.

```
for (i = 0; i < CY_FX_SIB_PORTS; i++)
{
    unitsFound = 0;

    /* Initialize LUN data structures with default values. */
    for (j = 0; j < CY_FX_SIB_PARTITIONS; j++)
    {
        glLunState[i * CY_FX_SIB_PARTITIONS + j]     = CyFalse;
        glLunUnit[i * CY_FX_SIB_PARTITIONS + j]      = 10;    /* Invalid value. */
        glLunWriteable[i * CY_FX_SIB_PARTITIONS + j] = CyTrue;
        glLunBlkSize[i * CY_FX_SIB_PARTITIONS + j]   = 0;
        glLunNumBlks[i * CY_FX_SIB_PARTITIONS + j]   = 0;
    }

    /* Query each of the storage ports to identify whether a device has been
      detected. */
    status = CyU3PSibQueryDevice (i, &glDevInfo[i]);
    if (status == CY_U3P_SUCCESS)
    {
        CyU3PDebugPrint (8, "Found a device on port %d\r\n", i);
        CyU3PDebugPrint (6, "\tType=%d, numBlks=%d, eraseSize=%d, clkRate=%d\r\n",
                glDevInfo[i].cardType, glDevInfo[i].numBlks, glDevInfo[i].eraseSize,
                glDevInfo[i].clkRate);
        CyU3PDebugPrint (6, "\tblkLen=%d removable=%d, writeable=%d, locked=%d\r\n",
                glDevInfo[i].blkLen, glDevInfo[i].removable, glDevInfo[i].writeable,
                glDevInfo[i].locked);
        CyU3PDebugPrint (6, "\tddrMode=%d, opVoltage=%d, busWidth=%d,
                             numUnits=%d\r\n", glDevInfo[i].ddrMode,
                             glDevInfo[i].opVoltage, glDevInfo[i].busWidth,
                             glDevInfo[i].numUnits);
        CyU3PDebugPrint (6, "\tcardCmdClass=%d\r\n", glDevInfo[i].ccc);

        /* Run through each of the units present and check how many data partitions
           are present. We skip all boot partitions in this application.
        */
        for (j = 0; j < glDevInfo[i].numUnits; j++)
        {
            status = CyU3PSibQueryUnit (i, j, &unitInfo);
            if (status != CY_U3P_SUCCESS)
```

```
            {
                CyU3PDebugPrint (2, "Error: Failed to query partition %d on port
                %d\r\n", j, i);
                break;
            }

            CyU3PDebugPrint (6, "Dev %d, Unit %d: location=%d numBlocks=%d\r\n",
                    i, j, unitInfo.location, unitInfo.numBlocks);
            if (unitInfo.location == CY_U3P_SIB_LOCATION_USER)
            {
                CyU3PDebugPrint (4, "Dev %d: Found user partition\r\n", i);
                unitsFound++;
            }
        }

    }
```

If an existing partition needs to be removed for the specified storage port, this can be done using the API CyU3PSibRemovePartitions. This reunifies the complete storage available on a device as a single data partition (volume). CyU3PSibPartitionStorage can be used to partition the storage device into multiple logical units that can be separately accessed.

```
        if ((unitsFound < CY_FX_SIB_PARTITIONS) && (glDevInfo[i].writeable))
        {
            /* Make sure any existing partitioning is removed. */
            CyU3PSibRemovePartitions (i);

            /* Makes sure that each storage device is partitioned into the
               required number of units. */
            for (j = 0; j < CY_FX_SIB_PARTITIONS; j++)
            {
                partsize[j] = (glDevInfo[i].numBlks / CY_FX_SIB_PARTITIONS);
                parttype[j] = CY_U3P_SIB_LUN_DATA;
            }

            CyU3PDebugPrint (4, "Port %d: Creating partitions as required\r\n",i);
            status = CyU3PSibPartitionStorage (i, CY_FX_SIB_PARTITIONS, partsize,
                     parttype);
            if (status != CY_U3P_SUCCESS)
            {
                CyU3PDebugPrint (2, "Error: Failed to partition storage device %d,
                                     status=%d\r\n", i, status);
                continue;
            }
        }
```

Note: APIs CyU3PSibPartitionStorage and CyU3PSibRemovePartitions are used to manage virtual partitions created by the firmware.

9.5.1.6 Handling SIB Events

The SIB events are based on the bits of the SD/MMC/SDIO interrupt request register (SDMMC_INTR). These bits are set to 1 whenever a bit in SDMMC_STATUS changes from 0 to 1.

For bit definitions, refer to SDMMC_STATUS register on page 658, SDMMC_INTR register on page 660, and SDMMC_INTR_MASK register on page 662.

CyU3PSibRegisterCbk can be used to register a callback function that will be called to provide a notification for storage-related events such as card insertion/removal and data transfer completion; as well as for the status of the storage read/write operations.

```
/* Register a callback for SIB events. */
CyU3PSibRegisterCbk (CyFxMscApplnSibCB);

void
CyFxMscApplnSibCB (
        uint8_t             portId,
        CyU3PSibEventType   evt,
        CyU3PReturnStatus_t status)
{
    CyU3PDmaSocketConfig_t sockConf;

    if (evt == CY_U3P_SIB_EVENT_XFER_CPLT)
    {
        if (status != CY_U3P_SUCCESS)
        {
            glMscCmdStatus     = 1;
            glSensePtr[portId] = CY_FX_MSC_SENSE_CRC_ERROR;

            /* Transfer has failed. Reset the DMA channel. */
            if (glCmdDirection)
            {
                CyU3PDmaSocketGetConfig ((uint16_t)(CY_U3P_UIB_SOCKET_CONS_0 |
CY_FX_MSC_EP_BULK_IN_SOCKET),
                        &sockConf);
                glMscResidue -= sockConf.xferCount;
                CyU3PDmaChannelReset (&glChHandleMscIn);
            }
            else
            {
                CyU3PDmaSocketGetConfig ((uint16_t)(CY_U3P_UIB_SOCKET_PROD_0 +
CY_FX_MSC_EP_BULK_OUT_SOCKET),
                        &sockConf);
                glMscResidue -= sockConf.xferCount;
                CyU3PDmaChannelReset (&glChHandleMscOut);
            }
        }
        else
        {
            glMscCmdStatus = 0;
            glMscResidue   = 0;
            glSensePtr[portId] = CY_FX_MSC_SENSE_OK;
        }

        CyU3PEventSet (&glMscAppEvent, CY_FX_MSC_SIBCB_EVENT_FLAG, CYU3P_EVENT_OR);
    }

    if (evt == CY_U3P_SIB_EVENT_INSERT)
    {
        uint8_t i = 0;

        CyU3PDebugPrint (2, "Insert event on port %d\r\n", portId);
        CyFxMscApplnQueryDevStatus ();
        for (i = 0; i < CY_FX_SIB_PARTITIONS; i++)
        {
```

```
            glSensePtr[portId * CY_FX_SIB_PARTITIONS + i] = CY_FX_MSC_SENSE_MEDIA_CHANGED;
        }
    }

    if (evt == CY_U3P_SIB_EVENT_REMOVE)
    {
        uint8_t i = 0, lun = 0;

        CyU3PDebugPrint (2, "Remove event on port %d\r\n", portId);
        for (i = 0; i < CY_FX_SIB_PARTITIONS; i++)
        {
            lun = portId * CY_FX_SIB_PARTITIONS + i;

            glLunState[lun]     = CyFalse;
            glLunUnit[lun]      = 10;
            glLunBlkSize[lun]   = 0;
            glLunNumBlks[lun]   = 0;
            glLunWriteable[lun] = CyTrue;
            glSensePtr[lun]     = CY_FX_MSC_SENSE_MEDIA_CHANGED;
        }
    }

    if ((evt == CY_U3P_SIB_EVENT_DATA_ERROR) || (evt == CY_U3P_SIB_EVENT_ABORT))
    {
        /* Transfer has failed. Reset the DMA channel. */
        if (glCmdDirection)
        {
            CyU3PDmaChannelReset ((CyU3PDmaChannel *) &glChHandleMscOut);
        }
        else
        {
            CyU3PDmaChannelReset ((CyU3PDmaChannel *) &glChHandleMscIn);
        }

        /* Make sure the request is aborted and that the controller is reset. */
        CyU3PSibAbortRequest (portId);
    }
}
```

9.5.2 Reads and Writes to SD/ MMC Using DMA Transfers

A DMA channel (out) is created to transfer data to the SD/ MMC device connected to any of the S-ports, and another DMA channel (in) is created to read data from the SD/MMC device. Refer to FX3 DMA Subsystem chapter on page 61 for more details on the DMA channels and sockets. Any of the six available S-port sockets can be used for these channels. The socket is defined as follows inside the FX3 SDK library:

```
    CY_U3P_SIB_SOCKET_0 = 0x0200,               /**< S-port socket number 0. */
    CY_U3P_SIB_SOCKET_1,                        /**< S-port socket number 1. */
    CY_U3P_SIB_SOCKET_2,                        /**< S-port socket number 2. */
    CY_U3P_SIB_SOCKET_3,                        /**< S-port socket number 3. */
    CY_U3P_SIB_SOCKET_4,                        /**< S-port socket number 4. */
    CY_U3P_SIB_SOCKET_5,                        /**< S-port socket number 5. */
```

The following is the code for performing reads and writes to/from the SD/MMC device.

```
  /* Initiating Write to SD/ MMC card */
```

```
 status = CyU3PDmaChannelSetXfer (&glChHandleMscOut, (numBlks *
                                  CY_FX_SIB_MAX_BLOCK_SIZE));
 if (status == CY_U3P_SUCCESS)
 {
     status = CyU3PSibReadWriteRequest (CY_FX_SIB_WRITE, ((lun >= CY_FX_SIB_PARTITIONS)
                                    ? 1 : 0), glLunUnit[lun], numBlks, startAddr, 0);
     if (status != CY_U3P_SUCCESS)
     {
          /* Abort the DMA Channel */
          CyU3PDmaChannelReset (&glChHandleMscIn);
     }
  }

 /* Initiating Read from SD/ MMC card */
status = CyU3PDmaChannelSetXfer (&glChHandleMscIn, (numBlks *
                                   CY_FX_SIB_MAX_BLOCK_SIZE));
if (status == CY_U3P_SUCCESS)
{
   status = CyU3PSibReadWriteRequest (CY_FX_SIB_READ, ((lun >= CY_FX_SIB_PARTITIONS) ?
                                     1 : 0), glLunUnit[lun], numBlks, startAddr, 1);
   if (status != CY_U3P_SUCCESS)
   {
        /* Abort the DMA Channel */
        CyU3PDmaChannelReset (&glChHandleMscIn);
   }
}
```

CyU3PSibReadWriteRequest initiates a read/write data request. This function is used to initiate a read/write request to a storage device. The open-ended read/write commands (CMD18 and CMD25) are used in all cases.

Inside this API, the block length and number of blocks, calculated from the parameters passed to it, are set in the appropriate registers: SDMMC_BLOCKLEN and SDMMC_BLOCK_COUNT.

For bit definitions, refer to SDMMC_BLOCK_COUNT register on page 651.

The active SIB socket number for read/write is set using SDMMC_CS.SOCKET before initiating reads/writes from/to the SD/MMC device. The DAT0 pin state can be monitored to learn the storage device busy state by polling for the bit SDMMC_SATUS.DAT0_STAT. Before starting the command transmission involving the data phase, the SDMMC_STATUS.DATA_SM_BUSY bit is polled to learn if the data state machine is busy. If so, the SIB is reset by setting the SDMMC_CS.RSTCONT bit and then polled for it to become 0. Hardware writes 0 when reset is complete.

Before initiating any command transmission involving data through DAT[0:3] for SD or DAT[0:7] for MMC, the DAT3_CHANGE interrupt is disabled by setting the SDMMC_INTR_MASK.DAT3_CHANGE bit. The interrupt is enabled again after command completion. Then command transmission is initiated and completed in the same manner as described in 9.5.1.5 Sending SD/MMC/SDIO Commands on page 212.

In a read operation (MULTI BLOCK read command CMD18), the command transmission is initiated by setting the bits SDMMC_CS. SNDCMD and SDMMC_CS. RDDCARD.

The CY_U3P_SIB_EVENT_XFER_CPLT event is sent to the caller through the register storage callback function when the specified amount of data has been completely transferred. This event is triggered based on the bits SDMMC_INTR.BLOCK_COMP and SDMMC_INTR.BLOCKS_RECEIVED.

The socket number to be passed as a parameter to the function CyU3PSibReadWriteRequest is the offset with respect to CY_U3P_SIB_SOCKET_0. For example, if CY_U3P_SIB_SOCKET_1 is used to create the DMA channel, then the socket number to be passed in CyU3PSibReadWriteRequest is "1". This function internally initiates the DMA channel for transferring the specified amount of data. It is therefore expected that the DMA channel be left in the idle state (CY_U3P_DMA_CONFIGURED) when calling this API.

CyU3PDmaChannelSetXfer sets up a one-to-one DMA channel for data transfer. The sockets corresponding to a DMA channel are left disabled when the channel is created, so that transfers are started only when desired by the user. This function enables a DMA channel to transfer a specified amount of data before suspending again. An infinite transfer can be started by specifying a transfer size of 0. This function should be called only when the channel is in the CY_U3P_DMA_CONFIGURED state. Refer to chapter 5 for more details on DMA.

9.5.2.1 Sending Vendor Commands to SD/ MMC

The CyU3PSibVendorAccess function is used to send general SD/MMC commands to storage device/cards attached to the FX3S device and receive the corresponding response. Custom commands can be implemented using this function. Command transmission is initiated and completed in the same manner as described in 9.5.1.5 Sending SD/MMC/SDIO Commands on page 212.

9.5.2.2 Setting the Granularity of Write Operations

CyU3PSibSetWriteCommitSize sets the sector granularity at which write operations are committed to the SD/MMC storage. Committing write data to SD/MMC storage in small chunks can lead to poor write transfer performance. The write performance to these devices can be improved by combining write operations with sequential addresses and performing a single commit (STOP_TRANSMISSION) operation.

```
/* Commit data in 8MB chunks: 16k sectors */
CyU3PSibSetWriteCommitSize (i, 16384);
```

By default, this value is set to 1, meaning that a write of one sector or more is immediately committed to the storage device.

9.5.2.3 Checking Card Status

The CyU3PSibGetCardStatus function sends the CMD13 (SEND_STATUS) command to the SD/MMC device connected on the specified storage port and provides the card status response that is received. This API can be used to check whether the storage device is accessible and is in the TRAN state, where it can receive new commands.

9.5.2.4 Aborting Ongoing Transaction to S-Port

The SIB is reset by writing to the bit SDMMC_CS. RSTCONT and then polled for it to become 0. Software writes '1' to bring the SIB FSMs to a known state. Hardware writes '0' when reset is complete. The values of the CFG registers are not affected by RSTCONT. Internal queues are flushed by RSTCONT. Then the SDMMC_CS.WRDCARD and SDMMC_CS.RDDCARD bits are selectively set, and the SIB is reset again by setting SDMMC_CS.RSTCONT.

The CyU3PSibAbortRequest API is used to abort any ongoing transactions on the specified port. Prior to calling this function, the corresponding DMA channel should be reset.

```
Void CyFxMscApplnSibCB (
        uint8_t             portId,
        CyU3PSibEventType   evt,
        CyU3PReturnStatus_t status)
{
    if ((evt == CY_U3P_SIB_EVENT_DATA_ERROR) || (evt == CY_U3P_SIB_EVENT_ABORT))
    {
        /* Transfer has failed. Reset the DMA channel. */
        if (glCmdDirection)
        {
            CyU3PDmaChannelReset ((CyU3PDmaChannel *) &glChHandleMscOut);
        }
        else
        {
            CyU3PDmaChannelReset ((CyU3PDmaChannel *) &glChHandleMscIn);
        }
```

```
        /* Make sure the request is aborted and that the controller is reset. */
        CyU3PSibAbortRequest (portId);
    }
}
```

9.5.3 Working with SDIO Cards

9.5.3.1 Configuration and Initialization

The configuration of the S-port and the initialization are very similar to that of the SD card, as explained in 9.5.1 S-port Initialization and Configuration on page 210.

The IOMatrix configuration is done similarly to that for SD cards and interface parameters are set independently for the appropriate S-port. The CyU3PSibStart API calls for starting the storage driver also identify and initialize the SDIO device connected on the two storage ports. The device type, "SDIO IO only" or "combo card," can be learned by calling CyU3PSibQueryDevice.

Note: The storage driver and library APIs in FX3 SDK 1.3.1 have been tested for SDIO I/O only cards. For combo cards, please contact Cypress Support.

CyU3PSdioQueryCard can be used to fetch SDIO card information and card common register information. This function returns a CYU3PSdioCardRegs object that contains card common register information from an initialized SDIO card on the port specified by portId.

9.5.3.2 Reads and Writes from SDIO Card Registers

9.5.3.3 IO_RW_DIRECT Command (CMD52)

The IO_RW_DIRECT command (CMD52) is the simplest means to access a single register within the total register space in any I/O function, including the common I/O area (CIA). This command reads or writes 1 byte using only 1 command/response pair. A common use is to initialize registers or monitor status values for I/O functions. This command is the fastest means to read or write single I/O registers, as it requires only a single command/response pair.

Figure 9-7. IO_RW_DIRECT Command *

S	D	Command Index 110100b	R/W Flag	Function Number	RAW Flag	Stuff	Register Address	Stuff	Write Data or Stuff Bits	CRC7	E
1	1	6	1	3	1	1	17	1	8	7	1

* Source: SDIO Specification, version 2.0

The IO_RW_DIRECT command contains the following fields:

- S: Start bit. Always 0.
- D: Direction. Always1; indicates transfer host to card.
- Command Index: Identifies the IO_RW_DIRECT command with a value of 110100b.
- R/W flag: This bit determines the direction of the I/O operation. If this bit is 0, this command will read data from the SDIO card at the address specified by the Function Number and the Register Address to the host. The data byte is returned in the response, R5. If this bit is set to 1, the command will write the bytes in the Write Data field to the I/O location addressed by the Function Number and Register Address. If the RAW flag is 0, then the data in the register that was written will be read and that value returned in the response.

- RAW flag: The Read after Write flag. If this bit is set to 1 and the R/W flag is set to 1, then the command will read the value of the register after the write. This is useful to allow writing to a control register and reading the status at the same address. If this bit is cleared, the value returned in the R5 response will be the same as the write data in the command. If this bit is set, the data field of the R5 response will contain the value read from the addressed register after the write operation.
- Function Number: The number of the function within the I/O card you wish to read or write. Note that function 0 selects the CIA.
- Register Address: This is the address of the byte of data inside the selected function to read or write. There are 17 bits of address available, so the register is located within the first 128K (131,072) addresses of that function.
- Write Data or Stuff Bits: For a direct write command (R/W=1), this is the byte that is written to the selected address. For a direct read command (R/W=0), this field is not used and is be set to 0.
- CRC7: Seven bits of CRC data.
- E: End bit, always 1.

9.5.3.3.1 CMD52 R5 Response (SD Mode)

If the communication between the card and host is in the 1-bit or 4-bit SD mode, the response will be in a 48-bit response (R5) as shown in Figure 9-8.

Figure 9-8. R5 IO_RW_DIRECT Response (SD Modes) *

S	D	Command Index 110100b	Stuff	Response Flag Bits 7----------0	Read or Write Data	CRC7	E
1	1	6	16	8	8	7	1

* Source: SDIO Specification, version 2.0

- S: Start bit. Always 0.
- D: Direction. 0 indicates transfer card to host (response).
- Command Index: Identifies the IO_RW_DIRECT command with a value of 110100b.
- Stuff: Not used. Will be set to 0.
- Response Flags Bit: Eight bits of flag data indicating the status of the SDIO card.
- Read or Write Data: For an I/O write (R/W=1) with the RAW flag set (RAW=1), this field will contain the value read from the addressed register after the write of the data contained in the command. Note that in this case, the read-back data may not be the same as the data written to the register, depending on the design of the hardware. For an I/O write with the RAW bit=0, the SDIO function will not do a read after the write operation, and the data in this field will be identical to the data byte in the write command. For an I/O read (R/W=0), the actual value read from that I/O location is returned in this field.
- CRC7: Seven bits of CRC data.
- E: End bit, always 1.

For more details on the command response format and bit descriptions, refer to the SDIO specification.

9.5.3.3.2 CyU3PSdioByteReadWrite API

CMD52 command transmission is initiated in the same manner as explained in 9.5.1.5 Sending SD/MMC/SDIO Commands on page 212. The 1-byte response data of the CMD52 read register command is contained in the lower 8 bits of the SDMMC_RESP_REG register.

CyU3PSdioByteReadWrite performs a direct I/O operation (single byte) to an SDIO card using CMD52. This function is used to read or write a single byte of data from/to a register on the SDIO card.

When the readAfterWrite flag is set, the data from the register is read back after the write has been completed. The following code snippet demonstrates the use of this API.

```
/* Initialize the Function */
bitFnNo  = (1 << funcNo);
data    |= bitFnNo;

status = CyU3PSdioByteReadWrite (portId, CY_U3P_SDIO_CIA_FUNCTION, CY_U3P_SDIO_WRITE, 0,
     CY_U3P_SDIO_REG_IO_ENABLE, &data);
if (status != CY_U3P_SUCCESS)
{
   ERROR_PRINT (2, "CMD52: write to IO ENABLE error status = %d\r\n", status);
   return status;
}

/* Wait till IO Ready is set or 100 attempts */
trials = 100;
data   = 0;
while ((!(data & bitFnNo)) && (trials > 0))
{

   status = CyU3PSdioByteReadWrite (portId, CY_U3P_SDIO_CIA_FUNCTION, CY_U3P_SDIO_READ, 0,
        CY_U3P_SDIO_REG_IO_READY, &data);
   if (status != CY_U3P_SUCCESS)
   {
      ERROR_PRINT (6, "CMD52: IO RDY error status = %d\r\n", status);
      return status;
   }

   trials--;
}

if (!(data & bitFnNo))
{
   ERROR_PRINT (6, "IO READY ERROR TIMEOUT \r\n");
   return CY_U3P_ERROR_TIMEOUT;
}
```

9.5.3.3.3 IO_RW_EXTENDED Command (CMD53)

In order to read and write multiple I/O registers with a single command, a new command, IO_RW_EXTENDED, is defined. This command allows reading or writing a large number of I/O registers with a single command. Since this is a data transfer command, it provides the highest possible transfer rate. For this operation, the address of the register is set into the Register Address field. Data is transferred on the DAT[0] or DAT[3:0] lines as defined for SD memory cards.

Figure 9-9. IO_RW_EXTENDED Command *

S	D	Command Index 110101b	R/W Flag	Function Number	Block Mode	OP Code	Register Address	Byte/ Block Count	CRC7	E
1	1	6	1	3	1	1	17	9	7	1

* Source: SDIO Specification, version 2.0

The IO_RW_EXTENDED command contains the following fields:

- S: Start bit. Always 0.
- D: Direction. Always1 indicates transfer host to card.
- Command Index: Identifies the IO_RW_EXTENDED command with a value of 110101b.
- R/W flag: This bit determines the direction of the I/O operation. If this bit is 0, this command reads data from the SDIO card at the address specified by the Function Number and the Register Address to the host; otherwise, it writes the bytes from the DAT[x] lines to the I/O location addressed by the Function Number and Register Address.
- Function Number: The number of the function within the I/O card you wish to read or write. Note that function 0x00 selects the CIA.
- Block Mode (optional): If set to 1, this bit indicates that the read or write operation will be performed on a block basis, rather than the normal byte basis. If this bit is set, the Byte/Block Count value contains the number of bytes/blocks to be read/written. The block size for functions 1-7 is set by writing the block size to the I/O block size register in the FBR. The block size for function 0 is set by writing to the FN0 block size register in the CCCR. Card and host support for the block I/O mode is optional. The block size used when Block Mode = 1 and the maximum byte count per command used when Block Mode = 0 can be read from the CIS in the tuple TPLFE_MAX_BLK_SIZE on a per-function basis (refer to the SDIO specification for more details). Table 9-3 shows the relationship between the value in the command and the actual number of bytes/blocks transferred.
- OP code: Defines the read/write operation as described in Table 9-2.

Table 9-2. OpCode Field Description

Op Code	Command Operation	Description
0	Multibyte R/W to fixed address	Used to read or write multiple bytes of data to/from a single I/O register address. This command is useful when I/O data is transferred using a FIFO inside the I/O card.
1	Multibyte R/W to incremental address	Used to read or write multiple bytes of data to/from an I/O register address that increments by 1 after each operation. This command is used when large amounts of I/O data exist within the I/O card in a RAM-like data buffer.

- Register Address: Start Address of the I/O register to read or write. Range is [0x1FFFF:0].
- Byte/ Block Count: If the command is operating on bytes (Block Mode = 0), this field contains the number of bytes to read or write. A value of 0x000 shall cause 512 bytes to be read or written. If the command is in block mode (Block Mode=1), the Block Count field specifies the number of Data Blocks to be transferred following this command. A value of 0x000 indicates that the count set to infinite.

Table 9-3. Byte Count Values

Count Value	0x000	0x001	0x002	-----	0x1FF
Bytes Transferred	512	1	2	-----	511
Blocks Transferred	∞	1	2	-----	511

- CRC7: Seven bits of CRC data.
- E: End bit, always 1.

9.5.3.3.4 CyU3PSdioExtendedReadWrite API

The CyU3PSdioExtendedReadWrite API performs extended I/O operations (multibyte/multiblock) to/from the SDIO card using CMD53. Arguments needed for CMD53 described in 9.5.3.3.3 IO_RW_EXTENDED Command (CMD53) are passed as parameters.

Inside this API, the block length and number of blocks, calculated from the parameters passed to it, are set in the appropriate registers: SDMMC_BLOCKLEN and SDMMC_BLOCK_COUNT. The logic for the same is as follows.

```
if (blockmode == CY_U3P_SDIO_RW_BLOCK_MODE)
{
        noOfBlocks = transferCount;
        blockSize  = glCardCtxt[portId].fnBlkSize[functionNumber];
```

```
}
else if (blockmode == CY_U3P_SDIO_RW_BYTE_MODE)
{
        noOfBlocks = 1;
        blockSize  = (transferCount != 0) ? transferCount : 512;
}
```

Note: "blockmode" and "transferCount" are parameters passed to CyU3PSdioExtendedReadWrite. Whereas the variable values "noOfBlocks" and "blockSize" are used to set the values of the registers-SDMMC_BLOCKLEN and SDMMC_BLOCK_COUNT-the active SIB socket number for read/write is set using SDMMC_CS.SOCKET before initiating reads/writes from/ to the SDIO device. The DAT0 pin state can be monitored to learn the storage device busy state by polling for the bit SDMMC_SATUS.DAT0_STAT.

In a read opertion, the BLOCKS_RECEIVED and RD_DATA_TIMEOUT interrupts are cleared in the SDMCC_INTR register by writing '1' to the bits. In a write operation, the BLOCK_COMP interrupt in the SDMMC_INTR register is cleared in the same way.

Since CMD53 involves data phase, through DAT[0:3] lines, the DAT3_CHANGE interrupt is disabled by setting SDMMC_INTR_MASK.DAT3_CHANGE. The interrupt is enabled again after command completion. Then command transmission is initiated and completed in the same manner as described in Sending SD/MMC/SDIO Commands on page 212.

In a CMD53 read operation, the command transmission is initiated by setting the bits SDMMC_CS. SNDCMD and SDMMC_CS. RDDCARD, whereas in a CMD53 write operation, the command transmission is initiated by setting the bits SDMMC_CS. SNDCMD and SDMC_CS. WRDCARD.

The data associated with the CMD53 data phase is read/written to/from the DMA channel associated with the socket ID passed in for the operation. This operation should use one of the SIB sockets as a producer (for read) or consumer (for write). Before calling this function, the necessary DMA channels need to be created. A DMA channel (out) is created to transfer data to the SDIO device connected to any of the S-ports, and another DMA channel (in) is created to read data from the SDIO device. Any of the six available S-port sockets can be used for these channels. Sockets are defined as follows inside the FX3 SDK library.

Transferring infinite blocks of data (by setting transferCount to 0 in block mode) is not supported on the FX3S devices. If transferCount is set to 0, FX3S will transfer 512 bytes of data in multibyte transfer mode, but will throw an error in multiblock mode. Table 9-4 shows the number of bytes and blocks transferred based on transferCount values.

Table 9-4. Byte/ Block Count Values

Transfer Count Value	0x000	0x001	0x002	-----	0x1FF
Bytes Transferred	512	1	2	-----	511
Blocks Transferred	illegal	1	2	-----	511

The following code snippet demonstrates the use of this API.

```
CyU3PDmaChannelReset (&glChHandleSdiotoCpu);

size = ((length+511)/512)*512; /* make sure that size is 512 aligned */
dmaBuffer.size   =  size;
dmaBuffer.buffer = buff;
dmaBuffer.count  = 0;
dmaBuffer.status = 0;

status = CyU3PDmaChannelSetupRecvBuffer (&glChHandleSdiotoCpu, &dmaBuffer);
if (status != CY_U3P_SUCCESS)
{
   ERROR_PRINT (6, "CyU3PDmaChannelSetupSendBuffer error status = %d\r\n", status);
   CyU3PDmaChannelReset (&glChHandleSdiotoCpu);
```

```
   return status;
}

status = CyU3PSdioExtendedReadWrite (0, 1 /*functionNumber*/, 0 /*isWrite*/,
             1 /*blockMode*/, 0 /* opCode */, 00 /*registerAddr*/, (size/512)
            /*transferCount*/,  glChHandleSdiotoCpu.prodSckId /*sckId*/);

if (status != CY_U3P_SUCCESS)
{
   ERROR_PRINT (6, "CyU3PSdioExtendedReadWrite error status = %d\r\n", status);
   CyU3PSibAbortRequest(0);
   return status;
}
```

9.5.3.4 Setting Function Block Size

The function CyU3PSdioSetBlockSize sets the block size for a function. The block size set should be lower than the maximum block size value read from the CISTPL_FUNCE CIS tuple for the function. The value set to the card using this API is saved in the driver and is used as the transfer block size in case of extended transfers.

```
DEBUG_PRINT (8, "setting Func %d block size to 512 \r\n", funcNo);
status = CyU3PSdioSetBlockSize (portId, funcNo, 0x200);
if (status != CY_U3P_SUCCESS)
{
   ERROR_PRINT (8, "failed to set block size %d \r\n", status);
   return status;
}
```

9.5.3.5 Initialization and Operation of SDIO Functions

The SDIO function initialization and application specific functionality or I/O-only card-specific functionalities can be implemented using the CyU3PSdioByteReadWrite and CyU3PSdioExtendedReadWrite APIs in the application firmware.

9.5.3.6 SDIO Interrupts

To allow the SDIO card to interrupt the host, an interrupt function is added to a pin on the SD interface. Pin number 8, which is used as DAT[1], is used to signal the card's interrupt to the host. The use of the interrupt is optional for each card or function within a card. The SDIO interrupt is "level sensitive," that is, the interrupt line is held active (low) until it is either recognized and acted upon by the host or deasserted due to the end of the interrupt period. Once the host has serviced the interrupt, it is cleared via some function-unique I/O operation.

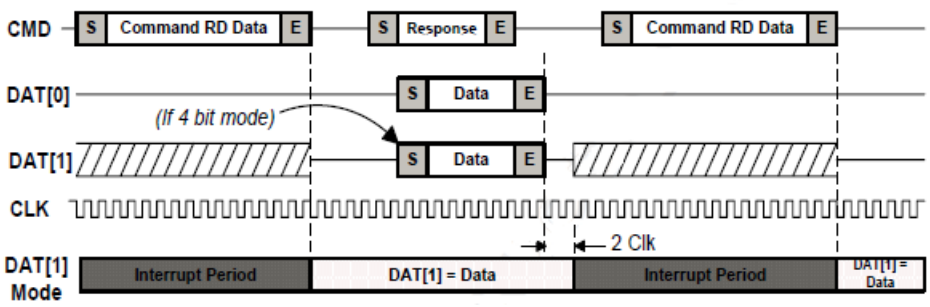
The 1-bit SD mode, pin 8 is dedicated to the interrupt function. Thus, in the SPI and SD 1-bit modes, there are no timing constraints on interrupts. A card in the SPI or 1-bit SD mode signals an interrupt to the host at any time by asserting pin 8 low.

Since pin 8 is shared between the IRQ and DAT[1] in the 4-bit SD mode, an interrupt is sent by the card and recognized by the host only during a specific time. The time that a low on pin 8 is recognized as an interrupt is defined as the interrupt period. An SDIO host samples the level on pin 8 (DAT[1]/IRQ) into the interrupt detector only during the interrupt period. At all other times, the host interrupt controller ignores the level on pin 8.

The interrupt period begins after a function is initialized (IOEx), the card is placed into 4-bit SD mode, and interrupts are enabled (IENM and IENx). A card, depending on design, may require a longer time until it is able to issue interrupts.

The interrupt period ends at the next clock from the end bit of a command that transfers data block(s) using DAT[x] lines. It resumes two clocks after the completion of the last data block transfer in a transaction.

Figure 9-10. Read Interrupt Cycle Timing *



* Source: SDIO Specification, version 2.0

This interrupt period is intended to prevent interaction between the DAT[1] data and the interrupt signals on a common pin. Figure 9-10 shows the timing of the interrupt period for single data transaction read cycles. For more details, refer to the SDIO specification.

9.5.3.7 Enabling and Disabling SDIO Interrupts

SDIO interrupts can be enabled or disabled by setting or clearing the bit SDMMC_INTR_MASK.SDIO_INTR_MASK. The CyU3PSdioInterruptControl API can be used to enable or disable interrupts on the SDIO card. A request to enable interrupts for an SDIO function (1-7) automatically enables the card's interrupt master.

9.5.3.8 Handling SDIO Interrupts

An SDIO interrupt is notified via the SDMMC_INTR.SDIO_INTR bit. The SDIO interrupt is notified to the application firmware in the form of an event registered through the SIB callback (explained in 9.5.1.6 Handling SIB Events on page 214).

```
/* Register a callback for SIB events. */
status = CyU3PSibRegisterCbk (CyFxSDIOUARTApplnSibCB);

/* SIB Callback for handling SIB events */
void
CyFxSDIOUARTApplnSibCB (
     uint8_t             portId,
     CyU3PSibEventType   evt,
     CyU3PReturnStatus_t status)
{

      if (evt == CY_U3P_SIB_EVENT_SDIO_INTR)
   {

         /* Handle SDIO interrupt event */

      }

}
```

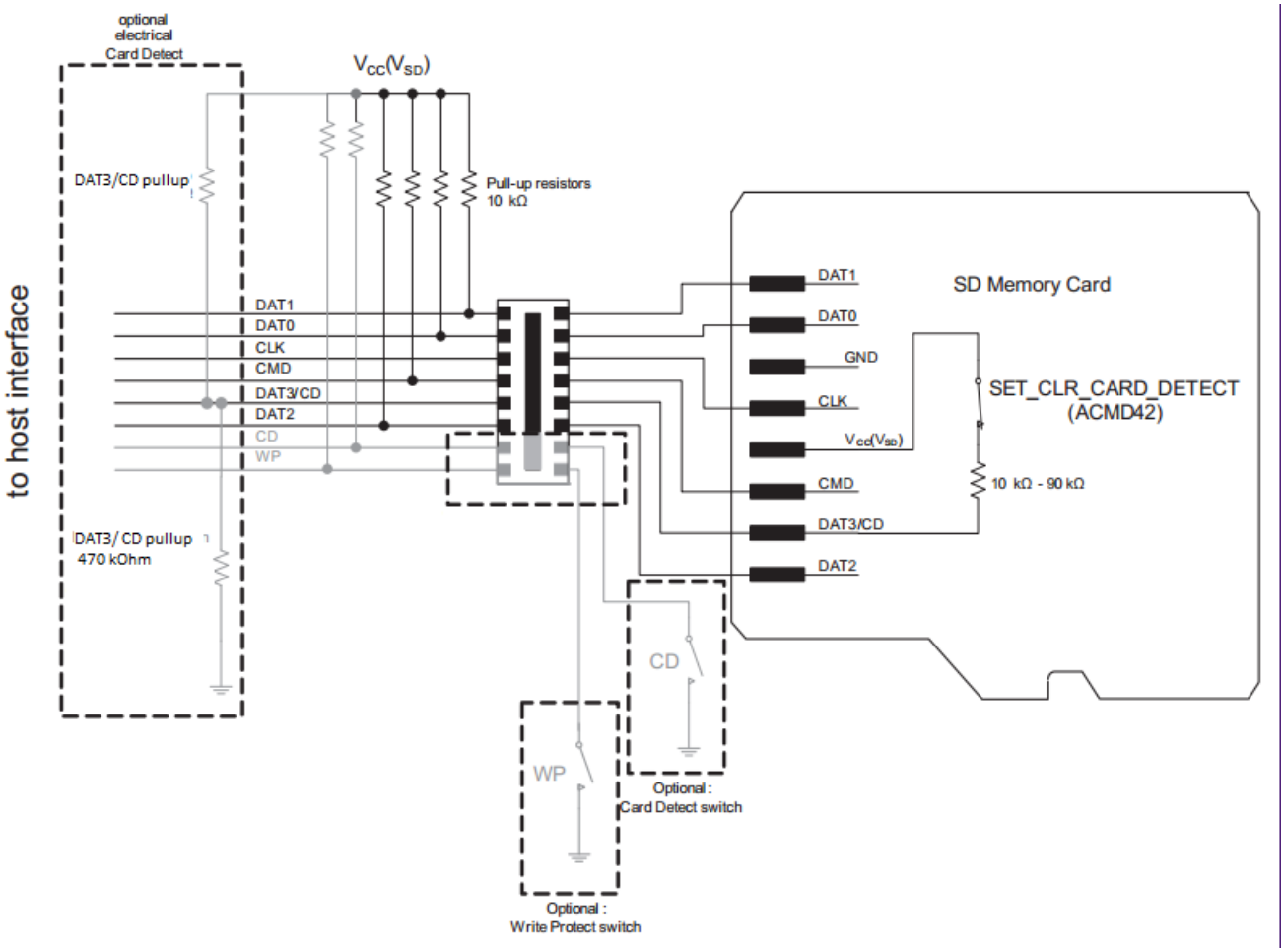
9.6 FX3S-Specific Features

9.6.1 Card Insertion and Removal Detection Mechanism

FX3S supports the two-card insertion and removal detection mechanisms.

- Use of SD_D[3] data: This signal must have an external 470-k pull-down resistor connected to SD_D[3]. SD cards have an internal 10-k pull-up resistor. When the card is inserted or removed from the SD/MMC connector, the voltage level at the SD_D[3] pin changes and triggers an interrupt to the CPU.
- Use of the S0/S1_INS pin: Some SD/MMC connectors include a microswitch for card insertion/removal detection. This microswitch can be connected to S0/S1_INS. When the card is inserted or removed from the SD/MMC connector, it turns the microswitch on and off. This changes the voltage level at the pin that triggers the interrupt to the CPU. S0/ S1_INS stays high or floating when card is not inserted and goes low when card is inserted. The card-detect microswitch polarity is assumed to be the same as the write-protect microswitch polarity. A low indicates that the card is inserted. This S0/ S1_INS pin is shared between the two S-ports. Register configuration determines which port gets to use the pin. This pin is mapped to the VIO2 power domain (see Table 9-1); if S0 and S1 are at different voltage levels, the pin cannot be used as S1_INS. Figure 9-11 depicts this mechanism.

Figure 9-11. Card Detection Using GPIO



9.6.2 Handling Card Detection in Software

The card detection method can be chosen in the firmware according to its support on the card as well as on the hardware. The card detection method is chosen using the field cardDetType through API CyU3PSibSetIntfParams. Accordingly, the interrupt will be enabled in the SDMMC_INTR_MASK register. The CARD_DETECT interrupt is enabled in the card detection method based on GPIO, or the DAT3_STAT interrupt is enabled in the card detection method based on the DAT3 line.

```
intfParams.resetGpio = 0xFF; /* No GPIO control on SD/MMC power. */
    intfParams.rstActHigh = CyTrue;  /* Don't care as no GPIO is selected. */
    intfParams.cardDetType = CY_U3P_SIB_DETECT_DAT_3;  /* Card detect based
                            on SD_DAT[3]. */
    intfParams.writeProtEnable = CyTrue; /* Write protect handling enabled.*/
    intfParams.lowVoltage = CyTrue;     /* Low voltage operation enabled. */
    intfParams.voltageSwGpio = 45;     /* Use GPIO_45 for voltage switch on
                  S0 port. */
    intfParams.lvGpioState = CyFalse;  /* Driving GPIO low selects 1.8 V on
                  SxVDDQ. */
    intfParams.useDdr = CyTrue;        /* DDR clocking enabled. */
    intfParams.maxFreq = CY_U3P_SIB_FREQ_104MHZ; /* No S port clock
                         limitation. */
    intfParams.cardInitDelay = 0; /* No delay required between SD card    insertion before
initialization. */

    status = CyU3PSibSetIntfParams (0, &intfParams);
```

The card detection mechanism supported is provided through the enumerated list CyU3PSibCardDetect, as follows. The card detection can be disabled by selecting CY_U3P_SIB_DETECT_NONE.

```
/** \brief List of card detection methods.

    **Description**\n
    The card insertion and removal detection can be based on either GPIO card detect
    or based on signal changes on the DAT3 line. */
typedef enum CyU3PSibCardDetect
{
    CY_U3P_SIB_DETECT_NONE = 0,         /**< Don't use any card detection mechanism */
    CY_U3P_SIB_DETECT_GPIO,             /**< Use a GPIO connected to a micro-switch on the
socket. This is
                                             only supported for the S0 storage port. */
    CY_U3P_SIB_DETECT_DAT_3            /**< Use voltage changes on the DAT[3] pin for card
detection. */
} CyU3PSibCardDetect;
```

If the card detect mechanism is chosen as GPIO based, then the GPIO44 is chosen as a default by the FX3S software driver and library. The FX3S library and driver configure the GPIO in such a case by default, and the card detect interrupt will be notified to the application firmware via the SIB callback registered (explained in 9.5.1.6 Handling SIB Events on page 214), as shown in the following example. This is the case even with DAT3-based card detection.

Note: Any unused GPIO can be used for card detection, which can be configured like any simple GPIO in the application firmware. Refer to 4.1 GPIO Pins on page 53 for more details.

```
/* Register a callback for SIB events. */
status = CyU3PSibRegisterCbk (CyFxSDIOUARTApplnSibCB);

/* SIB Callback for handling SIB events */
void
```

```
CyFxSDIOUARTApplnSibCB (
     uint8_t             portId,
     CyU3PSibEventType   evt,
     CyU3PReturnStatus_t status)
{

   if (evt == CY_U3P_SIB_EVENT_INSERT)
   {
      /* Handle card insert event */
   }
   if (evt == CY_U3P_SIB_EVENT_REMOVE)
   {
      /* Handle card remove event */
   }

}
```

9.6.3 Write Protection

The S0_WP/ S1_WP (SD write protection) on the S-port is used to connect to the WP microswitch of the SD/MMC card connector. This pin internally connects to a CPU-accessible GPIO for firmware to detect the SD card write protection. Write protection can be enabled using the writeProtEnable field of the structure passed to CyU3PSibSetIntfParams. If enabled. GPIO 43 acts as S0_WP and GPIO52 acts as S1_WP.

9.6.4 SD/MMC CLOCK STOP

FX3S supports the stop clock feature, which can save power if the internal buffer is full when receiving data from the SD/ MMC/SDIO.

SDMMC_MODE_CFG. RD_STOP_CLK_EN enables the SIB to detect overflow and stop the clock to the SD/SDIO/MMC card during data transfer.

SDMMC_MODE_CFG. WR_STOP_CLK_EN enables SIB to detect underflow and stop the clock to the SD/SDIO/MMC card during data transfer.

Refer to 9.5.1.5 Sending SD/MMC/SDIO Commands on page 212 for the SDMMC_MODE_CFG register description.

9.6.5 SD_CLK Output Clock Stop

During the data transfer, the SD_CLK clock can be enabled (on) or disabled (stopped) at any time by the internal flow control mechanism. The SD_CLK output frequency is dynamically configurable using a clock divisor from a system clock. Refer to Setting the S-Port Clock on page 212.

9.6.6 SDIO Read-Wait/ Suspend-Resume Feature

FX3S supports the optional read-wait and suspend-resume features as defined in the SDIO specification version 2.00 (January 30, 2007).

9.6.6.1 Read-Wait

Host devices built to the SD physical specification control the SDCLK to stop the read data block output from a card executing a multiple read command whenever the host cannot accept more data. During the time that the host has stopped the SDCLK, a CMD52 cannot be issued. This limitation creates the problem that a host device built to the SD physical specification cannot perform the I/O command during a multiple read cycle.

To eliminate this limitation, the SDIO specification adds the read-wait control to enable the host to issue CMD52 during a multiple read cycle. Read-wait uses the DAT[2] line to allow the host to signal the card to temporarily halt the sending of read

data by a card. This feature is optional for an SDIO or combo card. However, if an SDIO or combo supports read-wait, all functions and any memory will support read-wait. For more details, refer to the SDIO specification. Any card that supports suspend-resume will also support read-wait.

The CyU3PSdioReadWaitEnable API is used to enable read-wait on the SDIO bus. This function triggers read-wait on a card that supports the feature using the DAT[2] of the SDIO data bus. It is enabled by setting the bit SDMMC_CS. SDIO_READ_WAIT_EN.

9.6.6.2 Suspend-Resume Feature

Within a multifunction SDIO or a combo card, there are multiple devices (I/O and memory) that share access to the SD bus. In order to allow sharing of access to the host among multiple devices, SDIO and combo cards can implement the optional concept of suspend-resume. If a card supports suspend-resume, the host may temporarily halt a data transfer operation to one function or memory (suspend) in order to free the bus for a higher priority transfer to a different function or memory. Once this higher priority transfer is complete, the original transfer is restarted where it left off (resume). Support of suspend-resume is optional on a per-card basis. If suspend-resume is implemented, it will be supported by the memory (if any) of a combo card and all I/O functions except function 0. Note that the host can suspend multiple transactions and resume them in any order desired. I/O function 0 does not support suspend-resume.

The CyU3PSdioSuspendResumeFunction API is used to suspend or resume the specified SDIO function. Suspend-resume needs to be supported by the card to which these calls are being addressed. Support for suspend-resume can be ascertained by looking at the SBS bit of the card's card capability register. When the function is resumed, it also checks if any data is available.

9.6.6.3 SD3.0 Host Tuning Feature

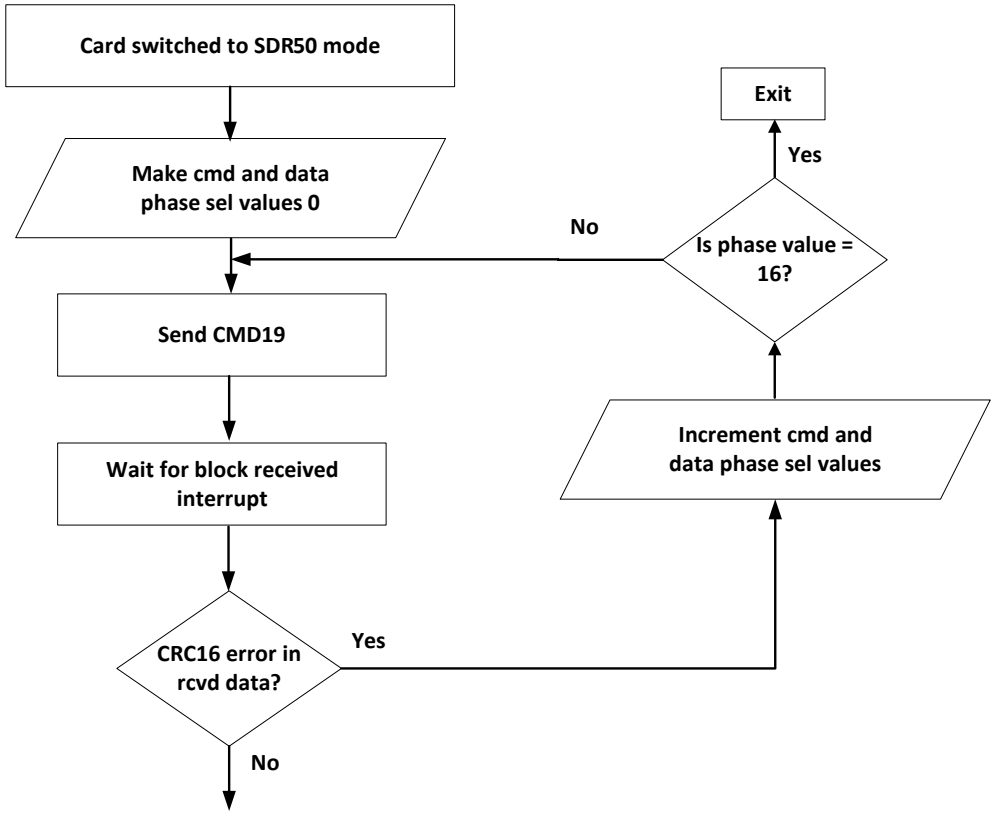
SD3.0 supports the tuning feature to find the optimal sampling point in the host during the SDR50, SDR104, and DDR104 modes of operation. CMD19 is the special command that is used in SD3.0 for starting the tuning operation. SDR104 and DDR104 are not supported in FX3S. The host generally supports two types of sampling:

- Fixed sampling: SDR-FD-SDR signaling, fixed delay (can't use tuning). This method is available up to only 100-MHz frequency.
- Variable sampling: SDR-VD-SDR signaling, variable delay (uses tuning block). The CMD19 is used by the host to read the tuning block.

The arguments of the CMD19 are zero, and response R1 is defined for this command. The card sends a 64-byte block of data on the DAT[3:0] lines with predefined CRC bits. This data is used by the host to compare with a predefined tuning block pattern and to increment the sampling control block by one step. Per the specification, there can be 40 executions of CMD19 in no more than 150 ms. CMD19 always returns a 64-byte tuning pattern sent by the card. After the CPU sets up the SIB sockets, command transmission is initiated and completed in the same manner as described in 9.5.1.5 Sending SD/MMC/SDIO Commands on page 212. Receipt of the 64-byte tuning pattern is indicated by the block received interrupt (SDMMC_INTR.BLOCKS_RECEIVED). The host verifies the data integrity of the tuning block and adjusts its sampling point accordingly. For register descriptions, refer to SDMMC_DLL_CTRL register on page 668.

The tuning can be implemented as follows. The SIB has registers for selecting the CMD and data phase values in the SDMMC DLL control register. Increment these values from 0 until a point where the tuning data reception does not lead to a CRC16 error. When the phase values exceed the maximum possible value (15), all 16 phase values possible have returned CRC16 errors and hence a fatal error is reported.

Figure 9-12. SD 3.0 Host Tuning Feature



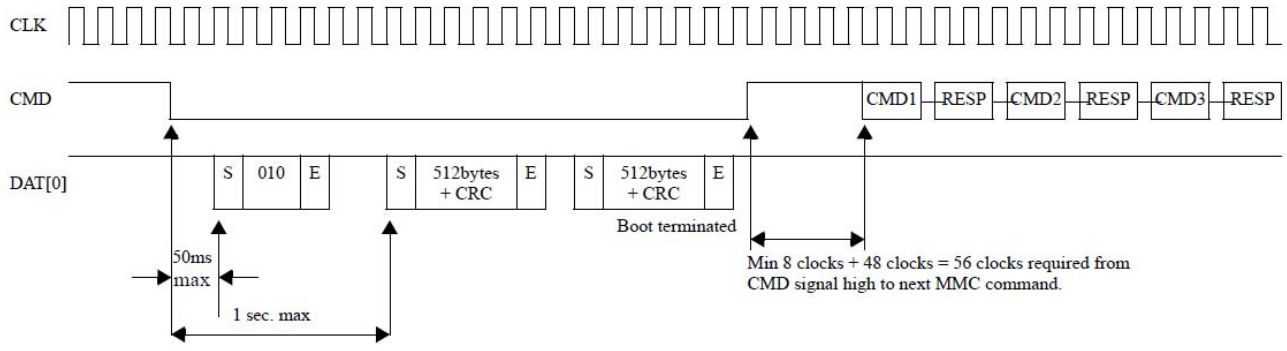
More details on the SD3.0 host tuning methodology are documented in the SD3.0 specification.

9.6.6.4 *Normal and Alternate eMMC4.4 Boot*

The eMMC 4.4 boot mode is supported in FX3S. Two types of boot operations are defined in eMMC 4.4: normal boot and alternate boot. FX3S supports both modes of operation.

In normal boot, the host (FX3S S-port) pulls the CMD line low for at least 74 clock cycles after power up or reset. SW writes '1' to the bit SDMMCU_CS.BOOTCMD to initiate a CMD_LINE_LOW boot (or alternate boot) command. With the alternate boot command, the CPU must send the command. The card recognizes that a boot is initiated and sends the boot acknowledge followed by the boot data. Once all boot data is read, the host pulls the CMD line HIGH, as illustrated in Figure 9-13.

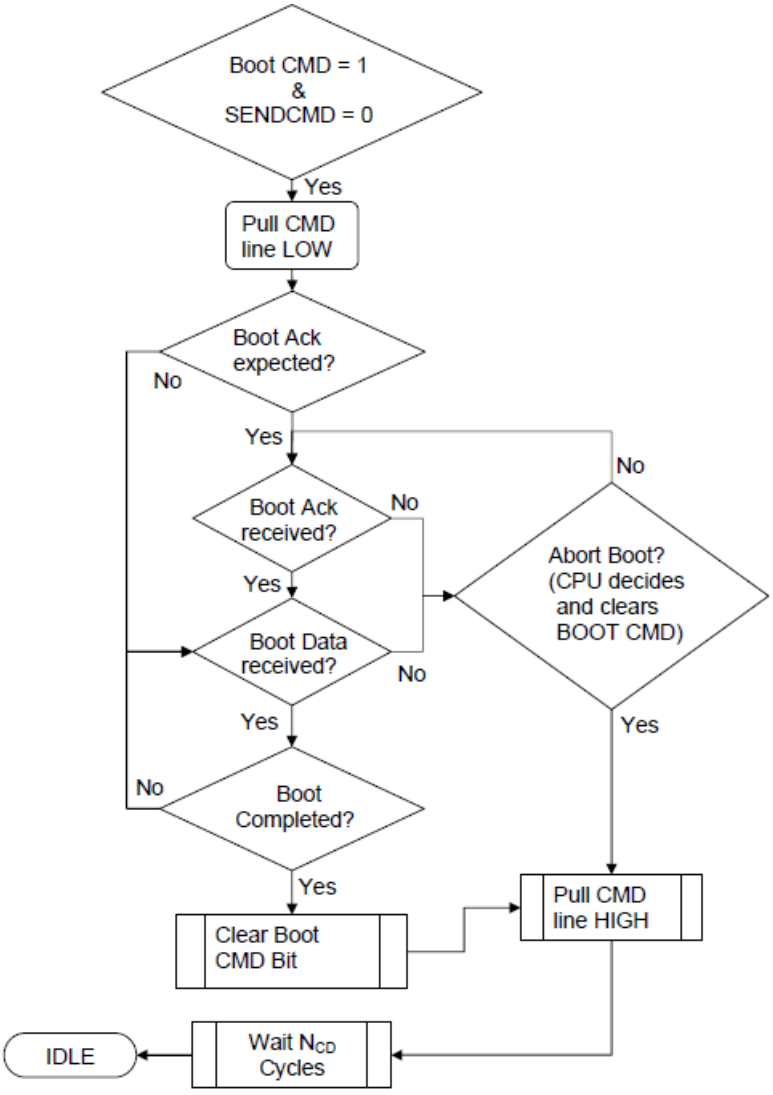
Figure 9-13. Normal Boot*



* Source: eMMC Specification, version 4.3

The boot's acknowledge can be disabled by setting the EXT_CSD register in the card. The boot operation can be terminated by the host by pulling the CMD line HIGH. The termination can occur during data transfer or between consecutive data blocks. The host does not interpret between them. Writing 1 to SDMMCU_CS.CLR_BOOTCMD causes the CPU to terminate the boot and return to IDLE. Figure 9-14 depicts the flow chart for normal boot.

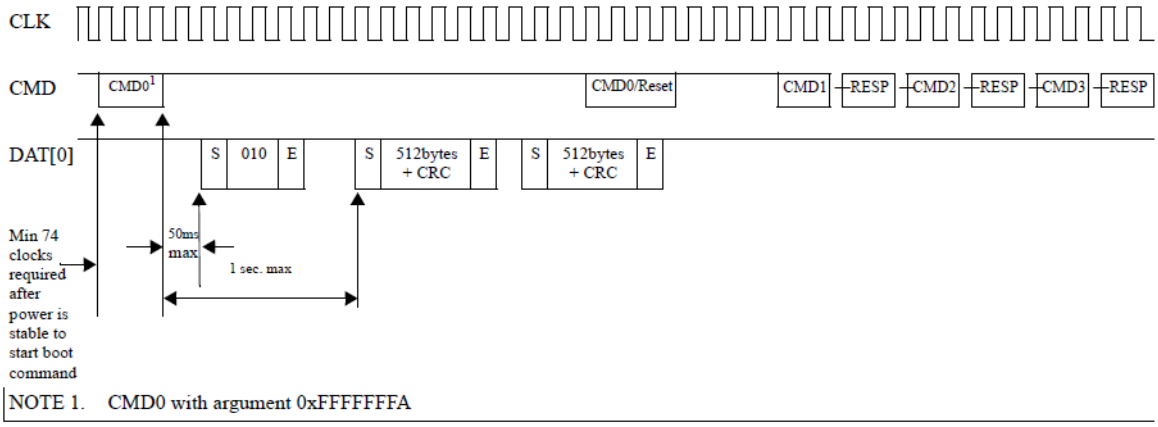
Figure 9-14. Flowchart for Normal Boot



The CPU may terminate a boot due to a CRC error or boot ack timeout. In a boot termination during data transfer, the CPU has to bring the SIB FSMs to a known state by setting the RSTCONT bit in the SDMMCU_CS register and also ensure that the associated sockets are brought back to a known state. More details on normal boot operation can be found in the SD 3.0 specification.

In alternate boot operation mode, If CMD0 is issued with argument 0XFFFFFFFA, 74 clock cycles after power up or reset, the card sends the boot acknowledgement status followed by the boot data. The boot acknowledgment can be disabled as in the case of normal boot. The termination of boot operation is triggered by issuing a reset command (CMD0 with argument 0XF0F0F0F0). As in the case of normal boot, the termination can occur during block boundary or data transfer. The condition for termination described previously for normal boot is the same for alternate boot.

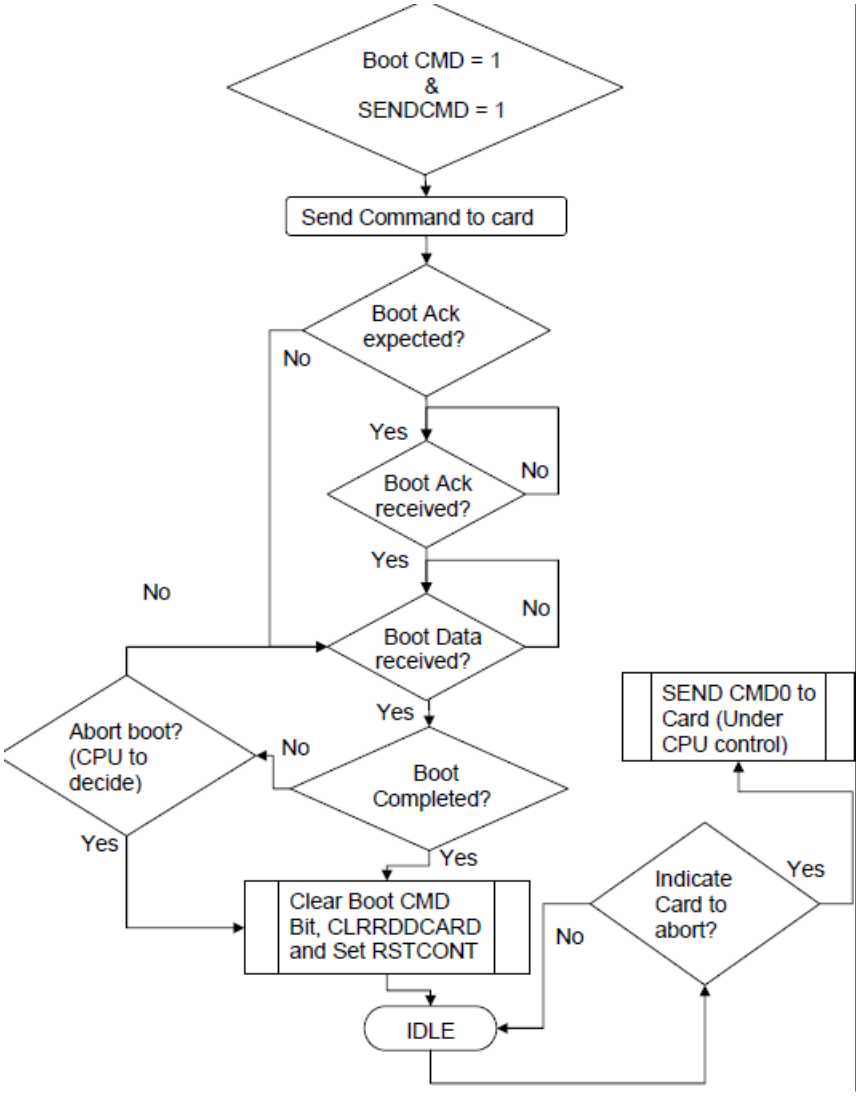
Figure 9-15. Alternate Boot



* Source: eMMC Specification, version 4.3

The flow chart shown in Figure 9-16 depicts the alternate boot operation flow.

Figure 9-16. Flow Chart for Alternate Boot Operation *



10. Registers

10.1 Introduction

This chapter describes the EZ-USB FX3 registers.

Table 10-1 provides an overview of the FX3 memory map:

Table 10-1. FX3 Memory Map

Offset	Size	Name	Description
0x00000000	0x4000	ITCM_MEMORY	16-KB Instruction Tightly Coupled Memory (visible to ARM only)
0x10000000	0x2000	DTCM_MEMORY	8-KB Data Tightly Coupled Memory (visible to ARM only)
0x40000000	0x80000	SYSMEM	512-KB Main System RAM
0xE0000000	0x10000000	MMIO	Main MMIO Space
0xFFFF0000	0x8000	BOOTROM	32-KB Boot ROM Memory
0xFFFFF000	0x1000	VIC	ARM PL192 VIC Vectored Interrupt Controller

Table 10-2 provides an overview of the MMIO register space starting at 0xE0000000 (as shown in Table 10-1):

Table 10-2. MMIO Register Space

Offset (from address 0xE0000000)	Size	Name	Description
0x00000	0x10000	LPP	Low Performance Peripherals register map
0x10000	0x10000	PIB	Processor Port register map
0x30000	0x10000	UIB	USB Port register map
0x40000	0x10000	UIBIN	USB SuperSpeed Ingress Adapter
0x50000	0xFC00	GCTL	Global Controller register map
0x5FC00	0x0400	CPU	ARM CPU register map (BIST only)

Detailed descriptions of all the registers in each IP block are provided in the following sections.

10.2 Register Conventions

Register Name			Register Function				Address
b7	b6	b5	b4	b3	b2	b1	b0
bitname	**bitname**	**bitname**	**bitname**	**bitname**	**bitname**	**bitname**	**bitname**
SW R, W access	SW R, W access	SW R, W access	SW R, W access	SW R, W access	SW R, W access	SW R, W access	SW R, W access
HW R, W access	HW R, W access	HW R, W access	HW R, W access	HW R, W access	HW R, W access	HW R, W access	HW R, W access
Default val	Default val	Default val	Default val	Default val	Default val	Default val	Default val

The register table above illustrates the register description format used in this chapter.

- The top line shows the register name, functional description, and address in the memory.
- The second line shows the bit position in the register.
- The third line shows the field name of the corresponding bit(s) in the register.
- The fourth line shows SW CPU accessibility: R(ead), W(rite), W0C (write 0 to clear), W1S (write 1 to set), or R/W.
- The fifth line shows HW CPU accessibility: R(ead), W(rite), W0C (write 0 to clear), W1S (write 1 to set), or R/W.
- The sixth line shows the default value. These values apply after a hard reset (refer to 4.3.3 Reset on page 59). When a default value extends across bytes, the borders between the bytes is a lighter line.
- Gray, empty cells indicate reserved bits. Do not read from or write to these bits.

10.3 Vectored Interrupt Controller (VIC) Registers

10.3.1 VIC_IRQ_STATUS

IRQ Status Register

VIC_IRQ_STATUS			IRQ Status after Masking				0xFFFFF000
b31	b30	b29	b28	b27	b26	b25	b24
IRQ_STATUS[31:24]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
VIC_IRQ_STATUS			IRQ Status after Masking				
b23	b22	b21	b20	b19	b18	b17	b16
IRQ_STATUS[23:16]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
VIC_IRQ_STATUS			IRQ Status after Masking				
b15	b14	b13	b12	b11	b10	b9	b8
IRQ_STATUS[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
VIC_IRQ_STATUS			IRQ Status after Masking				
b7	b6	b5	b4	b3	b2	b1	b0
IRQ_STATUS[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Name	Description	
31:0	**IRQ_STATUS[31:0]**	1	IRQ interrupt is raised by the following source:

10.3.2 VIC_FIQ_STATUS

FIQ Status Register

VIC_FIQ_STATUS			FIQ Status after Masking				0xFFFFF004
b31	b30	b29	b28	b27	b26	b25	b24
FIQ_STATUS[31:24]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
VIC_FIQ_STATUS			**FIQ Status after Masking**				
b23	b22	b21	b20	b19	b18	b17	b16
FIQ_STATUS[23:16]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
VIC_FIQ_STATUS			**FIQ Status after Masking**				
b15	b14	b13	b12	b11	b10	b9	b8
FIQ_STATUS[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
VIC_FIQ_STATUS			**FIQ Status after Masking**				
b7	b6	b5	b4	b3	b2	b1	b0
FIQ_STATUS[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Name		Description
31:0	**FIQ_STATUS[31:0]**	1	FIQ interrupt is raised at the line corresponding to the bit position.

10.3.3 VIC_RAW_STATUS

Raw Status Register

VIC_RAW_STATUS			IRQ Status before Masking				0xFFFFF008
b31	b30	b29	b28	b27	b26	b25	b24
RAW_STATUS[31:24]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
VIC_RAW_STATUS			**IRQ Status before Masking**				
b23	b22	b21	b20	b19	b18	b17	b16
RAW_STATUS[23:16]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
VIC_RAW_STATUS			**IRQ Status before Masking**				
b15	b14	b13	b12	b11	b10	b9	b8
RAW_STATUS[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
VIC_RAW_STATUS			**IRQ Status before Masking**				
b7	b6	b5	b4	b3	b2	b1	b0
RAW_STATUS[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Name		Description
31:0	**RAW_STATUS[31:0]**	1	FIQ/IRQ interrupt is raised at the line corresponding to the bit position regardless of its masked (disable) state.

10.3.4 VIC_INT_SELECT

IRQ/FIQ Designation Register

VIC_INT_SELECT			IRQ/FIQ Designation Register				0xFFFFF00C
b31	b30	b29	b28	b27	b26	b25	b24
FIQ_nIRQ[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
VIC_INT_SELECT			IRQ/FIQ Designation Register				
b23	b22	b21	b20	b19	b18	b17	b16
FIQ_nIRQ[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
VIC_INT_SELECT			IRQ/FIQ Designation Register				
b15	b14	b13	b12	b11	b10	b9	b8
FIQ_nIRQ[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
VIC_INT_SELECT			IRQ/FIQ Designation Register				
b7	b6	b5	b4	b3	b2	b1	b0
FIQ_nIRQ[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Bit	Name		Description
31:0	FIQ_nIRQ[31:0]	1	Designate the line corresponding to the bit position as FIQ. The software ensures that this register is a power of 2 (one FIQ only). It writes to this register only after disabling the interrupts it intends to change.

10.3.5 VIC_INT_ENABLE

Interrupt Enable Register

VIC_INT_ENABLE			Interrupt Enable Register				0xFFFFF010
b31	b30	b29	b28	b27	b26	b25	b24
INT_ENABLE[31:24]							
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
VIC_INT_ENABLE			**Interrupt Enable Register**				
b23	b22	b21	b20	b19	b18	b17	b16
INT_ENABLE[23:16]							
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
VIC_INT_ENABLE			**Interrupt Enable Register**				
b15	b14	b13	b12	b11	b10	b9	b8
INT_ENABLE[15:8]							
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
VIC_INT_ENABLE			**Interrupt Enable Register**				
b7	b6	b5	b4	b3	b2	b1	b0
INT_ENABLE[7:0]							
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Bit	Name		Description
31:0	**INT_ENABLE[31:0]**	1	Enable the interrupt at this bit position. All interrupts are disabled at reset. Software cannot write 0 here to disable interrupts. Use the VIC_INT_CLEAR register for this purpose.

10.3.6 VIC_INT_CLEAR

Interrupt Clear Register

VIC_INT_CLEAR			Interrupt Clear Register				0xFFFFF014
b31	b30	b29	b28	b27	b26	b25	b24
INT_CLEAR[31:24]							
W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C
R	R	R	R	R	R	R	R
NA	NA	NA	NA	NA	NA	NA	NA
VIC_INT_CLEAR			**Interrupt Clear Register**				
b23	b22	b21	b20	b19	b18	b17	b16
INT_CLEAR[23:16]							
W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C
R	R	R	R	R	R	R	R
NA	NA	NA	NA	NA	NA	NA	NA
VIC_INT_CLEAR			**Interrupt Clear Register**				
b15	b14	b13	b12	b11	b10	b9	b8
INT_CLEAR[15:8]							
W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C
R	R	R	R	R	R	R	R
NA	NA	NA	NA	NA	NA	NA	NA
VIC_INT_CLEAR			**Interrupt Clear Register**				
b7	b6	b5	b4	b3	b2	b1	b0
INT_CLEAR[7:0]							
W1C	W1C	W1C	W1C	W1C	W1C	W1C	W1C
R	R	R	R	R	R	R	R
NA	NA	NA	NA	NA	NA	NA	NA

This register disables the interrupt line and masks it if it was designated IRQ.

Bit	Name	Description	
31:0	**INT_CLEAR[31:0]**	1	Disable the interrupt at this bit position. Mask it if it is designated IRQ.

10.3.7 VIC_PRIORITY_MASK

Per-Priority Interrupt Mask Register

VIC_PRIORITY_MASK			Per-Priority Interrupt Mask Register				0xFFFFF024
b31	b30	b29	b28	b27	b26	b25	b24

VIC_PRIORITY_MASK			Per-Priority Interrupt Mask Register				
b23	b22	b21	b20	b19	b18	b17	b16

VIC_PRIORITY_MASK			Per-Priority Interrupt Mask Register				
b15	b14	b13	b12	b11	b10	b9	b8
PRIO_MASK[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

VIC_PRIORITY_MASK			Per-Priority Interrupt Mask Register				
b7	b6	b5	b4	b3	b2	b1	b0
PRIO_MASK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFF							

This register allows you to mask interrupts on a per-priority basis.

Bit	Name	Description	
15:0	**PRIO_MASK[15:0]**	1	Unmasked
		0	Interrupt at the priority level = bit position is masked.

10.3.8 VIC_VEC_ADDRESS

Interrupt Vector Register

There are 32 VIC_VEC_ADDRESS registers. The address of each is calculated as VIC_VEC_ADDRESS(x) = 0xFFFFF100 + (x*0x4). So, VIC_VEC_ADDRESS(0) is at address 0xFFFFF100, VIC_VEC_ADDRESS(1) is at address 0xFFFFF100 + 0x4 and so on. The definition of each of these is the same.

VIC_VEC_ADDRESS			Interrupt Vector Register				0xFFFFF100
b31	b30	b29	b28	b27	b26	b25	b24
ISR_ADD[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

VIC_VEC_ADDRESS			Interrupt Vector Register				
b23	b22	b21	b20	b19	b18	b17	b16
ISR_ADD[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

VIC_VEC_ADDRESS			Interrupt Vector Register				
b15	b14	b13	b12	b11	b10	b9	b8
ISR_ADD[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

VIC_VEC_ADDRESS			Interrupt Vector Register				
b7	b6	b5	b4	b3	b2	b1	b0
ISR_ADD[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This registers holds the address of ISR for interrupts. 32 registers for 32 lines. There are 32 of these registers, which are contiguous. Note the offset for this register bank.

Bit	Name	Description
31:0	**ISR_ADD[31:0]**	The firmware accesses this register only after disabling the corresponding interrupt. Holds the address to the ISR for interrupt number = the position of this register in the bank of 32 registers.

10.3.9 VIC_VECT_PRIORITY

Interrupt Priority Register

There are 32 VIC_VECT_PRIORITY registers. The address of each is calculated as VIC_VECT_PRIORITY(x) = 0xFFFFF200 + (x*0x4). So, VIC_VECT_PRIORITY(0) is at address 0xFFFFF200, VIC_VECT_PRIORITY(1) is at address FFFFF200 + 0x4,and so on. The definition of each of these is the same.

VIC_VECT_PRIORITY			Interrupt Priority Register				0xFFFFF200
b31	b30	b29	b28	b27	b26	b25	b24

VIC_VECT_PRIORITY			Interrupt Priority Register				
b23	b22	b21	b20	b19	b18	b17	b16

VIC_VECT_PRIORITY			Interrupt Priority Register				
b15	b14	b13	b12	b11	b10	b9	b8

VIC_VECT_PRIORITY			Interrupt Priority Register				
b7	b6	b5	b4	b3	b2	b1	b0
				VIC_INTR_PRI[3:0]			
				R/W	R/W	R/W	R/W
				R	R	R	R
				0xF			

This register holds the priority designation of the interrupts. There are 32 registers for 32 lines. The 32 registers are contiguous. Note the offset for this register bank.

Bit	Name	Description
3:0	VIC_INTR_PRI	The firmware accesses this register only after disabling the corresponding interrupt. It assigns one of the 16 priorities to the interrupt number. This equals the position of this register in the bank of 32 registers. When any two interrupts with the same priority arrive at the VIC, the one connected to the lower numbered line wins.

.

10.3.10 VIC_ADDRESS

Active ISR Address Register

VIC_ADDRESS			Active ISR Address Register				0xFFFFFF00
b31	b30	b29	b28	b27	b26	b25	b24
ACTIVE_ISR[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
VIC_ADDRESS			Active ISR Address Register				
b23	b22	b21	b20	b19	b18	b17	b16
ACTIVE_ISR[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
VIC_ADDRESS			Active ISR Address Register				
b15	b14	b13	b12	b11	b10	b9	b8
ACTIVE_ISR[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
VIC_ADDRESS			Active ISR Address Register				
b7	b6	b5	b4	b3	b2	b1	b0
ACTIVE_ISR[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

This register returns the ISR address of the interrupt deemed active after masking and priority resolution. This is what the CPU reads when interrupted.

Bit	Name	Description
31:0	ACTIVE_ISR[31:0]	Upon interruption, software reads this register. This marks the active interrupt as being serviced, which prevents interrupts of equal or lower priority from raising interrupt. Upon completion, the software writes any value to this register, which clears the active interrupt. software does not access this register at any other time.

10.4 Global Controller Registers

10.4.1 GCTL_IOMATRIX

I/O Matrix Configuration Register

GCTL_IOMATRIX			I/O Matrix Configuration Register				0xE0051008
b31	b30	b29	b28	b27	b26	b25	b24

GCTL_IOMATRIX			I/O Matrix Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_IOMATRIX			I/O Matrix Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
					S1CFG[2:0]		
					R/W	R/W	R/W
					R	R	R
					0	0	0

GCTL_IOMATRIX			I/O Matrix Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
			S0CFG			CARKIT	
			R/W			R/W	
			R			R	
			0			0	

This register defines port usage and implementation for the GPIO port.

Bit	Name	Description
10:8	S1CFG[2:0]	GPIO[57:46] is configured with the following encoding: 0 Connected to GPIO 1 Connected to GPIO + UART 2 Connected to GPIO + SPI 3 Connected to GPIO + I2S 4 Connected to UART+SPI+I2S 5 Connected to PIB_DQ+UART+I2S
4	S0CFG	GPIO[45:33] is configured with the following encoding: 0 Connected to GPIO 1 Connected to PIB_DQ[16]..PIB_DQ[27] (GPIF-II mode)
1	CARKIT	Carkit UART configuration: 0 Use LPP_UART (LPP_UART not available to S1) 1 Use PIB_CTL11/PIB_CTL12 pins for carkit UART (enabling the USB2 PHY for Carkit UART operation must be done separately in UIB_PHY*).

10.4.2 GCTL_GPIO_SIMPLE

GPIO Override Configuration Register

GCTL_GPIO_SIMPLE			GPIO Override Configuration Register				0xE005100C
b63	b62	b61	b60	b59	b58	b57	b56
			OVERRIDE[60:56]				
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	0	0	0

GCTL_GPIO_SIMPLE			GPIO Override Configuration Register				
b55	b54	b53	b52	b51	b50	b49	b48
OVERRIDE[55:48]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_GPIO_SIMPLE			GPIO Override Configuration Register				
b47	b46	b45	b44	b43	b42	b41	b40
OVERRIDE[47:40]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_GPIO_SIMPLE			GPIO Override Configuration Register				
b39	b38	b37	b36	b35	b34	b33	b32
OVERRIDE[39:32]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_GPIO_SIMPLE			GPIO Override Configuration Register				
b31	b30	b29	b28	b27	b26	b25	b24
OVERRIDE[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	0	0	0	0	0	0	0

GCTL_GPIO_SIMPLE			GPIO Override Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16
OVERRIDE[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_GPIO_SIMPLE			GPIO Override Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
OVERRIDE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.4.2 GCTL_GPIO_SIMPLE *(continued)*

GCTL_GPIO_SIMPLE			GPIO Override Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
OVERRIDE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register is used to determine if the simple GPIO function should override the (GCTL_IOMATRIX specified) function of each pin on a per-pin basis. GCTL_GPIO_COMPLEX takes precedence over GCTL_GPIO_SIMPLE.

Bit	Name	Description
60:0	**OVERRIDE[60:0**	When bit <n> is set, the corresponding pin maps to simple GPIO <n>.

10.4.3 GCTL_GPIO_COMPLEX

GPIO Override Configuration Register

GCTL_GPIO_COMPLEX			GPIO Override Configuration Register				0xE0051014
b63	b62	b61	b60	b59	b58	b57	b56
			OVERRIDE[60:56]				
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	0	0	0

GCTL_GPIO_COMPLEX			GPIO Override Configuration Register				
b55	b54	b53	b52	b51	b50	b49	b48
OVERRIDE[55:48]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_GPIO_COMPLEX			GPIO Override Configuration Register				
b47	b46	b45	b44	b43	b42	b41	b40
OVERRIDE[47:40]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_GPIO_COMPLEX			GPIO Override Configuration Register				
b39	b38	b37	b36	b35	b34	b33	b32
OVERRIDE[39:32]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_GPIO_COMPLEX			GPIO Override Configuration Register				
b31	b30	b29	b28	b27	b26	b25	b24
OVERRIDE[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	0	0	0	0	0	0	0

GCTL_GPIO_SIMPLE			GPIO Override Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16
OVERRIDE[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_GPIO_COMPLEX			GPIO Override Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
OVERRIDE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.4.3 GCTL_GPIO_COMPLEX *(continued)*

GCTL_GPIO_COMPLEX			GPIO Override Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
OVERRIDE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register is used to determine if the complex GPIO function should override the (GCTL_IOMATRIX specified) function of each pin on a per-pin basis. GCTL_GPIO_COMPLEX takes precedence over GCTL_GPIO_SIMPLE.

Bit	Name	Description
60:0	**OVERRIDE[60:0]**	When bit <n> is set, the corresponding pin maps to complex GPIO_PIN <n> Mod 8. If multiple pins are mapped onto the same GPIO_PIN the behavior of the hardware is undefined.

10.4.4 GCTL_DS

I/O Drive Strength Configuration Register

GCTL_DS			I/O Drive Strength Configuration Register				0xE005101C
b31	b30	b29	b28	b27	b26	b25	b24

GCTL_DS			I/O Drive Strength Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16
				I2CDS_G[1:0]		S1LDS_G[1:0]	
				R/W	R/W	R/W	R/W
				R	R	R	R
				2	2	2	2

GCTL_DS			I/O Drive Strength Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
S1DS_G[1:0]		S0DS_G[1:0]		PDS_G[1:0]		I2CDS[1:0]	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
2	2	2	2	2	2	2	2

GCTL_DS			I/O Drive Strength Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
S1LDS[1:0]		S1DS[1:0]		S0DS[1:0]		PDS[1:0]	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
2	2	2	2	2	2	2	2

This register defines pull-up and pull-down drive strength capability for I/O pins. Pins that are configured as GPIO use a different drive strength value from pins that have their 'normal' function.

Bit	Name	Description
19:18	**I2CDS_G[1:0]**	Drive strength for CHGDET pin, VIO5 power domain
17:16	**S1LDS_G[1:0]**	Drive strength for GPIOs, VIO4 power domain
15:14	**S1DS_G[1:0]**	Drive strength for GPIOs, VIO3 power domain
13:12	**S0DS_G[1:0]**	Drive strength for GPIOs, VIO2 power domain
11:10	**PDS_G[1:0]**	Drive strength for P-Port GPIOs, VIO1 power domain
9:8	**I2CDS[1:0]**	Drive strength for I^2C pins, VIO5 power domain 00 Slow speed 1-MHz gpio (push-pull) driver 01 I^2C fast-mode 1.5 to 3.6 V (open-drain only) 10 I^2C fast-mode 1.2 to 1.5 V, standard-mode 1.2 to 3.6 V (open-drain only) 11 I^2C fast-mode plus (open-drain only)
7:6	**S1LDS[1:0]**	Drive strength for I^2C pins, VIO5 power domain

continued on next page

10.4.4 GCTL_DS *(continued)*

5:4	**S1DS[1:0]**	Drive strength for GPIOs, VIO3 power domain
3:2	**S0DS[1:0]**	Drive strength for GPIOs, VIO2 power domain
1:0	**PDS[1:0]**	Drive strength for P-Port, VIO1 power domain 0 Quarter strength 1 Half strength 2 Three quarter strength 3 Full strength

10.4.5 GCTL_WPU_CFG

I/O Pull-Up Configuration Register

GCTL_WPU_CFG			I/O Pull-Up Configuration Register				0xE0051020
b63	b62	b61	b60	b59	b58	b57	b56
				WPU[59:56]			
				R/W	R/W	R/W	R/W
				R	R	R	R
				0	0	0	0

GCTL_WPU_CFG			I/O Pull-Up Configuration Register				
b55	b54	b53	b52	b51	b50	b49	b48
WPU[55:48]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_WPU_CFG			I/O Pull-Up Configuration Register				
b47	b46	b45	b44	b43	b42	b41	b40
WPU[47:40]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_WPU_CFG			I/O Pull-Up Configuration Register				
b39	b38	b37	b36	b35	b34	b33	b32
WPU[39:32]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_WPU_CFG			I/O Pull-Up Configuration Register				
b31	b30	b29	b28	b27	b26	b25	b24
WPU[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	0	0	0	0	0	0	0

GCTL_WPU_CFG			I/O Pull-Up Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16
WPU[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_WPU_CFG			I/O Pull-Up Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
WPU[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.4.5 GCTL_WPU_CFG *(continued)*

GCTL_WPU_CFG			I/O Pull-Up Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
WPU[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

In conjunction with the GCTL_DS register, this register determines whether a weak pull-up on each I/O pin should be engaged. Firmware should not enable WPU and WPD simultaneously on a given I/O. A weak pull-up or weak pull-down takes about 5 µs to be effective at the pads.

Bit	Name	Description
59:0	**WPU[59:0]**	When set, a weak pull-up is connected for the pin associated with the corresponding GPIO #.

10.4.6 GCTL_WPD_CFG

I/O Pull-Down Configuration Register

GCTL_WPD_CFG				I/O Pull-Down Configuration Register			0xE0051028
b63	b62	b61	b60	b59	b58	b57	b56
				WPD[59:56]			
				R/W	R/W	R/W	R/W
				R	R	R	R
				0	0	0	0

GCTL_WPD_CFG				I/O Pull-Down Configuration Register			
b55	b54	b53	b52	b51	b50	b49	b48
WPD[55:48]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_WPD_CFG				I/O Pull-Down Configuration Register			
b47	b46	b45	b44	b43	b42	b41	b40
WPD[47:40]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_WPD_CFG				I/O Pull-Down Configuration Register			
b39	b38	b37	b36	b35	b34	b33	b32
WPD[39:32]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_WPD_CFG				I/O Pull-Down Configuration Register			
b31	b30	b29	b28	b27	b26	b25	b24
WPD[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	0	0	0	0	0	0	0

GCTL_WPD_CFG				I/O Pull-Down Configuration Register			
b23	b22	b21	b20	b19	b18	b17	b16
WPD[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_WPD_CFG				I/O Pull-Down Configuration Register			
b15	b14	b13	b12	b11	b10	b9	b8
WPD[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.4.6 GCTL_WPD_CFG *(continued)*

GCTL_WPD_CFG			I/O Pull-Down Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
WPD[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

In conjunction with the GCTL_DS register, this register determines whether a weak pull-down on each I/O pin should be engaged. Firmware should not enable WPU and WPD simultaneously on a given I/O. A weak pull-up or weak pull-down takes about 5 µs to be effective at the pads.

Bit	Name	Description
59:0	**WPD[59:0]**	When set, a weak pull-down is connected for the pin associated with the corresponding GPIO #.

10.4.7 GCTL_IOPOWER

I/O Power Observability Register

GCTL_IOPOWER			I/O Power Observability Register				0xE0051030
b31	b30	b29	b28	b27	b26	b25	b24

GCTL_IOPOWER		I/O Power Observability Register					
b23	b22	b21	b20	b19	b18	b17	b16
		REG_BYPASS_EN	**REG_CARKIT_EN**	**REG_CARKIT_SEL[1:0]**		**USB_REGULATOR_TRIM[1:0]**	
		R/W	R/W	R/W	R/W	R/W	R/W
		R	R	R	R	R	R
		0	0	0	0	0	0

GCTL_IOPOWER			I/O Power Observability Register				
b15	b14	b13	b12	b11	b10	b9	b8
0	**0**	**USB_POWER_GOOD**	**VBUS_TH**	**PDS_G[1:0]**	**VBUS**	**USB25REG**	**USB33REG**
R	R	R	R	R	R	R	R
R	R	R/W	R/W	R/W	R/W	R/W	R/W
0	0	X	X	X	X	X	X

GCTL_IOPOWER			I/O Power Observability Register				
b7	b6	b5	b4	b3	b2	b1	b0
VIO5	**CVDDQ**	**EFVDDQ**	**VIO4**	**VIO3**	**VIO2**	**0**	**VIO1**
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
X	X	X	X	X	X	0	X

This register can be used to detect I/O Power status on various I/O domains

Bit	Name	Description
21	**REG_BYPASS_EN**	Controls the 3.3-V active regulator's bypass mode. This mode enables an external voltage to be applied to the internal 3.3-V node by bypassing the 3.3-V regulator. It allows the 2.5-V regulator to be tested and characterized across a different range of input supply voltages rather than from a single regulated 3.3-V supply. This bit has no function when the regulator is in standby mode. 0 3.3-V active regulator is not in bypass mode 1 3.3-V active regulator is in bypass mode
20	**REG_CARKIT_EN**	Enables the regulator to provide different output voltage from the 3.3-V regulator. Output value is selected by the reg_carkit_sel[1:0] bits. 1 Enables regulator to output different voltages for the carkit mode. NOT ALLOWED for chip-level carkit UART mode because the regulator 2.5-V supply output is disabled, which is not supported by the USB2.0 PHY. 0 Regulator operates normally.

continued on next page

10.4.7 GCTL_IOPOWER *(continued)*

19:18	**REG_CARKIT_SEL[1:0]**	Defines the output value of the 3.3-V regulator output. All values other than the default are for characterization and future use. This must not be used by software/firmware. 00 Default output of the 3.3-V regulator 01 Regulator output is nominal 3.0 V (experimental) 10 Regulator output is nominal 2.7 V (experimental) 11 Not allowed
17:16	**USB_REGULATOR_TRIM[1:0]**	Trim value for USB Main Regulator output voltage. See USB I/O Subsystem for more details.
13	**USB_POWER_GOOD**	Indicates that internal supplies of the regulator are stable. For debug purposes: 0 Internal power is not stable 1 Internal power is stable
12	**VBUS_TH**	Vbus level detected by regulator. The interrupt associated with this can be used to detect the over current condition when using an A-device. 0 Vbus not detected to be < 4.1 V 1 Vbus detected to be > 4.4 V
11	**VBAT**	Indicates VBAT voltage is present and within operating range. Internal regulator voltages may not yet be stable when this pin asserts.
10	**VBUS**	Indicates VBUS voltage is present and within operating range. Internal regulator voltages may not yet be stable when this pin asserts.
9	**USB25REG**	Indicates internal 2.5-V regulated voltage to USB2 PHY is present and stable.
8	**USB33REG**	Indicates internal 3.3-V regulator is present and stable.
7	**VIO5**	Indicates all I/O power domains for this block are powered and active. Any time needed for internal voltages to stabilize cells to become active has passed before this bit asserts.
6	**CVDDQ**	Indicates all I/O power domains for this block are powered and active. Any time needed for internal voltages to stabilize cells to become active has passed before this bit asserts. **Note** CVDDQ is required for chip operation (clock and reset). This bit will always be 1 when it can be accessed.
5	**EFVDDQ**	Indicates the eFuse programming voltage pin is supplied with either 1.2 V or 2.5 V. The presence of programming voltage at 2.5 V is not detectable. Programming will fail but reading succeeds when supplying this pin with 1.2 V.
4	**VIO4**	Indicates all I/O power domains for this block are powered and active. Any time needed for internal voltages to stabilize cells to become active has passed before this bit asserts.
3	**VIO3**	Indicates all I/O power domains for this block are powered and active. Any time needed for internal voltages to stabilize cells to become active has passed before this bit asserts.
2	**VIO2**	Indicates all I/O power domains for this block are powered and active. Any time needed for internal voltages to stabilize cells to become active has passed before this bit asserts.
0	**VIO1**	Indicates all I/O power domains for this block are powered and active. Any time needed for internal voltages to stabilize cells to become active has passed before this bit asserts.

10.4.8 GCTL_IOPOWER_INTR

I/O Power Change Interrupt Register

GCTL_IOPOWER_INTR			I/O Power Change Interrupt Register				0xE0051034
b31	b30	b29	b28	b27	b26	b25	b24

GCTL_IOPOWER_INTR			I/O Power Change Interrupt Register				
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_IOPOWER_INTR			I/O Power Change Interrupt Register				
b15	b14	b13	b12	b11	b10	b9	b8
		USB_POWER_GOOD	**VBUS_TH**	**VBAT**	**VBUS**	**USB25REG**	**USB33REG**
		R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
		W1S	W1S	W1S	W1S	W1S	W1S
		0	0	0	0	0	0

GCTL_IOPOWER_INTR			I/O Power Change Interrupt Register				
b7	b6	b5	b4	b3	b2	b1	b0
VIO5	**CVDDQ**	**EFVDDQ**	**VIO4**	**VIO3**	**VIO2**		**VIO1**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C		R/W1C
W1S	W1S	W1S	W1S	W1S	W1S		W1S
0	0	0	0	0	0		0

This register generates an interrupt when an interface power domain is powered up or down. Note that a power domain that is already up when the device resets (including wakeup from standby) also causes an interrupt in this register (which is not seen by the CPU until the corresponding mask bit is set).

Bit	Name	Description
13	**USB_POWER_GOOD**	Interrupt request. Must be cleared by firmware.
12	**VBUS_TH**	Interrupt request. Must be cleared by firmware.
11	**VBAT**	Interrupt request. Must be cleared by firmware.
10	**VBUS**	Interrupt request. Must be cleared by firmware.
9	**USB25REG**	Interrupt request. Must be cleared by firmware.
8	**USB33REG**	Interrupt request. Must be cleared by firmware.
7	**VIO5**	Interrupt request. Must be cleared by firmware.

continued on next page

10.4.8 GCTL_IOPOWER_INTR *(continued)*

6	**CVDDQ**	Interrupt request. Must be cleared by firmware. **Note** CVDDQ is required for chip operation (clock and reset). This interrupt will never trigger when it is observable.
5	**EFVDDQ**	Interrupt request. Must be cleared by firmware.
4	**VIO4**	Interrupt request. Must be cleared by firmware.
3	**VIO3**	Interrupt request. Must be cleared by firmware.
2	**VIO2**	Interrupt request. Must be cleared by firmware.
0	**VIO1**	Interrupt request. Must be cleared by firmware.

10.4.9 GCTL_IOPOWER_INTR_MASK

I/O Power Change Interrupt Mask Register

GCTL_IOPOWER_INTR_MASK			I/O Power Change Interrupt Mask Register				0xE0051038
b31	b30	b29	b28	b27	b26	b25	b24

GCTL_IOPOWER_INTR_MASK		I/O Power Change Interrupt Mask Register					
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_IOPOWER_INTR_MASK		I/O Power Change Interrupt Mask Register					
b15	b14	b13	b12	b11	b10	b9	b8
		USB_POWER_GOOD	**VBUS_TH**	**VBAT**	**VBUS**	**USB25REG**	**USB33REG**
		R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
		W1S	W1S	W1S	W1S	W1S	W1S
		0	0	0	0	0	0

GCTL_IOPOWER_INTR_MASK		I/O Power Change Interrupt Mask Register					
b7	b6	b5	b4	b3	b2	b1	b0
VIO5	**CVDDQ**	**EFVDDQ**	**VIO4**	**VIO3**	**VIO2**		**VIO1**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C		R/W1C
W1S	W1S	W1S	W1S	W1S	W1S		W1S
0	0	0	0	0	0		0

Set to 1 to activate interrupt. Affects reporting of interrupts only, not logging.

Bit	Name	Description
13	**USB_POWER_GOOD**	Set to 1 to report interrupt to CPU
12	**VBUS_TH**	Set to 1 to report interrupt to CPU
11	**VBAT**	Set to 1 to report interrupt to CPU
10	**VBUS**	Set to 1 to report interrupt to CPU
9	**USB25REG**	Set to 1 to report interrupt to CPU
8	**USB33REG**	Set to 1 to report interrupt to CPU
7	**VIO5**	Set to 1 to report interrupt to CPU
6	**CVDDQ**	Set to 1 to report interrupt to CPU. **Note** CVDDQ is required for chip operation (clock and reset). This interrupt will never trigger when it is observable.

continued on next page

10.4.9 GCTL_IOPOWER_INTR_MASK *(continued)*

5	**EFVDDQ**	Set to 1 to report interrupt to CPU
4	**VIO4**	Set to 1 to report interrupt to CPU
3	**VIO3**	Set to 1 to report interrupt to CPU
2	**VIO2**	Set to 1 to report interrupt to CPU
0	**VIO1**	Set to 1 to report interrupt to CPU

10.4.10 GCTL_SW_INT

Software Interrupt Register

GCTL_SW_INT			Software Interrupt Register				0xE005104C
b31	b30	b29	b28	b27	b26	b25	b24
SWINT	ARGUMENT[30:24]						
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_SW_INT			Software Interrupt Register				
b23	b22	b21	b20	b19	b18	b17	b16
ARGUMENT[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_SW_INT			Software Interrupt Register				
b15	b14	b13	b12	b11	b10	b9	b8
ARGUMENT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GCTL_SW_INT			Software Interrupt Register				
b7	b6	b5	b4	b3	b2	b1	b0
ARGUMENT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register is a software interrupt register with 31b argument.

Bit	Name	Description
31	**SWINT**	Software interrupt request. Must be set by issuer and cleared by ISR.
30:0	**ARGUMENT[30:0]**	31-bit argument that can be set by issuer, read by ISR.

10.4.11 GCTL_PLL_CFG

PLL Configuration Register

GCTL_PLL_CFG				PLL Configuration Register			0xE0052000
b31	b30	b29	b28	b27	b26	b25	b24

GCTL_PLL_CFG				PLL Configuration Register			
b23	b22	b21	b20	b19	b18	b17	b16
				PLL_LOCK	FSLC[2:0]		
				R	R	R	R
				R/W	R/W	R/W	R/W
				0	X	X	X

GCTL_PLL_CFG				PLL Configuration Register			
b15	b14	b13	b12	b11	b10	b9	b8
		CP_CFG[1:0]					REFDIV
		R/W	R/W				R/W
		R	R				R/W
		0	0				H

GCTL_PLL_CFG				PLL Configuration Register			
b7	b6	b5	b4	b3	b2	b1	b0
OUTDIV[1:0]		FBDIV[5:0]					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R/W	R/W	R/W	R/W	R/W	R/W
0	0	H	H	H	H	H	H

On power up, this value is initialized depending on the FSLC[1:0] pin values as follows:

0: 19.2-MHz input clock, FBDIV = 20, REFDIV0 = 0, OUTDIV(1:0) = 00, oscclk = 384 MHz
FX3 supports only 19.2 MHz.

Bit	Name	Description
19	**PLL_LOCK**	Asserted when PLL locks
18:16	**FSLC**	Value presented on FSLC[2:0] pins on the device that identify reference clock frequency/crystal provided. FSLC[2] selects between reference clock or crystal, FSLC[1:0] select reference clock frequency when FSLC[2] = 0.
13:12	**CP_CFG**	PLL charge pump configuration 0 2.5 µA 1 5 µA 2 7.5 µA 3 10 µA The charge pump bit setting varies depending on both the refclk frequency and the configuration divider bits.

continued on next page

10.4.11 GCTL_PLL_CFG *(continued)*

8	**REFDIV**	PLL input reference divider configuration. This field must be 0.
7:6	**OUTDIV**	PLL output divider configuration. This field must be 0.
5:0	**FBDIV**	PLL feedback divider configuration. This field must be 0x14.

10.4.12 GCTL_CPU_CLK_CFG

CPU and Bus Clock Configuration Register

GCTL_CPU_CLK_CFG			CPU and Bus Clock Configuration Register				0xE0052004
b31	b30	b29	b28	b27	b26	b25	b24

GCTL_CPU_CLK_CFG			CPU and Bus Clock Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_CPU_CLK_CFG			CPU and Bus Clock Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
MMID_DIV[3:0]				DMA_DIV[11:8]			
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1				1			

GCTL_CPU_CLK_CFG			CPU and Bus Clock Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
0	0	SRC[1:0]		CPU_DIV[3:0]			
R	R	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	2		1			

Bit	Name	Description
15:12	MMIO_DIV[3:0]	MMIO bus clock divider value. This determines how much to divide the CPU clock. The actual divider is DIV + 1. Zero (divide by 1) is illegal and results in undefined behavior. In other words, the range of divider values is 2 to16. The following must be true: (MMIO_DIV + 1) = N × (DMA_DIV +1).
11:8	DMA_DIV[3:0]	DMA bus clock divider value. This determines how much to divide the CPU clock. The actual divider is DIV + 1. Zero (divide by 1) is illegal and results in undefined behavior. In other words, the range of divider values is 2 to 16.
5:4	SRC[1:0]	Clock source select. This field selects between one of the following four prestage system clocks 00 sys16_clk (sys_clk_pll divided by 16) 01 sys4_clk (sys_clk_pll divided by 4) 10 sys2_clk (sys_clk_pll divided by 2) 11 sys_clk_pll On power up, the CPU clock will be PLL clock divided by 2, which is around 100 MHz. The expectation is that Boot ROM/Firmware changes this value to get to final CPU frequency.
3:0	CPU_DIV	CPU clock divider value. This determines how much to divide the source clock selected by the SRC field of this register. The actual divider is DIV + 1. Zero (divide by 1) is illegal and results in undefined behavior. In other words, the range of divider values is 2 to 16.

10.4.13 GCTL_UIB_CORE_CLK

UIB Clock Configuration Register

GCTL_UIB_CORE_CLK			UIB Clock Configuration Register				0xE0052008
b31	b30	b29	b28	b27	b26	b25	b24
CLK_EN							
R/W							
R							
0							

GCTL_UIB_CORE_CLK			UIB Clock Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_UIB_CORE_CLK			UIB Clock Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8

GCTL_UIB_CORE_CLK			UIB Clock Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
				PCLK_SRC[1:0]		EPMCLK_SRC[1:0]	
				R/W	R/W	R/W	R/W
				R	R	R	R
				2		2	

Bit	Name	Description
31	CLK_EN	Enable clock multiplexer
3:2	PCLK_SRC[1:0]	Clock source for SuperSpeed section of UIB block: 0 Not defined 1 USB3 PHY 125 MHz (spread spectrum clock) 2 Bus clock (typ 100 MHz) 3 Standby clock (typ 32 kHz) This field drives a simple clock mux; the actual presence and configuration of the clock inputs used is defined in the appropriate registers.
1:0	EPMCLK_SRC[1:0]	Clock source for EPM section of UIB block: 0 USB2 PHY 480 MHz divided by 4 (120 MHz) 1 USB3 PHY 125 MHz (spread spectrum clock) 2 Bus clock (typ 100 MHz) 3 Standby clock (typ 32 kHz) This field drives a simple clock mux; the actual presence and configuration of the clock inputs used is defined in the appropriate registers. **Note** In GTM test mode, make sure the USB2 PHY clock is running for at least 40 µs before selecting EPMCLK_SRC = 0.

10.4.14 GCTL_PIB_CORE_CLK

PIB Clock Configuration Register

GCTL_PIB_CORE_CLK			PIB Clock Configuration Register				0xE005200C
b31	b30	b29	b28	b27	b26	b25	b24
CLK_EN							
R/W							
R							
0							

GCTL_PIB_CORE_CLK			PIB Clock Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_PIB_CORE_CLK			PIB Clock Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
			SRC[1:0]		HALFDIV	DIV;9:8]	
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	0		

GCTL_PIB_CORE_CLK			PIB Clock Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
DIV[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1							

Bit	Name	Description
31	CLK_EN	Enable clock divider. Both the divider itself and its output are gated.
12:11	SRC[1:0]	Clock source select. This field selects between one of the following four prestage system clocks: 00 sys16_clk (sys_clk_pll divided by 16) 01 sys4_clk (sys_clk_pll divided by 4) 10 sys2_clk (sys_clk_pll divided by 2) 11 sys_clk_pll
10	HALFDIV	Nonintegral divider select. This field adds 0.5 to the divider value selected by the DIV field. This yields divider values from 2.5 to 256.5.
9:0	DIV[9:0]	Clock divider value. This determines how much to divide the PLL system clock. The actual divider is DIV + 1. Zero (divide by 1) is illegal and results in undefined behavior. In other words, the range of divider values is 2 to 1024.

10.4.15 GCTL_GPIO_FAST_CLK

GPIO Fast Clock Configuration Register

GCTL_GPIO_FAST_CLK			GPIO Fast Clock Configuration Register				0xE0052018
b31	b30	b29	b28	b27	b26	b25	b24
CLK_EN							
R/W							
R							
0							

GCTL_GPIO_FAST_CLK			GPIO Fast Clock Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_GPIO_FAST_CLK			GPIO Fast Clock Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
							SIMPLE1
							R/W
							R

GCTL_GPIO_FAST_CLK			GPIO Fast Clock Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
SIMPLE0	**SRC[1:0]**		**HALFDIV**	**DIV[3:0]**			
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
2	0	0	0	1			

Bit	Name	Description
31	**CLK_EN**	Enable clock divider. Both the divider itself and its output are gated.
8:7	**SIMPLE[1:0]**	Divider value for simple GPIOs relative to fast GPIO clock 0 Divide by 2 1 Divide by 4 2 Divide by 16 3 Divide by 64
6:5	**SRC[1:0]**	Clock source select. This field selects between one of the following four prestage system clocks: 00 sys16_clk (sys_clk_pll divided by 16) 01 sys4_clk (sys_clk_pll divided by 4) 10 sys2_clk (sys_clk_pll divided by 2) 11 sys_clk_pll
4	**HALFDIV**	Nonintegral divider select. This field adds 0.5 to the divider value selected by the DIV field. This yields divider values from 2.5 to 16.5. **Note** Do not use HALFDIV for FAST_CLK unless neither GPIO_SLOW_CLK or GPIO_SIMPLE_CLK is used. They will not function with HALFDIV = 1. **Note** Do not change HALFDIV after CLK_EN is set to 1 at least once, without first applying a hardware reset. It can be set together with CLK_EN in a single register write.

continued on next page

10.4.15 GCTL_GPIO_FAST_CLK *(continued)*

3:0	**DIV[3:0]**	Clock divider value. This determines how much to divide the PLL system clock. The actual divider is DIV + 1. Zero (divide by 1) is illegal and results in undefined behavior. In other words, the range of divider values is 2 to 16.

Note Any two writes to GCTL_GPIO_FAST_CLK and GPIO_SLOW_CLK must be spaced at least 35cy @ busclk apart. This holds for back-to-back writes to the same register as well as writes to both of these registers in either order.

10.4.16 GCTL_GPIO_SLOW_CLK

GPIO Slow Clock Configuration Register

GCTL_GPIO_SLOW_CLK			GPIO Slow Clock Configuration Register				0xE005201C
b31	b30	b29	b28	b27	b26	b25	b24
CLK_EN							
R/W							
R							
0							

GCTL_GPIO_SLOW_CLK			GPIO Slow Clock Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_GPIO_SLOW_CLK			GPIO Slow Clock Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8

GCTL_GPIO_SLOW_CLK		GPIO Slow Clock Configuration Register					
b7	b6	b5	b4	b3	b2	b1	b0
		DIV[5:0]					
		R/W	R/W	R/W	R/W	R/W	R/W
		R	R	R	R	R	R
		1					

Bit	Name	Description
31	**CLK_EN**	Enable clock divider. Both the divider itself and its output are gated.
5:0	**DIV[5:0]**	Clock divider value. This determines how much to divide the GPIO_FAST_CLK system clock. The actual divider is DIV + 1. Zero (divide by 1) is illegal and results in undefined behavior. In other words, the range of divider values is 2 to 64.

Note Any two writes to GCTL_GPIO_FAST_CLK and GPIO_SLOW_CLK must be spaced at least 35cy @ busclk apart. This holds for back-to-back writes to the same register as well as writes to both of these registers in either order.

10.4.17 GCTL_I2C_CORE_CLK

I²C Core Clock Configuration Register

GCTL_I2C_CORE_CLK			I²C Core Clock Configuration Register				0xE0052020
b31	b30	b29	b28	b27	b26	b25	b24
CLK_EN							
R/W							
R							
0							

GCTL_I2C_CORE_CLK			I²C Core Clock Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_I2C_CORE_CLK			I²C Core Clock Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
	SRC[1:0]		HALFDIV	DIV[11:8]			
	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	R	R	R	R	R	R	R
	0	0	0				

GCTL_I2C_CORE_CLK			I²C Core Clock Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
DIV[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1							

This register contains the divider for the I²C clock block. It must be set to a frequency that is 10x the bit rate on the interface (so 1 MHz, 4 MHz, 10 MHz, and so on).

Bit	Name	Description
31	CLK_EN	Enable clock divider. Both the divider itself and its output are gated.
14:13	SRC[1:0]	Clock source select. This field selects between one of the following four prestage system clocks: 00 sys16_clk (sys_clk_pll divided by 16) 01 sys4_clk (sys_clk_pll divided by 4) 10 sys2_clk (sys_clk_pll divided by 2) 11 sys_clk_pll
12	HALFDIV	Nonintegral divider select. This field adds 0.5 to the divider value selected by the DIV field. This yields divider values from 2.5 to 4096.5.
11:0	DIV[11:0]	Clock divider value. This determines how much to divide the PLL system clock. The actual divider is DIV + 1. Zero (divide by 1) is illegal and results in undefined behavior. In other words, the range of divider values is 2 to 4096.

10.4.18 GCTL_UART_CORE_CLK

UART Core Clock Configuration Register

GCTL_UART_CORE_CLK			UART Core Clock Configuration Register				0xE0052024
b31	b30	b29	b28	b27	b26	b25	b24
CLK_EN							
R/W							
R							
0							

GCTL_UART_CORE_CLK			UART Core Clock Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16
					SRC[1:0]		**HALFDIV**
					R/W	R/W	R/W
					R	R	R
					0	0	0

GCTL_UART_CORE_CLK			UART Core Clock Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
DIV[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GCTL_UART_CORE_CLK			UART Core Clock Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
DIV[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1							

This is the core clock provided to the UART peripheral. The UART uses 8x oversampling, so the resulting baud rate is 1/8th the frequency provided by this clock.

Bit	Name	Description
31	**CLK_EN**	Enable clock divider. Both the divider itself and its output are gated.
18:17	**SRC[1:0]**	Clock source select. This field selects between one of the following four prestage system clocks: 00 sys16_clk (sys_clk_pll divided by 16) 01 sys4_clk (sys_clk_pll divided by 4) 10 sys2_clk (sys_clk_pll divided by 2) 11 sys_clk_pll
16	**HALFDIV**	Nonintegral divider select. This field adds 0.5 to the divider value selected by the DIV field. This yields divider values from 2.5 to 65536.5.
15:0	**DIV[15:0]**	Clock divider value. This determines how much to divide the PLL system clock. The actual divider is DIV + 1. Zero (divide by 1) is illegal and results in undefined behavior. In other words, the range of divider values is 2 to 65536.

10.4.19 GCTL_SPI_CORE_CLK

SPI Core Clock Configuration Register

GCTL_SPI_CORE_CLK			SPI Core Clock Configuration Register				0xE005202C
b31	b30	b29	b28	b27	b26	b25	b24
CLK_EN							
R/W							
R							
0							

GCTL_SPI_CORE_CLK			SPI Core Clock Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16
					SRC[1:0]		**HALFDIV**
					R/W	R/W	R/W
					R	R	R
					0	0	0

GCTL_SPI_CORE_CLK			SPI Core Clock Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
DIV[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GCTL_SPI_CORE_CLK			SPI Core Clock Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
DIV[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1							

Bit	Name	Description
31	**CLK_EN**	Enable clock divider. Both the divider itself and its output are gated.
18:17	**SRC[1:0]**	Clock source select. This field selects between one of the following four prestage system clocks: 00 sys16_clk (sys_clk_pll divided by 16) 01 sys4_clk (sys_clk_pll divided by 4) 10 sys2_clk (sys_clk_pll divided by 2) 11 sys_clk_pll
16	**HALFDIV**	Nonintegral divider select. This field adds 0.5 to the divider value selected by the DIV field. This yields divider values from 2.5 to 65536.5.
15:0	**DIV[15:0]**	Clock divider value. This determines how much to divide the PLL system clock. The actual divider is DIV + 1. Zero (divide by 1) is illegal and results in undefined behavior. In other words, the range of divider values is 2 to 65536.

10.4.20 GCTL_I2S_CORE_CLK

I2S Core Clock Configuration Register

GCTL_I2S_CORE_CLK			I2S Core Clock Configuration Register				0xE0052034
b31	b30	b29	b28	b27	b26	b25	b24
CLK_EN	**MCLK_IN**						
R/W	R/W						
R	R						
0	0						

GCTL_I2S_CORE_CLK			I2S Core Clock Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16
						SRC[1:0]	
						R/W	R/W
						R	R
						0	0

GCTL_I2S_CORE_CLK			I2S Core Clock Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
HALFDIV	**DIV[14:8]**						
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0							

GCTL_I2S_CORE_CLK			I2S Core Clock Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
DIV[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1							

Bit	Name	Description
31	**CLK_EN**	Enable clock divider. Both the divider itself and its output are gated.
30	**MCLK_IN**	0 I2S_MCLK is an output, generated from this divider 1 I2S_MCLK is taken as the input to this divider instead of the PLL clock **Note** Never change this value after CLK_EN is set without first issuing a reset to the device.
17:16	**SRC[1:0]**	Clock source select. This field selects between one of the following four prestage system clocks: 00 sys16_clk (sys_clk_pll divided by 16) 01 sys4_clk (sys_clk_pll divided by 4) 10 sys2_clk (sys_clk_pll divided by 2) 11 sys_clk_pll
15	**HALFDIV**	Nonintegral divider select. This field adds 0.5 to the divider value selected by the DIV field. This yields divider values from 2.5 to 32768.5.
14:0	**DIV[15:0]**	Clock divider value. This determines how much to divide the PLL system clock. The actual divider is DIV + 1. Zero (divide by 1) is illegal and results in undefined behavior. In other words, the range of divider values is 2 to 32768.

10.5 Global Controller Always On Registers

10.5.1 GCTL_WAKEUP_EN

Wakeup Enable Register

GCTL_WAKEUP_EN			Wakeup Enable Register				0xE0050004
b31	b30	b29	b28	b27	b26	b25	b24

GCTL_WAKEUP_EN			Wakeup Enable Register				
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_WAKEUP_EN		Wakeup Enable Register					
b15	b14	b13	b12	b11	b10	b9	b8
		EN_WATCHDOG2	**EN_WATCHDOG1**	**EN_UIB_VBUS**	**EN_UIB_SSRX**	**EN_UIB_OTGID**	**EN_UIB_DM**
		R/W	R/W	R/W	R/W	R	R/W
		R	R	R	R	R/RW	R
		0	0	0	0	0	0

GCTL_WAKEUP_EN			Wakeup Enable Register				
b7	b6	b5	b4	b3	b2	b1	b0
EN_UIB_DP	**EN_UART_CTS**	**EN_GPIO[44]**	**EN_GPIO[47]**	**EN_GPIO[34]**	**EN_PIB_CLK**	**EN_PIB_CMD**	**EN_PIB_CTRL0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register indicates which wakeup sources are enabled to wake up the system from suspend/standby modes. It is in the always-on power domain. Unless otherwise noted, wakeup sources work from both standby and suspend modes. The wakeup detectors are active across all power modes and can also be used as interrupt sources in active/idle mode. If wakeup pins are defined as outputs (in GPIO or PIB), wakeup functionality is not guaranteed to work. GCTL_CONTROL.WAKEUP_CPU_INT must be set before entering suspend/standby to enable wakeup.

Bit	Name	Description
13	**EN_WATCHDOG2**	Enables wakeup when WatchDog #2 generates an interrupt.
12	**EN_WATCHDOG1**	Enables wakeup when WatchDog #1 generates an interrupt.
11	**EN_UIB_VBUS**	Enables wakeup when VBUS is asserted. Works in either standby or suspend modes.
10	**EN_UIB_SSRX**	Enables wakeup when an LFPS signal appears on the USB3 SSRX pins. This wakeup source does not work from standby mode.
9	**EN_UIB_OTGID**	Enables wakeup when an impedance change is detected on the USB2 OTGID pin (USB ATTACH event). This wakeup source does not work from standby mode. This wakeup source works from suspend only when an external 32-kHz clock source is present. *continued on next page*

10.5.1 GCTL_WAKEUP_EN *(continued)*

8	**EN_UIB_DM**	Enables wakeup when the USB2 D+ line transitions to from 0 to 1 or from 1 to 0 (CONNECT in host mode). This wakeup source does not work from standby mode.
7	**EN_UIB_DP**	Enables wakeup when the USB2 D+ line transitions to from 1 to 0 (USB RESUME) or from 0 to 1 (CONNECT in host mode). This wakeup source does not work from standby mode.
6	**EN_UART_CTS**	Enables wakeup from the UART_CTS pin. Wakeup occurs when the level set in POL_UART_CTS is detected.
5	**EN_GPIO[44]**	Enables wakeup from the GPIO[44] pin (card insertion). Wakeup occurs when the level set in POL_GPIO[44] is detected.
4	**EN_GPIO[47]**	Enables wakeup from the GPIO[47] pin (SD1). Wakeup occurs when the level set in POL_GPIO[47] is detected.
3	**EN_GPIO[34]**	Enables wakeup from the GPIO[34] pin (SD0). Wakeup occurs when the level set in POL_GPIO[34] is detected.
2	**EN_PIB_CLK**	Enables wakeup from the PIB_CLK pin. Wakeup occurs on any change on this pin after wakeup.
1	**EN_PIB_CMD**	Enable wakeup from a CMD5 on the PIB_CMD pin. This wakeup source does not work from standby mode. This wakeup source works from suspend only when an external 32-kHz clock source is present.
0	**EN_PIB_CTRL0**	Enables wakeup from the PIB CTL0 pin (CE# typically). Wakeup occurs on any change on this pin after wakeup.

10.5.2 GCTL_WAKEUP_POLARITY

Wakeup Signal Polarity Register

GCTL_WAKEUP_POLARITY			Wakeup Signal Polarity Register				0xE0050008
b31	b30	b29	b28	b27	b26	b25	b24

GCTL_WAKEUP_POLARITY			Wakeup Signal Polarity Register				
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_WAKEUP_POLARITY			Wakeup Signal Polarity Register				
b15	b14	b13	b12	b11	b10	b9	b8
				POL_UIB_VBUS			**POL_UIB_DM**
				R/W			R/W
				R			R
				1			0

GCTL_WAKEUP_POLARITY			Wakeup Signal Polarity Register				
b7	b6	b5	b4	b3	b2	b1	b0
POL_UIB_DP	**POL_UART_CTS**	**POL_GPIO[44]**	**POL_GPIO[47]**	**POL_GPIO[34]**			
R/W	R/W	R/W	R/W	R/W			
R	R	R	R	R			
0	0	0	0	0			

This register indicates polarity for wakeup detect signals. It is in the always on power domain.

Bit	Name	Description
11	**POL_UIB_VBUS**	Polarity of the VBUS signal: 0 Wakeup when LOW 1 Wakeup when HIGH
8	**POL_UIB_DM**	Polarity of the USB D– signal: 0 Wakeup when LOW 1 Wakeup when HIGH
7	**POL_UIB_DP**	Polarity of the USB D+ signal: 0 Wakeup when LOW 1 Wakeup when HIGH
6	**POL_UART_CTS**	Polarity of the UART_CTS signal: 0 Wakeup when LOW 1 Wakeup when HIGH
5	**POL_GPIO[44]**	Polarity of the GPIO[44] signal: 0 Wakeup when LOW 1 Wakeup when HIGH

continued on next page

10.5.2 GCTL_WAKEUP_POLARITY *(continued)*

4	**POL_GPIO[47]**	Polarity of the GPIO[47] signal: 0 Wakeup when LOW (Applicable for SDIO interrupt wakeup) 1 Wakeup when HIGH (Not required for SDIO interrupt wakeup)
3	**POL_GPIO[34]**	Polarity of the GPIO[34] signal: 0 Wakeup when LOW (Applicable for SDIO interrupt wakeup) 1 Wakeup when HIGH (Not required for SDIO interrupt wakeup)

10.5.3 GCTL_WAKEUP_EVENT

Wakeup Event Register

GCTL_WAKEUP_EVENT				Wakeup Event Register			0xE005000C
b31	b30	b29	b28	b27	b26	b25	b24

GCTL_WAKEUP_EVENT				Wakeup Event Register			
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_WAKEUP_EVENT				Wakeup Event Register			
b15	b14	b13	b12	b11	b10	b9	b8
		EV_WATCHDOG2	EV_WATCHDOG1	EV_UIB_VBUS	EV_UIB_SSRX	EV_UIB_OTGID	EV_UIB_DM
		R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
		R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
		0	0	0	0	0	0

GCTL_WAKEUP_EVENT				Wakeup Event Register			
b7	b6	b5	b4	b3	b2	b1	b0
EV_UIB_DP	EV_UART_CTS	EV_GPIO[44]	EV_GPIO[47]	EV_GPIO[34]	EV_PIB_CLK	EV_PIB_CMD	EV_PIB_CTRL0
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

This register indicates which wakeup source caused the system wakeup. Multiple bits may be set (although this is unlikely). After power-on or reset, these bits are zero. This register is in the always-on power domain. The wakeup detectors are active across all power modes and can also be used as interrupt sources in active/idle mode.

Bit	Name	Description
13	EV_WATCHDOG2	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.
12	EV_WATCHDOG1	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.
11	EV_UIB_VBUS	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.
10	EV_UIB_SSRX	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.
9	EV_UIB_OTGID	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.
8	EV_UIB_DM	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.

continued on next page

10.5.3 GCTL_WAKEUP_EVENT *(continued)*

7	**EV_UIB_DP**	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.
6	**EV_UART_CTS**	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.
5	**EV_GPIO[44]**	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.
4	**EV_GPIO[47]**	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.
3	**EV_GPIO[34]**	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.
2	**EV_PIB_CLK**	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.
1	**EV_PIB_CMD**	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.
0	**EV_PIB_CTRL0**	Indicates that this wakeup source was the reason for system wakeup from standby/suspend mode. See GCTL_WAKEUP_EN for more information.

10.5.4 GCTL_FREEZE

I/O Freeze Control Register

GCTL_FREEZE			I/O Freeze Control Register				0xE0050010
b31	b30	b29	b28	b27	b26	b25	b24

GCTL_FREEZE			I/O Freeze Control Register				
b23	b22	b21	b20	b19	b18	b17	b16

GCTL_FREEZE			I/O Freeze Control Register				
b15	b14	b13	b12	b11	b10	b9	b8

GCTL_FREEZE			I/O Freeze Control Register				
b7	b6	b5	b4	b3	b2	b1	b0
VIO4IO_FRZ[1:0]		VIO3IO_FRZ[1:0]		VIO2IO_FRZ[1:0]		VIO1IO_FRZ[1:0]	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register indicates the state of the I/Os in each power domain when GCTL_CONTROL.FREEZE_IO is asserted. It is in the always-on power domain.

Bit	Name	Description
7:6	**VIO4IO_FRZ[1:0]**	Frozen state of I/Os in the VIO4 domain.
5:4	**VIO3IO_FRZ[1:0]**	Frozen state of I/Os in the VIO3 domain.
3:2	**VIO2IO_FRZ[1:0]**	Frozen state of I/Os in the VIO2 domain.
1:0	**VIO1IO_FRZ[1:0]**	Frozen state of I/Os in the VIO1 domain 0 Sample and hold state 1 High impedance 2 Drive 0 at full strength (outputs only) 3 Drive 1 at full strength (outputs only) Note that states 2 and 3 only override the output value driven to a fixed value, but do not change the drive mode of a pin from off to on. In other words, pins that are currently inputs remain inputs and are not forced to drive.

10.5.5 GCTL_WATCHDOG_CS

Watchdog Timers Command and Control Register

GCTL_WATCHDOG_CS			Watchdog Timers Command and Control Register				0xE0050014
b31	b30	b29	b28	b27	b26	b25	b24
BACKUP_CLK	**BACKUP_DIVIDER[14:8]**						
R/W1S	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0							

GCTL_WATCHDOG_CS			Watchdog Timers Command and Control Register				
b23	b22	b21	b20	b19	b18	b17	b16
BACKUP_DIVIDER[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1							

GCTL_WATCHDOG_CS			Watchdog Timers Command and Control Register				
b15	b14	b13	b12	b11	b10	b9	b8
BITS1[4:0]					**INTR1**	**MODE1[1:0]**	
R/W	R/W	R/W	R/W	R/W	R/W1C	R/W	R/W
R	R	R	R	R	R/W1S	R	R
0	0	0	0	0	0	3	

GCTL_WATCHDOG_CS			Watchdog Timers Command and Control Register				
b7	b6	b5	b4	b3	b2	b1	b0
BITS0[4:0]					**INTR0**	**MODE0[1:0]**	
R/W	R/W	R/W	R/W	R/W	R/W1C	R/W	R/W
R	R	R	R	R	R/W1S	R	R
0	0	0	0	0	0	3	

This register controls two watchdog timers. Each timer can be used in free-running, interrupt, or reset mode and has a selectable number of significant bits. The frequency is fixed at 32768 Hz, and counters count down only. Firmware can protect this register against unwanted writes (see GCTL_WAKEUP_EN).

Bit	Name	Description
31	**BACKUP_CLK**	Switches the watchdog clocks from the 32-kHz clock input to a 'backup' 32-kHz clock derived from the main reference clock using BACKUP_DIVIDER. This field is sticky and cannot change after it is set to 1. This bit can only be cleared with the RESET# pin or POR.
30:16	**BACKUP_DIVIDER[14:0]**	Divider used to generate a 'backup' 32-kHz clock. This is relevant for systems where no 32-kHz clock is present as an input signal. The external reference clock (OSCCLK) is divided by (BACKUP_DIVIDER + 1). In other words, a value of 1 means divided by 2. The behavior of the divider is undefined when this value is 0. This field must not change after the BACKUP_CLK bit is set.
15:11	**BITS1[4:0]**	Number of least significant bits to be used when checking for counter limit (useful only for MODE = 1, 2).
10	**INTR1**	Interrupt signal (Mode 1 only). This bit indicates when the WDOG timer has issued an Interrupt to the CPU while the system is powered-up. Refer to the GCTL_WAKEUP_EVENT register when the WDOG timer is used to wake up the system from a power-down state.

continued on next page

10.5.5 GCTL_WATCHDOG_CS *(continued)*

9:8	**MODE1[1:0]**	Counter mode: 0 Free-running mode, counter wraps around after 32 bits. 1 Interrupt mode, interrupt when COUNTER & ~((~0)<<BITS) = 0. 2 Reset mode, full chip RESET when COUNTER & ~((~0)<<BITS) = 0. 3 Disable - counter does not run
7:3	**BITS0[4:0]**	Number of least significant bits to be used when checking for counter limit (useful only for MODE = 1, 2)
2	**INTR0**	Interrupt signal (Mode 1 only). This bit indicates when the WDOG timer has issued an interrupt to the CPU while the system is powered up. Refer to the GCTL_WAKEUP_EVENT register when the WDOG timer is used to wake up the system from a power-down state.
1:0	**MODE0**	Counter mode: 0 Free-running mode, counter wraps around after 32 bits. 1 Interrupt mode, interrupt when COUNTER & ~((~0)<<BITS) = 0. 2 Reset mode, full chip RESET when COUNTER & ~((~0)<<BITS) = 0. 3 Disable - counter does not run

10.5.6 GCTL_WATCHDOG_TIMER0

Watchdog Timer Value 0 Register

GCTL_WATCHDOG_TIMER0			Watchdog Timer Value 0 Register				0xE0050018
b31	b30	b29	b28	b27	b26	b25	b24
COUNTER[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GCTL_WATCHDOG_TIMER0			Watchdog Timer Value 0 Register				
b23	b22	b21	b20	b19	b18	b17	b16
COUNTER[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GCTL_WATCHDOG_TIMER0			Watchdog Timer Value 0 Register				
b15	b14	b13	b12	b11	b10	b9	b8
COUNTER[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GCTL_WATCHDOG_TIMER0			Watchdog Timer Value 0 Register				
b7	b6	b5	b4	b3	b2	b1	b0
COUNTER[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFFFFFF							

This register holds the watchdog timer/counter. It counts down and generates an interrupt or reset when reaching low-limit (see GCTL_WATCHDOG_CS). The counter is free running and wraps around after 32 bits. In watchdog mode this value must be reloaded periodically to avoid full chip reset. Firmware can protect this register against unwanted writes (see GCTL_WAKEUP_EN).

Bit	Name	Description
31:0	COUNTER[31:0]	Current counter value. Note that, because of synchronization, it may take up to 100 µs before a value written to this register can be read back. Earlier reads will return the previous value. The CPU should wait 100 µs between successive writes to this register. The CPU should not write the same value to this register successively; instead, it should alter the value. If the CPU wants to write value x, every interval, it must write x, x – 1, x, x – 1, … in successive intervals.

10.5.7 GCTL_WATCHDOG_TIMER1

Watchdog Timer Value 1 Register

GCTL_WATCHDOG_TIMER1			Watchdog Timer Value 1 Register				0xE005001C
b31	b30	b29	b28	b27	b26	b25	b24
COUNTER[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GCTL_WATCHDOG_TIMER1			Watchdog Timer Value 1 Register				
b23	b22	b21	b20	b19	b18	b17	b16
COUNTER[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GCTL_WATCHDOG_TIMER1			Watchdog Timer Value 1 Register				
b15	b14	b13	b12	b11	b10	b9	b8
COUNTER[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GCTL_WATCHDOG_TIMER1			Watchdog Timer Value 1 Register				
b7	b6	b5	b4	b3	b2	b1	b0
COUNTER[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFFFFFF							

This register holds the watchdog timer/counter. It counts down and generates an interrupt or reset when reaching low-limit (see GCTL_WATCHDOG_CS). The counter is free running and wraps around after 32 bits. In watchdog mode this value must be reloaded periodically to avoid full chip reset. Firmware can protect this register against unwanted writes (see GCTL_WAKEUP_EN).

Bit	Name	Description
31:0	COUNTER[31:0]	Current counter value. Note that, because of synchronization, it may take up to 100 µs before a value written to this register can be read back. Earlier reads will return the previous value. The CPU should wait 100 µs between successive writes to this register. The CPU should not write the same value to this register successively; instead, it should alter the value. If the CPU wants to write value x, every interval, it must write x, x – 1, x, x – 1, … in successive intervals.

10.6 PIB Registers

10.6.1 PIB_CONFIG

PIB Configuration Register

PIB_CONFIG			PIB Configuration Register				0xE0010000
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE	**MMIO_ENABLE**	**PP_CFGMODE**	**PCFG**				
R/W	R/W0C	R/W	R/W				
R	R	R/W	R				
1	1	1	0				

PIB_CONFIG			PIB Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

PIB_CONFIG			PIB Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
DEVICE_ID[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

PIB_CONFIG			PIB Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
DEVICE_ID[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register contains general configuration parameters. Default values upon reset for these fields depend on the extern configuration pins and on the contents on the eFuses, as described in the boot chapter.

Bit	Name	Description
31	**ENABLE**	Enables the entire P-Port IP.
30	**MMIO_ENABLE**	Enables the PP_MMIO protocol for accessing individual MMIO registers externally over P-Port. After it is disabled, this function cannot be re-enabled. 0 Disabled 1 Enabled
29	**PP_CFGMODE**	Variable indicating initialization mode to Application processor (PP_CONFIG.CFGMODE). Cleared by firmware after P-Port is properly initialized and ready to transact.

continued on next page

10.6.1 PIB_CONFIG *(continued)*

28	**PCFG**	P-Port configuration 0 GPIF-II mode 1 Reserved The Boot ROM interprets the PMODE pins through GPIO and chooses the appropriate GPIF and P-Port configuration.
15:8	**DEVICE_ID[7:0]**	Provides a device ID. This field is visible in PP_INIT registers. This must be provided by Boot ROM. To prevent spoofing, this register is not writable when GCTL_CONTROL.BOOTROM_EN = 0.

10.6.2 PIB_INTR

PIB Interrupt Request Register

PIB_INTR			PIB Interrupt Request Register				0xE0010004
b31	b30	b29	b28	b27	b26	b25	b24
GPIF_ERR		**PIB_ERR**					
R/W1C		R/W1C					
W1S		W1S					
0		0					

PIB_INTR			PIB Interrupt Request Register				
b23	b22	b21	b20	b19	b18	b17	b16

PIB_INTR			PIB Interrupt Request Register				
b15	b14	b13	b12	b11	b10	b9	b8
					RD_THRESHOLD	**WR_THRESHOLD**	**CONFIG_CHANGE**
					R/W1C	R/W1C	R/W1C
					W1S	W1S	W1S
					0	0	0

PIB_INTR			PIB Interrupt Request Register				
b7	b6	b5	b4	b3	b2	b1	b0
CLOCK_LOST	**DLL_LOST_LOCK**	**DLL_LOCKED**	**GPIF_INTERRUPT**			**WR_MB_FULL**	**RD_MB_EMPTY**
R/W1C	R/W1C	R/W1C	R			R/W1C	R/W1C
W1S	W1S	W1S	W			R/W1S	R/W1S
0	0	0	0			0	0

The PIB_INTR and PIB_INTR_MASK registers control the interrupt behavior of the P-Port to the FX3 CPU. PIB_INTR indicates interrupt cause, and PIB_INTR_MASK masks the causes that may assert INTR. These interrupts represent the P-Port specific interrupts. The DMA adapter for P-Port has a number of socket-related interrupt causes that are outlined in the DMA chapter.

Bit	Name	Description
31	**GPIF_ERR**	An error occurred in the GPIF. Firmware clears this bit after handling the error. The error code is indicated in PIB_ERROR.GPIF_ERR_CODE.
29	**PIB_ERR**	The socket-based link controller encountered an error and needs attention. Firmware clears this bit after handling the error. The error code is indicated in PIB_ERROR.PIB_ERR_CODE.
10	**RD_THRESHOLD**	Indicates that AP has written to PP_RD_THRESHOLD register.
9	**WR_THRESHOLD**	Indicates that AP has written to PP_WR_THRESHOLD register.
8	**CONFIG_CHANGE**	AP has written a new value into PP_CONFIG.
7	**CLOCK_LOST**	PIB_CLK is no longer present. See PIB_CLOCK_DETECT for more details.

continued on next page

10.6.2 PIB_INTR *(continued)*

6	**DLL_LOST_LOCK**	DLL has lost phase lock. Interrupt clears after writing 1.
5	**DLL_LOCKED**	DLL has achieved phase lock. Interrupt clears after writing 1.
4	**GPIF_INTERRUPT**	Indicates that the interrupt is from the GPIF II block. Consult the GPIF_STATUS register.
1	**WR_MB_FULL**	Indicates that a message is present in the WR_MAILBOX and must be read. This bit sets when AP writes a message in the WR_MAILBOX. It must be cleared by firmware, which causes PP_EVENT.WR_MB_EMPTY to assert.
0	**RD_MB_EMPTY**	Indicates that the RD_MAILBOX is empty and a new message can be written. This bit sets when AP writes PP_EVENT.RD_MB_FULL = 0. It must be cleared by firmware.

10.6.3 PIB_INTR_MASK

PIB Interrupt Mask Register

PIB_INTR_MASK			PIB Interrupt Mask Register				0xE0010008
b31	b30	b29	b28	b27	b26	b25	b24
GPIF_ERR		**PIB_ERR**					
R/W		R/W					
R		R					
0		0					

PIB_INTR_MASK			PIB Interrupt Mask Register				
b23	b22	b21	b20	b19	b18	b17	b16

PIB_INTR_MASK			PIB Interrupt Mask Register				
b15	b14	b13	b12	b11	b10	b9	b8
					RD_THRESHOLD	**WR_THRESHOLD**	**CONFIG_CHANGE**
					R/W	R/W	R/W
					R	R	R
					0	0	0

PIB_INTR_MASK			PIB Interrupt Mask Register				
b7	b6	b5	b4	b3	b2	b1	b0
CLOCK_LOST	**DLL_LOST_LOCK**	**DLL_LOCKED**	**GPIF_INTERRUPT**			**WR_MB_FULL**	**RD_MB_EMPTY**
R/W	R/W	R/W	R/W			R/W	R/W
R	R	R	R			R	R
0	0	0	0			0	0

The PIB_INTR and PIB_INTR_MASK registers control the interrupt behavior of the P-Port to the FX3 CPU. PIB_INTR indicates interrupt cause, and PIB_INTR_MASK masks the causes that may assert INTR. These interrupts represent the P-Port specific interrupts. The DMA adapter for P-Port has a number of socket-related interrupt causes that are outlined in the DMA chapter.

Bit	Name	Description
31	**GPIF_ERR**	Mask for corresponding interrupt in PIB_INTR
29	**PIB_ERR**	Mask for corresponding interrupt in PIB_INTR
10	**RD_THRESHOLD**	Mask for corresponding interrupt in PIB_INTR
9	**WR_THRESHOLD**	Mask for corresponding interrupt in PIB_INTR
8	**CONFIG_CHANGE**	Mask for corresponding interrupt in PIB_INTR
7	**CLOCK_LOST**	Mask for corresponding interrupt in PIB_INTR
6	**DLL_LOST_LOCK**	Mask for corresponding interrupt in PIB_INTR

continued on next page

10.6.3 PIB_INTR_MASK *(continued)*

5	**DLL_LOCKED**	Mask for corresponding interrupt in PIB_INTR
4	**GPIF_INTERRUPT**	Mask for corresponding interrupt in PIB_INTR
1	**WR_MB_FULL**	Mask for corresponding interrupt in PIB_INTR
0	**RD_MB_EMPTY**	Mask for corresponding interrupt in PIB_INTR

10.6.4 PIB_CLOCK_DETECT

PIB Clock Detector Configuration Register

PIB_CLOCK_DETECT				PIB Clock Detector Configuration Register			0xE001000C
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE	CLOCK_PRESENT						
R/W	R						
R	R/W						
0	0						

PIB_CLOCK_DETECT				PIB Clock Detector Configuration Register			
b23	b22	b21	b20	b19	b18	b17	b16
				INTF_CYCLES[3:0]			
				R/W	R/W	R/W	R/W
				R	R	R	R
				0	0	0	0

PIB_CLOCK_DETECT				PIB Clock Detector Configuration Register			
b15	b14	b13	b12	b11	b10	b9	b8
BUS_CYCLES[15:8]							
R	R	R	R	R	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

PIB_CLOCK_DETECT				PIB Clock Detector Configuration Register			
b7	b6	b5	b4	b3	b2	b1	b0
BUS_CYCLES[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register enables and configures the clock input presence detector for PIB. This detector is independent from the wakeup source detector that triggers on any edge. This detector can establish whether a valid clock is present in active mode and is coupled to the PIB_INTR.CLOCK_LOST interrupt.

Bit	Name	Description
31	**ENABLE**	Enables detector
30	**CLOCK_PRESENT**	Indicates latest clock presence measurement
19:16	**INTF_CYCLES[3:0]**	Minimum number of positive edges required on PIBCLK pin to declare clock presence during each measurement period.
15:0	**BUS_CYCLES[15:0]**	Number of busclk cycles for each measurement period.

10.6.5 PIB_RD_MAILBOX

Read (Egress) Mailbox Register

PIB_RD_MAILBOX			Read (Egress) Mailbox Register				0xE0010010
b63	b62	b61	b60	b59	b58	b57	b56
PP_RD_MAILBOX[63:56]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

PIB_RD_MAILBOX			Read (Egress) Mailbox Register				
b55	b54	b53	b52	b51	b50	b49	b48
PP_RD_MAILBOX[55:48]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

PIB_RD_MAILBOX			Read (Egress) Mailbox Register				
b47	b46	b45	b44	b43	b42	b41	b40
PP_RD_MAILBOX[47:40]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

PIB_RD_MAILBOX			Read (Egress) Mailbox Register				
b39	b38	b37	b36	b35	b34	b33	b32
PP_RD_MAILBOX[39:32]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

PIB_RD_MAILBOX			Read (Egress) Mailbox Register				
b31	b30	b29	b28	b27	b26	b25	b24
PP_RD_MAILBOX[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

PIB_RD_MAILBOX			Read (Egress) Mailbox Register				
b23	b22	b21	b20	b19	b18	b17	b16
PP_RD_MAILBOX[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

PIB_RD_MAILBOX			Read (Egress) Mailbox Register				
b15	b14	b13	b12	b11	b10	b9	b8
PP_RD_MAILBOX[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.6.5 PIB_RD_MAILBOX *(continued)*

PIB_RD_MAILBOX			Read (Egress) Mailbox Register				
b7	b6	b5	b4	b3	b2	b1	b0
PP_RD_MAILBOX[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register mirrors the P-Port RD mailbox register and is used to send short messages to the AP using MMIO and interrupt behavior. Writing to the high word of this register sets the PP_EVENT.RD_MB_FULL flag. The procedure to use is described as part of the socket-based link controller section.

Bit	Name	Description
63:0	**PP_RD_MAILBOX[63:0]**	Mailbox message from FX3 to the application processor

10.6.6 PIB_WR_MAILBOX

Write (Ingress) Mailbox Register

PIB_WR_MAILBOX			Write (Ingress) Mailbox Register				0xE0010018
b63	b62	b61	b60	b59	b58	b57	b56
PP_WR_MAILBOX[63:56]							
R	R	R	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PIB_WR_MAILBOX			**Write (Ingress) Mailbox Register**				
b55	b54	b53	b52	b51	b50	b49	b48
PP_WR_MAILBOX[55:48]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PIB_WR_MAILBOX			**Write (Ingress) Mailbox Register**				
b47	b46	b45	b44	b43	b42	b41	b40
PP_WR_MAILBOX[47:40]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PIB_WR_MAILBOX			**Write (Ingress) Mailbox Register**				
b39	b38	b37	b36	b35	b34	b33	b32
PP_WR_MAILBOX[39:32]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PIB_WR_MAILBOX			**Write (Ingress) Mailbox Register**				
b31	b30	b29	b28	b27	b26	b25	b24
PP_WR_MAILBOX[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	0	0	0	0	0	0	0
PIB_WR_MAILBOX			**Write (Ingress) Mailbox Register**				
b23	b22	b21	b20	b19	b18	b17	b16
PP_WR_MAILBOX[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PIB_WR_MAILBOX			**Write (Ingress) Mailbox Register**				
b15	b14	b13	b12	b11	b10	b9	b8
PP_WR_MAILBOX[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.6.6 PIB_RD_MAILBOX *(continued)*

PIB_WR_MAILBOX			Write (Ingress) Mailbox Register				
b7	b6	b5	b4	b3	b2	b1	b0
PP_WR_MAILBOX[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register mirrors the P-Port WR mailbox register and is used to receive short messages from the AP using MMIO and interrupt behavior. Reading from these registers has no side effect. The procedure to use is described as part of the socket-based link controller section.

Bit	Name	Description
63:0	**PP_WR_MAILBOX[63:0]**	Mailbox message from the application processor to FX3

10.6.7 PIB_ERROR

PIB Error Indicator Register

PIB_ERROR	PIB Error Indicator Register						0xE0010020
b31	b30	b29	b28	b27	b26	b25	b24

PIB_ERROR	PIB Error Indicator Register						
b23	b22	b21	b20	b19	b18	b17	b16

PIB_ERROR	PIB Error Indicator Register						
b15	b14	b13	b12	b11	b10	b9	b8
0	**GPIF_ERR_CODE[4:0]**						
R	R	R	R	R	R		
R	R/W	R/W	R/W	R/W	R/W		
0	0	0	0	0	0		

PIB_ERROR	PIB Error Indicator Register						
b7	b6	b5	b4	b3	b2	b1	b0
		PIB_ERR_CODE[5:0]					
		R	R	R	R	R	R
		R/W	R/W	R/W	R/W	R/W	R/W
		0	0	0	0	0	0

This register indicates the error codes associated with PIB_INTR.PIB_ERR, MMC_ERR, and GPIF_ERR. The different values for these error codes are documented in the P-Port BROS document. This register is also visible to the AP as PP_ERROR.

Bit	Name	Description
14:10	**GPIF_ERR_CODE[4:0]**	Error code for the first error code after ERROR = 1 Error codes are specified in detail the USB30PIB BROS.

Error Code	Default	Description
IIN_ADDR_OVER_WRITE	1	Attempt to push to the active address thread which is not dma_ready
EG_ADDR_NOT_VALID	2	Attempt to push to the active address thread which is not dma_ready
DMA_DATA_RD_ERROR	3	Attempt to push to the active address thread which is not dma_ready
DMA_DATA_WR_ERROR	4	Attempt to push to the active address thread which is not dma_ready
DMA_ADDR_RD_ERROR	5	Attempt to push to the active address thread which is not dma_ready
DMA_ADDR_WR_ERROR	6	Attempt to push to the active address thread which is not dma_ready
SERIAL_PARITY_ERROR	7	Received data in serial mode has parity error
INVALID_STATE_ERROR	8	State machine has transitioned to an invalid state

continued on next page

10.6.7 PIB_ERROR *(continued)*

5:0 **PIB_ERR_CODE[5:0]** The socket-based link controller encountered an error and needs attention. Error codes are further described in the BROS. Corresponds to interrupt bit PIB_ERROR.

Error Code	Default	Description
TH0_DIR_ERROR	1	Write being done to a read socket or read being done to a write socket
TH1_DIR_ERROR	2	Write being done to a read socket or read being done to a write socket
TH2_DIR_ERROR	3	Write being done to a read socket or read being done to a write socket
TH3_DIR_ERROR	4	Write being done to a read socket or read being done to a write socket
TH0_WR_OVERFLOW	5	Write exceeds the space available in the buffer
TH1_WR_OVERFLOW	6	Write exceeds the space available in the buffer
TH2_WR_OVERFLOW	7	Write exceeds the space available in the buffer
TH3_WR_OVERFLOW	8	Write exceeds the space available in the buffer
TH0_RD_UNDERRUN	9	Reads exceeds the byte count of the buffer
TH1_RD_UNDERRUN	10	Reads exceeds the byte count of the buffer
TH2_RD_UNDERRUN	11	Reads exceeds the byte count of the buffer
TH3_RD_UNDERRUN	12	Reads exceeds the byte count of the buffer
TH0_SCK_ACTIVE	0x12	Socket has gone inactive within a DMA transfer
TH0_ADAP_OVERFLOW	0x13	Adapter unable to service write request though buffer is available
TH0_ADAP_UNDERFLOW	0x14	Adapter unable to service read request though buffer is available
TH0_READ_FORCE_END	0x15	A read socket is wrapped up
TH0_READ_BURST_ERR	0x16	A read socket with burstsize > 0 is switched before the 8-byte boundary
TH1_SCK_ACTIVE	0x1A	A socket has gone inactive within a DMA Transfer
TH1_ADAP_OVERFLOW	0x1B	Adapter unable to service write request though buffer is available
TH1_ADAP_UNDERFLOW	0x1C	Adapter unable to service read request though buffer is available
TH1_READ_FORCE_END	0x1D	A read socket is wrapped up
TH1_READ_BURST_ERR	0x1E	A read socket with burstsize > 0 is switched before the 8-byte boundary
TH2_ADAP_OVERFLOW	0x22	A socket has gone inactive within a DMA transfer
TH2_ADAP_UNDERFLOW	0x23	Adapter unable to service write request though buffer is available
TH2_ADAP_UNDERFLOW	0x24	Adapter unable to service read request though buffer is available
TH2_READ_FORCE_END	0x25	A read socket is wrapped up
TH2_READ_BURST_ERR	0x26	A read socket with burstsize > 0 is switched before 8-byte boundary
TH3_SCK_ACTIVE	0x2A	A socket has gone inactive within a DMA transfer
TH3_ADAP_OVERFLOW	0x2B	Adapter unable to service write request though buffer is available
TH3_ADAP_UNDERFLOW	0x2C	Adapter unable to service read request though buffer is available
TH3_READ_FORCE_END	0x2D	A read socket is wrapped up
TH3_READ_BURST_ERR	0x2E	A read socket with burstsize > 0 is switched before the 8-byte boundary

10.6.8 PIB_EOP_EOT

PIB EOP/EOT Configuration Register

PIB EOP/EOT			Configuration Register				0xE0010024
b31	b30	b29	b28	b27	b26	b25	b24
PIB_EOP_EOT_CFG[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	1	1	1	1	1	1	1
PIB EOP/EOT			**Configuration Register**				
b23	b22	b21	b20	b19	b18	b17	b16
PIB_EOP_EOT_CFG[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	1	1	1	1	1	1	1
PIB EOP/EOT			**Configuration Register**				
b15	b14	b13	b12	b11	b10	b9	b8
PIB_EOP_EOT_CFG[15:18]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	1	1	1	1	1	1	1
PIB EOP/EOT			**Configuration Register**				
b7	b6	b5	b4	b3	b2	b1	b0
PIB_EOP_EOT_CFG[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	1	1	1	1	1	1	1

This register specifies how EOP is set in descriptors of ingress P-port sockets and how EOP is interpreted for egress P-port sockets.

Bit	Name	Description
31:0	**PIB_EOP_EOT_CFG[31:0]**	This register specifies how EOP bits are set or interpreted for ingress and egress sockets, respectively. 0 Stream mode behavior 1 Packet mode behavior

10.6.9 PIB_DLL_CTRL

DLL Configuration Register

PIB_DLL_CTRL			DLL Configuration Register				0xE0010028
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE_RESET_ON_ERR	**DLL_RESET_N**						
R/W	R/W						
R	R						
0	0						

PIB_DLL_CTRL			DLL Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

PIB_DLL_CTRL			DLL Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8

PIB_DLL_CTRL			DLL Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
					DLL_STAT	**HIGH_FREQ**	**ENABLE**
					R	R/W	R/W
					W	R	R
					0	0	0

This register configures the DLL phases and enables the DLL.

Bit	Name	Description
31	**ENABLE_RESET_ON_ERR**	0 Hardware does not reset the DLL when DLL code overrun/underrun occurs 1 Hardware resets the DLL when DLL code overrun/underrun occurs
30	**DLL_RESET_N**	Resets the DLL 0 DLL is reset 1 DLL is not reset

continued on next page

10.6.9 PIB_DLL_CTRL *(continued)*

		011	Phase Comparator DFT #1. The used dll_out signals are available for monitoring on TDO pin in clock_observation_mode
		100	Scan test: monitor test_so. This mode is not cannot be selected through register control.
		101	Phase Comparator DFT #2: The used dll_out signals are available for monitoring on TDO pin in clock_observation_mode
		110	Error code reset: The error_code[1:0] output of the dll is available for monitoring on the designated GPIO in clock_observation_mode
		111	This setting is used either for fault grading or monotonicity testing, based on the settings of the “mode” input. DLL_MODE =0: Fault grading. The DLL's phase comparator drives dllrdy, which is available for monitoring on the designated GPIO in clock_observation_mode DLL_MODE =1: Monotonicity test. The used dll_out signals are available for monitoring on TDO pin in clock_observation_mode
2	**DLL_STAT**	0	DLL is not in phase lock
		1	DLL has achieved phase lock
1	**HIGH_FREQ**	0	23 to 80 MHz
		1	70 to 230 MHz
0	**ENABLE**	Drives the DLLEN input	
		0	DLL is disabled (internally power gated)
		1	DLL is enabled

10.6.10 PIB_WR_THRESHOLD

Write Threshold Register

PIB_WR_THRESHOLD			Write Threshold Register				0xE001002C
b31	b30	b29	b28	b27	b26	b25	b24
PIB_WR_THRESHOLD			**Write Threshold Register**				
b23	b22	b21	b20	b19	b18	b17	b16
PIB_WR_THRESHOLD			**Write Threshold Register**				
b15	b14	b13	b12	b11	b10	b9	b8
VALUE16[15:18]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PIB_WR_THRESHOLD			**Write Threshold Register**				
b7	b6	b5	b4	b3	b2	b1	b0
VALUE16[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

This register specifies the write buffer free event threshold; the CPU writes a 16-bit value that is visible using the PP_WR_THRESHOLD register. When AP writes the PP_WR_THRESHOLD register, the PIB_WR_THRESHOLD register is updated with the WR_THRESHOLD interrupt.

Bit	Name	Description
15:0	**VALUE16[15:0]**	The CPU writes to this field during initialization. The value written to this register is made available by hardware in PP_WR_THRESHOLD register. When AP writes to the PP_WR_THRESHOLD register this register is updated with the new value and WR_THRESHOLD interrupt is provided.

10.6.11 PIB_RD_THRESHOLD

Read Threshold Register

PIB_RD_THRESHOLD			Read Threshold Register				0xE0010030
b31	b30	b29	b28	b27	b26	b25	b24

PIB_RD_THRESHOLD			Read Threshold Register				
b23	b22	b21	b20	b19	b18	b17	b16

PIB_RD_THRESHOLD			Read Threshold Register				
b15	b14	b13	b12	b11	b10	b9	b8
VALUE16[15:18]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PIB_RD_THRESHOLD			Read Threshold Register				
b7	b6	b5	b4	b3	b2	b1	b0
VALUE16[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Read buffer data available to the event threshold as a number of bytes available. The device side must have the intelligence to activate the interrupt either when this threshold is exceeded or when it sees it useful (end of packet).

Bit	Name	Description
15:0	**VALUE16[15:0]**	The CPU writes to this field during initialization. The value written to this register is made available by hardware in the PP_RD_THRESHOLD register. When AP writes to the PP_RD_THRESHOLD register, this register is updated with the new value and the RD_THRESHOLD interrupt is provided.

10.6.12 PIB_ID

Block Identification and Version Number Register

PIB_ID	Block Identification and Version Number Register						0xE0017F00
b31	b30	b29	b28	b27	b26	b25	b24
BLOCK_VERSION[15:18]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R

PIB_ID	Block Identification and Version Number Register						
b23	b22	b21	b20	b19	b18	b17	b16
BLOCK_VERSION[7:0]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0x0001							

PIB_ID	Block Identification and Version Number Register						
b15	b14	b13	b12	b11	b10	b9	b8
BLOCK_ID[15:18]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

PIB_ID	Block Identification and Version Number Register						
b7	b6	b5	b4	b3	b2	b1	b0
BLOCK_ID[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0x0001							

Every IP block implements a few MMIO registers at offset 0 in its MMIO space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31:16	**BLOCK_VERSION[15:0]**	Version number for the IP
15:0	**BLOCK_ID[15:0]**	A unique number identifying the IP in the memory space

10.6.13 PIB_POWER

Power, Clock, and Reset Control Register

PIB_POWER			Power, Clock, and Reset Control Register				0xE0017F04
b31	b30	b29	b28	b27	b26	b25	b24
RESETN							
R/W							
R							
0							

PIB_RD_THRESHOLD			Read Threshold Register				
b23	b22	b21	b20	b19	b18	b17	b16

PIB_RD_THRESHOLD			Read Threshold Register				
b15	b14	b13	b12	b11	b10	b9	b8

PIB_RD_THRESHOLD			Read Threshold Register				
b7	b6	b5	b4	b3	b2	b1	b0
							ACTIVE
							R
							W
							0

Every IP block implements a few MMIO registers at offset 0 in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31	**RESETN**	Active low reset signal for all logic in the block. Note that reset is active on all flops in the block when either system reset is asserted (RESET# pin or SYSTEM_POWER.RESETN is asserted) or this signal is active. After setting this bit to 1, firmware polls and waits for the ‘active’ bit to assert. Reading ‘1’ from ‘resetn’ does not indicate the block is out of reset. This may take some time depending on initialization tasks and clock frequencies.
0	**ACTIVE**	For blocks that must perform initialization after reset before becoming operational, this signal will remain deasserted until initialization is complete. In other words, reading ACTIVE = 1 indicates that the block is initialized and ready for operation.

10.7 GPIF Registers

10.7.1 GPIF_CONFIG

GPIF Configuration Register

GPIF_CONFIG				GPIF Configuration Register			0xE0014000
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE	**PP_MODE**						
R/W	R/W						
R	R						
0	0						

GPIF_CONFIG				GPIF Configuration Register			
b23	b22	b21	b20	b19	b18	b17	b16
				A7OVERRIDE			
				R/W			
				R			
				0			

GPIF_CONFIG				GPIF Configuration Register			
b15	b14	b13	b12	b11	b10	b9	b8
THREAD_IN_STATE		**DATA_COMP_TOGGLE**	**CTRL_COMP_TOGGLE**	**ADDR_COMP_TOGGLE**	**ENDIAN**		**SYNC**
R/W		R/W	R/W	R/W	R/W		R/W
R		R	R	R	R		R
0		0	0	0	0		0

GPIF_CONFIG				GPIF Configuration Register			
b7	b6	b5	b4	b3	b2	b1	b0
DOUT_POP_EN	**DDR_MODE**	**CLK_OUT**	**CLK_SOURCE**	**CLK_INVERT**	**DATA_COMP_ENABLE**	**ADDR_COMP_ENABLE**	**CTRL_COMP_ENABLE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	1	0	0	0	0	0

This register provides static configuration information for GPIF.

Bit	Name	Description
31	**ENABLE**	Enables entire GPIF block.
30	**PP_MODE**	Main operating mode of PP registers 0 Master mode or simple slave mode - PP* registers are not honored 1 P-Port mode - PP* registers are used to setup transfers
19	**A7OVERRIDE**	Overrides the use of AIN[7] to enable selection of register versus DMA access on different pins in PP_MODE = 1. If A7OVERRIDE = 1, register accesses are determined by beta (pp_access) instead.
15	**THREAD_IN_STATE**	0 Normal operation 1 The thread number for an operation comes from the state description rather than the THREAD_CONFIG register (see GPIF_Modes for more information)

continued on next page

10.7.1 GPIF_CONFIG *(continued)*

13	**DATA_COMP_TOGGLE**	0	Comparator outputs true when bits match a target value.
		1	Comparator outputs true when any of the unmasked bits change value.
12	**CTRL_COMP_TOGGLE**	0	Comparator outputs true when bits match a target value.
		1	Comparator outputs true when any of the unmasked bits change value.
11	**ADDR_COMP_TOGGLE**	0	Comparator outputs true when bits match a target value.
		1	Comparator outputs true when any of the unmasked bits change value.
10	**ENDIAN**	Endianness of interface when PP_MODE==0	
		0	Little Endian
		1	Big Endian
8	**SYNC**	0	Operate in asynchronous mode.
		1	Operate in synchronous mode.
		These mode have different clocking structures. GPIF_CONFIG.SYNC should be set to indicate synchronous or Asynchronous MODE before programming GPIF_BUS_CONFIG.DLE* and GPIF_BUS_CONFIG.ALE*	
7	**DOUT_POP_EN**	0	rq_pop is a separate beta
		1	Use update_dout (alpha) to also trigger rq_pop (which is normally beta)
6	**DDR_MODE**	0	Select 1X clock as the core clock
		1	Select 2X clock as the core clock
5	**CLK_OUT**	Indicates whether to drive CLK pin with GPIF clock. No effect when CLK_SOURCE = 0 or PMMC mode.	
		0	Do not output clock on CLK pin (typical of async mode)
		1	Output clock on CLK pin (typical of sync master mode)
4	**CLK_SOURCE**	Indicates whether clock is sourced from GCTL or pin. Has no effect in PMMC mode.	
		0	External clock input on CLK pin
		1	Internal clock generated from GCTL
3	**CLK_INVERT**	0	Normal clock polarity (clock on positive edge)
		1	Inverted clock polarity (clock on negative edge)
2	**DATA_COMP_ENABLE**	0	Disable the data comparator
		1	Enable the data comparator
1	**ADDR_COMP_ENABLE**	0	Disable the address comparator
		1	Enable the address comparator
0	**CTRL_COMP_ENABLE**	0	Disable the control comparator
		1	Enable the control comparator

10.7.2 GPIF_BUS_CONFIG

Bus Configuration Register

GPIF_BUS_CONFIG			Bus Configuration Register				0xE0014004
b31	b30	b29	b28	b27	b26	b25	b24
FIO1_CONF[3:0]				FIO0_CONF[3:0]			
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_BUS_CONFIG			Bus Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16
CE_CLKSTOP	INT_CTRL		DRQ_ASSERT_MODE	DRQ_MODE[1:0]		ALE_PRESENT	CNTR_PRESENT
R/W	R/W		R/W	R/W	R/W	R/W	R/W
R	R		R	R	R	R	R
0	0		0	0	0	0	0

GPIF_BUS_CONFIG			Bus Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
FIO1_PRESENT	FIO0_PRESENT	DRQ_PRESENT	OE_PRESENT	DLE_PRESENT	WE_PRESENT	CE_PRESENT	ADR_CTRL[3]
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_BUS_CONFIG			Bus Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
ADR_CTRL[2:0]			MUX_MODE	BUS_WIDTH[1:0]		PIN_COUNT[1:0]	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register specifies how the pins on the GPIF interface are distributed over data, address and control buses and which special (hard-wired) functions are present on the control bus.

Bit	Name	Description
31:28	**FIO1_CONF[3:0]**	Designates control to be used to enable output drivers of FIO1 (CTRL[8]) 0 to 3 Use the alpha 4-7 to switch FIO1 direction. 8 to 11 Use beta 0-3.
27:24	**FIO0_CONF[3:0]**	Designates control to be used to enable output drivers of FIO0 (CTRL[7]) 0 to 3 Use the alpha 4 to 7 to switch FIO0 direction. 8 to 11 Use beta 0-3.
23	**CE_CLKSTOP**	0 Normal operation 1 CTRL[0] is CE; it should be used to clock gate most of GPIF when deasserted
22	**INT_CTRL**	0 Normal operation of INT pin 1 Override INT pin and connect to CTRL[15]
20	**DRQ_ASSERT_MODE**	0 Do nothing 1 Assert DRQ on rising edge of DMA_READY. Typical case, DRQ_MODE = 2, this bit 1.

continued on next page

10.7.2 GPIF_BUS_CONFIG *(continued)*

19:18	**DRQ_MODE[1:0]**	0 Assert DRQ on deassertion of DACK 1 Assert DRQ on assertion of DACK 2 Deassert DRQ on deassertion of DACK 3 Deassert DRQ on assertion of DACK
17	**ALE_PRESENT**	CTRL[10] is ALE used to latch address from the DQ lines.
16	**CNTR_PRESENT**	CTRL[9] is connected to the selected control counter bit instead of a control signal
15	**FIO1_PRESENT**	CTRL[8] is to be treated as I/O that is driven out when the alpha specified in FIO1_CONF is asserted (ignore CTRL_BUS_DIRECTION)
14	**FIO0_PRESENT**	CTRL[7] is to be treated as I/O that is driven out when the alpha specified in FIO0_CONF is asserted (ignore CTRL_BUS_DIRECTION)
13	**DRQ_PRESENT**	CTRL[4] is directly influenced by CTRL[3]/DACK as defined by DRQ_MODE This setting also overrides the CTRL_BUS_SELECT selection for this pin, in favor of betas 'assert drq' and 'deassert_drq'
12	**OE_PRESENT**	CTRL[2] is OE and should be used to tristate DQ lines. If WE_PRESENT = 1 also, then OE will take precedence over WE. In other words, when WE is asserted, then output drivers are off, regardless of value of OE input.
11	**DLE_PRESENT**	CTRL[1] is DLE and should be used to latch data from DQ lines Can be used together with WE_PRESENT
10	**WE_PRESENT**	CTRL[1] is WE and should be used to disable DQ drivers Can be used together with DLE_PRESENT
9	**CE_PRESENT**	CTRL[0] is CE and should be used to disable DQ drivers
8:5	**ADR_CTRL[3:0]**	Number of control lines overridden by address lines. Control signals CTRL[15] to CTRL[16-ADR_CTRL] are not connected to pins. Instead those pins are designated as address signals. Which address signals depends on the other mode fields above. In other words: if ADR_CTRL = 0 all CTRL lines are connected to pins, if ADR_CTRL = 1, CTRL[15] is not connected and so on.
4	**MUX_MODE**	0 Address and data are separate lines. 1 Address is sampled from DQ, time multiplexed with data.
3:2	**BUS_WIDTH[1:0]**	0 DQ is 8b wide 1 DQ is 16b wide 2 DQ is 24b wide 3 DQ is 32b wide
1:0	**PIN_COUNT[1:0]**	Number of pins allocated to GPIF interface. Needs to be consistent with GCTL_IOMATRIX: 0 47-pin interface 1 43-pin interface 2 35-pin interface 3 31-pin interface

10.7.3 GPIF_BUS_CONFIG2

Bus Configuration Register #2

GPIF_BUS_CONFIG2			Bus Configuration Register #2				0xE0014008
b31	b30	b29	b28	b27	b26	b25	b24
			STATE7[4:0]				
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	0	0	0
GPIF_BUS_CONFIG2			Bus Configuration Register #2				
b23	b22	b21	b20	b19	b18	b17	b16
			STATE6[4:0]				
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	0	0	0
GPIF_BUS_CONFIG2			Bus Configuration Register #2				
b15	b14	b13	b12	b11	b10	b9	b8
			STATE5[4:0]				
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	0	0	0
GPIF_BUS_CONFIG2			Bus Configuration Register #2				
b7	b6	b5	b4	b3	b2	b1	b0
					STATE_FROM_CTRL[2:0]		
					R/W	R/W	R/W
					R	R	R
					0	0	0

This register configures state number overrides directly using control signals CTRL[i] in the state number.

Bit	Name	Description
28:24	**STATE7[4:0]**	Lambda number to be used for state number bit 7
20:16	**STATE6[4:0]**	Lambda number to be used for state number bit 6
12:8	**STATE5[4:0]**	Lambda number to be used for state number bit 5
2:0	**STATE_FROM_CTRL[1:0]**	0 Normal operation 1,2,3 STATE7 indicates Lambda number to be used for state number bit 7 2,3 STATE6 indicates Lambda number to be used for state number bit 6 3 STATE5 indicates Lambda number to be used for state number bit 5

10.7.4 GPIF_AD_CONFIG

Address/Data Configuration Register

GPIF_AD_CONFIG				Address/Data Configuration Register			0xE001400C
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_AD_CONFIG				Address/Data Configuration Register			
b23	b22	b21	b20	b19	b18	b17	b16
				DATA_THREAD[1:0]		ADDRESS_THREAD[1:0]	
				R/W	R/W	R/W	R/W
				R	R	R	R
				0	0	0	0

GPIF_AD_CONFIG				Address/Data Configuration Register			
b15	b14	b13	b12	b11	b10	b9	b8
						AIN_DATA	DOUT_SELECT
						R/W	R/W
						R	R
						0	0

GPIF_AD_CONFIG				Address/Data Configuration Register			
b7	b6	b5	b4	b3	b2	b1	b0
AOUT_SELECT[1:0]		AIN_SELECT[1:0]		A_OEN_CFG[1:0]		DQ_OEN_CFG[1:0]	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Configuration for DQ lines into DQ, Address, control buses and serial modes.

Bit	Name	Description
19:18	**DATA_THREAD[1:0]**	Thread number to be used for data; only relevant when AIN_DATA = 1 When this register is used to select the thread it must have been initialized by firmware. When both are used, ADDRESS_THREAD must be different from DATA_THREAD.
17:16	**ADDRESS_THREAD[1:0]**	Thread number to be used for addresses; only relevant when AIN_SELECT! = 0 When this register is used to select the thread it must have been initialized by firmware. When both are used, ADDRESS_THREAD must be different from DATA_THREAD.
9	**AIN_DATA**	This field determines which thread number to use for data accesses: 0 Specified by A1:A0. 1 Specified by DATA_THREAD If AIN_SELECT=0 this field also determines the thread number for which to change the active socket on awq_push: 0 The active socket of thread A1:A0 is changed (3 bits only) 1 The active socket of thread DATA_THREAD is changed (full 5 bits)
8	**DOUT_SELECT**	0 Connect DOUT to the socket pointed to by AIN_DATA or EGRESS_DATA register (as determined by beta 'register_access') 1 Connect DOUT to DATA_COUNTER

continued on next page

10.7.4 GPIF_AD_CONFIG *(continued)*

7:6	**AOUT_SELECT[1:0]**	0	Connect AOUT to ADDR_COUNTER
		1	Connect AOUT to EGRESS_ADDRESS register
		2	Connect AOUT to the socket pointed to by ADDRESS_THREAD register
5:4	**AIN_SELECT[1:0]**	0	Connect AIN to the active socket number of thread specified by AIN_DATA
		1	Connect AIN to INGRESS_ADDRESS register
		2	Connect AIN to the socket pointed to by ADDRESS_THREAD register
3:2	**A_OEN_CFG[1:0]**	Address lines (if there are any) are	
		0	Always outputs
		1	Always inputs
		2	Controlled by the beta: “a_oen”
		3	Reserved
1:0	**DQ_OEN_CFG[1:0]**	Data lines are	
		0	Always outputs
		1	Always inputs
		2	Direction controlled by the alpha: “dq_oen”
		3	Reserved
		Note that CTRL[2] can be OE, controlling the data output drivers directly, overriding what's specified here (this field should be set to 0 if that is used)	

10.7.5 GPIF_STATUS

GPIF Status Register

GPIF_STATUS	GPIF Status Register						0xE0014010
b31	b30	b29	b28	b27	b26	b25	b24
INTERRUPT_STATE[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

GPIF_STATUS	GPIF Status Register						
b23	b22	b21	b20	b19	b18	b17	b16
IN_DATA_VALID[3:0]				EG_DATA_EMPTY[3:0]			
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0xF			

GPIF_STATUS	GPIF Status Register						
b15	b14	b13	b12	b11	b10	b9	b8
					WAVEFORM_BUSY	CTRL_COMP_HIT	DATA_COMP_HIT
					R	R	R
					R/W	R/W	R/W
					0	0	0

GPIF_STATUS	GPIF Status Register						
b7	b6	b5	b4	b3	b2	b1	b0
ADDR_COMP_HIT	CTRL_COUNT_HIT	DATA_COUNT_HIT	ADDR_COUNT_HIT	CRC_ERROR	SWITCH_TIMEOUT	GPIF_INTR	GPIF_DONE
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Live status bits. Most of these bits can trigger an interrupt on their rising edge (see GPIF_INTR).

Bit	Name	Description
31:24	**INTERRUPT_STATE[7:0]**	State that raised the interrupt through GPIF_INTR. Note that these bits do not have individual interrupt and mask bits in GPIF_INTR and GPIF_MASK.
23:20	**IN_DATA_VALID[3:0]**	Indicates corresponding INGRESS_DATA register is full.
19:16	**EG_DATA_EMPTY[3:0]**	Indicates corresponding EGRESS_DATA register is empty
10	**WAVEFORM_BUSY**	CPU tried to access waveform memory without clearing WAVEFORM_VALID
9	**CTRL_COMP_HIT**	Control comparator hits
8	**DATA_COMP_HIT**	Data comparator hits
7	**ADDR_COMP_HIT**	Address comparator hits
6	**CTRL_COUNT_HIT**	Control counter is at limit

continued on next page

10.7.5 GPIF_STATUS *(continued)*

5	**DATA_COUNT_HIT**	Data counter is at limit
4	**ADDR_COUNT_HIT**	Address counter is at limit
3	**CRC_ERROR**	Indicates that an incorrect CRC was received
2	**SWITCH_TIMEOUT**	Indicates that the SWITCH_TIMEOUT was reached (see WAVEFORM_SWITCH). This bit clears when a new WAVEFORM_SWTICH is initiated.
1	**GPIF_INTR**	Indicates that GPIF state machine has raised an interrupt.
0	**GPIF_DONE**	1 GPIF has reached the DONE state. Nonsticky.

10.7.6 GPIF_INTR

GPIF Interrupt Request Register

GPIF_INTR	GPIF Interrupt Request Register						0xE0014014
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_INTR	GPIF Interrupt Request Register						
b23	b22	b21	b20	b19	b18	b17	b16
IN_DATA_VALID[3:0]				EG_DATA_EMPTY[3:0]			
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

GPIF_INTR	GPIF Interrupt Request Register						
b15	b14	b13	b12	b11	b10	b9	b8
					WAVEFORM_BUSY	CTRL_COMP_HIT	DATA_COMP_HIT
					R/W1C	R/W1C	R/W1C
					R/W1S	R/W1S	R/W1S
					0	0	0

GPIF_INTR	GPIF Interrupt Request Register						
b7	b6	b5	b4	b3	b2	b1	b0
ADDR_COMP_HIT	CTRL_COUNT_HIT	DATA_COUNT_HIT	ADDR_COUNT_HIT	CRC_ERROR	SWITCH_TIMEOUT	GPIF_INTR	GPIF_DONE
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

Interrupt request bits. Bits assert when corresponding bit in GPIF_STATUS asserts and must be cleared by software.

Bit	Name	Description
23:20	IN_DATA_VALID[3:0]	Interrupt request corresponding to same bit in GPIF_STATUS
19:16	EG_DATA_EMPTY[3:0]	Interrupt request corresponding to same bit in GPIF_STATUS
10	WAVEFORM_BUSY	Interrupt request corresponding to same bit in GPIF_STATUS
9	CTRL_COMP_HIT	Interrupt request corresponding to same bit in GPIF_STATUS
8	DATA_COMP_HIT	Interrupt request corresponding to same bit in GPIF_STATUS
7	ADDR_COMP_HIT	Interrupt request corresponding to same bit in GPIF_STATUS
6	CTRL_COUNT_HIT	Interrupt request corresponding to same bit in GPIF_STATUS
5	DATA_COUNT_HIT	Interrupt request corresponding to same bit in GPIF_STATUS
4	ADDR_COUNT_HIT	Interrupt request corresponding to same bit in GPIF_STATUS

continued on next page

10.7.6 GPIF_INTR *(continued)*

3	**CRC_ERROR**	Interrupt request corresponding to same bit in GPIF_STATUS
2	**SWITCH_TIMEOUT**	Interrupt request corresponding to same bit in GPIF_STATUS
1	**GPIF_INTR**	Interrupt request corresponding to same bit in GPIF_STATUS
0	**GPIF_DONE**	Interrupt request corresponding to same bit in GPIF_STATUS

10.7.7 GPIF_INTR_MASK

GPIF Interrupt Mask Register

GPIF_INTR_MASK	GPIF Interrupt Mask Register						0xE0014018
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_INTR_MASK	GPIF Interrupt Mask Register						
b23	b22	b21	b20	b19	b18	b17	b16
IN_DATA_VALID[3:0]				EG_DATA_EMPTY[3:0]			
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_INTR_MASK	GPIF Interrupt Mask Register						
b15	b14	b13	b12	b11	b10	b9	b8
					WAVEFORM_BUSY	CTRL_COMP_HIT	DATA_COMP_HIT
					R/W	R/W	R/W
					R	R	R
					0	0	0

GPIF_INTR_MASK	GPIF Interrupt Mask Register						
b7	b6	b5	b4	b3	b2	b1	b0
ADDR_COMP_HIT	CTRL_COUNT_HIT	DATA_COUNT_HIT	ADDR_COUNT_HIT	CRC_ERROR	SWITCH_TIMEOUT	GPIF_INTR	GPIF_DONE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Interrupt mask. Report interrupt to firmware only when corresponding bit in this register is set.

Bit	Name	Description
23:20	**IN_DATA_VALID[3:0]**	Mask bit that controls reporting of corresponding bit in GPIF_INTR
19:16	**EG_DATA_EMPTY[3:0]**	Mask bit that controls reporting of corresponding bit in GPIF_INTR
10	**WAVEFORM_BUSY**	Mask bit that controls reporting of corresponding bit in GPIF_INTR
9	**CTRL_COMP_HIT**	Mask bit that controls reporting of corresponding bit in GPIF_INTR
8	**DATA_COMP_HIT**	Mask bit that controls reporting of corresponding bit in GPIF_INTR
7	**ADDR_COMP_HIT**	Mask bit that controls reporting of corresponding bit in GPIF_INTR
6	**CTRL_COUNT_HIT**	Mask bit that controls reporting of corresponding bit in GPIF_INTR
5	**DATA_COUNT_HIT**	Mask bit that controls reporting of corresponding bit in GPIF_INTR
4	**ADDR_COUNT_HIT**	Mask bit that controls reporting of corresponding bit in GPIF_INTR

continued on next page

10.7.7 GPIF_INTR *(continued)*

3	**CRC_ERROR**	Mask bit that controls reporting of corresponding bit in GPIF_INTR
2	**SWITCH_TIMEOUT**	Mask bit that controls reporting of corresponding bit in GPIF_INTR
1	**GPIF_INTR**	Mask bit that controls reporting of corresponding bit in GPIF_INTR
0	**GPIF_DONE**	Mask bit that controls reporting of corresponding bit in GPIF_INTR

10.7.8 GPIF_CTRL_BUS_DIRECTION

Control Bus In/Out Direction Register

GPIF_CTRL_BUS_DIRECTION			Control Bus In/Out Direction Register				0xE0014024
b31	b30	b29	b28	b27	b26	b25	b24
DIRECTION[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_CTRL_BUS_DIRECTION			**Control Bus In/Out Direction Register**				
b23	b22	b21	b20	b19	b18	b17	b16
DIRECTION[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_CTRL_BUS_DIRECTION			**Control Bus In/Out Direction Register**				
b15	b14	b13	b12	b11	b10	b9	b8
DIRECTION[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_CTRL_BUS_DIRECTION			**Control Bus In/Out Direction Register**				
b7	b6	b5	b4	b3	b2	b1	b0
DIRECTION[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Two bits specify type of each bit in the 16-bit CTRL/ADDR Bus. Settings specified here may be overridden by GPIF_BUS_CONFIG bits.

Bit	Name	Description
31:0	**DIRECTION[31:0]**	Bit at (bit_number/2) has following direction: 00 Input 01 Output 10 Bidirectional I/O 11 Open drain I/O

10.7.9 GPIF_CTRL_BUS_DEFAULT

Control Bus Default Values Register

GPIF_CTRL_BUS_DEFAULT			Control Bus Default Values Register				0xE0014028
b31	b30	b29	b28	b27	b26	b25	b24
GPIF_CTRL_BUS_DEFAULT			**Control Bus Default Values Register**				
b23	b22	b21	b20	b19	b18	b17	b16
GPIF_CTRL_BUS_DEFAULT			**Control Bus Default Values Register**				
b15	b14	b13	b12	b11	b10	b9	b8
DEFAULT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_CTRL_BUS_DEFAULT			**Control Bus Default Values Register**				
b7	b6	b5	b4	b3	b2	b1	b0
DEFAULT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Reset/initialization value for the CTRL[15:0] signals.

Bit	Name	Description
15:0	**DEFAULT[15:0]**	One bit for each CTRL signal indicating default value 0 Asserted (see POLARITY) 1 Deasserted (see POLARITY)

10.7.10 GPIF_CTRL_BUS_POLARITY

Control Bus SIgnal Polarity Register

GPIF_CTRL_BUS_POLARITY			Control Bus SIgnal Polarity Register				0xE001402C
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_CTRL_BUS_POLARITY			Control Bus SIgnal Polarity Register				
b23	b22	b21	b20	b19	b18	b17	b16

GPIF_CTRL_BUS_POLARITY			Control Bus SIgnal Polarity Register				
b15	b14	b13	b12	b11	b10	b9	b8
POLARITY[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_CTRL_BUS_POLARITY			Control Bus SIgnal Polarity Register				
b7	b6	b5	b4	b3	b2	b1	b0
POLARITY[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Polarity of each of the CTRL[15:0] signals.

Bit	Name	Description
15:0	**POLARITY[15:0]**	One bit for each CTRL signal indicating polarity 0 Asserted when 1 1 Asserted when 0

10.7.11 GPIF_CTRL_BUS_TOGGLE

Control Bus Output Toggle Mode Register

GPIF_CTRL_BUS_TOGGLE			Control Bus Output Toggle Mode Register				0xE0014030
b31	b30	b29	b28	b27	b26	b25	b24
GPIF_CTRL_BUS_TOGGLE			**Control Bus Output Toggle Mode Register**				
b23	b22	b21	b20	b19	b18	b17	b16
GPIF_CTRL_BUS_TOGGLE			**Control Bus Output Toggle Mode Register**				
b15	b14	b13	b12	b11	b10	b9	b8
TOGGLE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_CTRL_BUS_TOGGLE			**Control Bus Output Toggle Mode Register**				
b7	b6	b5	b4	b3	b2	b1	b0
TOGGLE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Polarity of each of the CTRL[15:0] signals.

Bit	Name	Description
15:0	**TOGGLE[15:0]**	One bit for each CTRL signal indicating toggle mode 0 Normal mode, set value from alpha/beta 1 Toggle mode, toggle value when alpha/beta is 1, do nothing when 0

10.7.12 GPIF_CTRL_BUS_SELECT

Control Bus Connection Matrix Register

There are 16 GPIF_CTRL_BUS_SELECT registers. The address of each is calculated as GPIF_CTRL_BUS_SELECT(x) = 0xE0014034 + (x*0x4). Hence GPIF_CTRL_BUS_SELECT(0) is at address 0xE0014034, GPIF_CTRL_BUS_SELECT(1) is at address 0xE0014034 + 0x4 and so on. The definition of each of these is the same.

GPIF_CTRL_BUS_SELECT			Control Bus Connection Matrix Register				0xE0014034
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_CTRL_BUS_SELECT			Control Bus Connection Matrix Register				
b23	b22	b21	b20	b19	b18	b17	b16

GPIF_CTRL_BUS_SELECT			Control Bus Connection Matrix Register				
b15	b14	b13	b12	b11	b10	b9	b8

GPIF_CTRL_BUS_SELECT			Control Bus Connection Matrix Register				
b7	b6	b5	b4	b3	b2	b1	b0
			OMEGA_INDEX[4:0]				
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	0	0	0

These registers specify which of the alphas, betas or flags are to appear on each of CTRL[15:0]. They are only honored if the CTRL line is declared as an output.

Bit	Name	Description
4:0	OMEGA_INDEX[4:0]	For each omega, 5-bits specify what is driven at to the output: 0–3 Connect to alpha 0–3 8–11 Connect to beta 0–3 16–19 Empty/Full flags for thread 0–3 20–23 Partial flag for thread 0–3 24 Empty/Full flag for current thread 25 Partial flag for current thread 26 PP_DRQR5 signal (see PP_DRQR5_MASK) 27–31 Connected to logic 0 (cannot be used together with CTRL_BUS_TOGGLE)

10.7.13 GPIF_CTRL_COUNT_CONFIG

Control Counter Configuration Register

GPIF_CTRL_COUNT_CONFIG			Control Counter Configuration Register				0xE0014074
b31	b30	b29	b28	b27	b26	b25	b24
GPIF_CTRL_COUNT_CONFIG			**Control Counter Configuration Register**				
b23	b22	b21	b20	b19	b18	b17	b16
GPIF_CTRL_COUNT_CONFIG			**Control Counter Configuration Register**				
b15	b14	b13	b12	b11	b10	b9	b8
GPIF_CTRL_COUNT_CONFIG			**Control Counter Configuration Register**				
b7	b6	b5	b4	b3	b2	b1	b0
CONNECT[3:0				**SW_RESET**	**RELOAD**	**DOWN_UP**	**ENABLE**
R/W	R/W	R/W	R/W	R/W1S	R/W	R/W	R/W
R	R	R	R	R/W0C	R	R	R
0	0	0	0	0	1	1	0

Configures the 16-bit control counter

Bit	Name	Description	
7:4	**CONNECT[3:0]**	Connect the specified bit of this counter to CTRL[9]	
3	**SW_RESET**	0	Hardware write 0 to signal that counter has reset
		1	Software writes one to reset/load the counter
2	**RELOAD**	0	Saturate on reaching the limit
		1	Reload on reaching the limit
1	**DOWN_UP**	0	Down count
		1	Up count
0	**ENABLE**	0	This counter is not used.
		1	This counter is used.

10.7.14 GPIF_CTRL_COUNT_RESET

Control Counter Reset Register

GPIF_CTRL_COUNT_RESET			Control Counter Reset Register				0xE0014078
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_CTRL_COUNT_RESET			Control Counter Reset Register				
b23	b22	b21	b20	b19	b18	b17	b16

GPIF_CTRL_COUNT_RESET			Control Counter Reset Register				
b15	b14	b13	b12	b11	b10	b9	b8
RESET_LOAD[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_CTRL_COUNT_RESET			Control Counter Reset Register				
b7	b6	b5	b4	b3	b2	b1	b0
RESET_LOAD[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Configures the reset/load value of the control counter.

Bit	Name	Description
15:0	**RESET_LOAD[15:0]**	Reset counter to this value. Reload to this value when limit is reached if specified.

10.7.15 GPIF_CTRL_COUNT_LIMIT

Control Counter Limit Register

GPIF_CTRL_COUNT_LIMIT			Control Counter Reset Register				0xE001407C
b31	b30	b29	b28	b27	b26	b25	b24
GPIF_CTRL_COUNT_LIMIT			**Control Counter Reset Register**				
b23	b22	b21	b20	b19	b18	b17	b16
GPIF_CTRL_COUNT_LIMIT			**Control Counter Reset Register**				
b15	b14	b13	b12	b11	b10	b9	b8
LIMIT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
GPIF_CTRL_COUNT_LIMIT			**Control Counter Reset Register**				
b7	b6	b5	b4	b3	b2	b1	b0
LIMIT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFF							

Configures the limit value of the control counter.

Bit	Name	Description
15:0	**LIMIT[15:0]**	Stop counting when counter reaches this value

10.7.16 GPIF_ADDR_COUNT_CONFIG

Address Counter Configuration Register

GPIF_ADDR_COUNT_CONFIG			Address Counter Configuration Register				0xE0014080
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_ADDR_COUNT_CONFIG			Address Counter Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

GPIF_ADDR_COUNT_CONFIG			Address Counter Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
INCREMENT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1							

GPIF_ADDR_COUNT_CONFIG			Address Counter Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
				DOWN_UP	SW_RESET	RELOAD	ENABLE
				R/W	R/W1S	R/W	R/W
				R	R/W0C	R	R
				1	0	1	0

Configures the 16-bit address counter.

Bit	Name	Description	
7:0	INCREMENT[7:0]	8-bit quantity to be added/subtracted to the counter on each clock	
3	DOWN_UP	0	Down count
		1	Up count
2	SW_RESET	0	Hardware write 0 to signal that counter has reset
		1	Software writes one to reset/load the counter
1	RELOAD	0	Saturate on reaching the limit
		1	Reload on reaching the limit
0	ENABLE	0	This counter is not used.
		1	This counter is used.

10.7.17 GPIF_ADDR_COUNT_RESET

Address Counter Reset Register

GPIF_ADDR_COUNT_RESET			Address Counter Reset Register				0xE0014084
b31	b30	b29	b28	b27	b26	b25	b24
RESET_LOAD[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_ADDR_COUNT_RESET			**Address Counter Reset Register**				
b23	b22	b21	b20	b19	b18	b17	b16
RESET_LOAD[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_ADDR_COUNT_RESET			**Address Counter Reset Register**				
b15	b14	b13	b12	b11	b10	b9	b8
RESET_LOAD[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_ADDR_COUNT_RESET			**Address Counter Reset Register**				
b7	b6	b5	b4	b3	b2	b1	b0
RESET_LOAD[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Sets the reset/reload value of the data counter.

Bit	Name	Description
31:0	**RESET_LOAD[31:0]**	Reset counter to this value. Reload to this value when limit is reached if specified.

10.7.18 GPIF_ADDR_COUNT_LIMIT

Address Counter Limit Register

GPIF_ADDR_COUNT_LIMIT			Address Counter Limit Register				0xE0014088
b31	b30	b29	b28	b27	b26	b25	b24
LIMIT[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GPIF_ADDR_COUNT_LIMIT			Address Counter Limit Register				
b23	b22	b21	b20	b19	b18	b17	b16
LIMIT[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GPIF_ADDR_COUNT_LIMIT			Address Counter Limit Register				
b15	b14	b13	b12	b11	b10	b9	b8
LIMIT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GPIF_ADDR_COUNT_LIMIT			Address Counter Limit Register				
b7	b6	b5	b4	b3	b2	b1	b0
LIMIT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFF							

Configures the limit value of the address counter

Bit	Name	Description
31:0	**LIMIT[31:0]**	Stop counting when counter reaches this value.

10.7.19 GPIF_STATE_COUNT_CONFIG

State Counter Configuration Register

GPIF_STATE_COUNT_CONFIG	State Counter Configuration Register						0xE001408C
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_STATE_COUNT_CONFIG	State Counter Configuration Register						
b23	b22	b21	b20	b19	b18	b17	b16

GPIF_STATE_COUNT_CONFIG	State Counter Configuration Register						
b15	b14	b13	b12	b11	b10	b9	b8

GPIF_STATE_COUNT_CONFIG	State Counter Configuration Register						
b7	b6	b5	b4	b3	b2	b1	b0
						SW_RESET	**ENABLE**
						R/W1S	R/W
						R/W0C	R
						0	0

Configures the first 16-bit state counter.

Bit	Name		Description
1	**SW_RESET**	0	Hardware write 0 to signal that counter has reset
		1	Software writes one to reset/load the counter
0	**ENABLE**	0	This counter is not used.
		1	This counter is used.

10.7.20 GPIF_STATE_COUNT_LIMIT

State Counter Limit Register

GPIF_STATE_COUNT_LIMIT			State Counter Limit Register				0xE0014090
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_STATE_COUNT_LIMIT			State Counter Limit Register				
b23	b22	b21	b20	b19	b18	b17	b16

GPIF_STATE_COUNT_LIMIT			State Counter Limit Register				
b15	b14	b13	b12	b11	b10	b9	b8
LIMIT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GPIF_STATE_COUNT_LIMIT			State Counter Limit Register				
b7	b6	b5	b4	b3	b2	b1	b0
LIMIT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFF							

Configures the reset/load and limit values of counters.

Bit	Name	Description
15:0	**LIMIT[15:0]**	Generate an output tick, reset and start counting again if enabled when this limit is reached.

10.7.21 GPIF_DATA_COUNT_CONFIG

Data Counter Configuration Register

GPIF_DATA_COUNT_CONFIG			Data Counter Configuration Register				0xE0014094
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_DATA_COUNT_CONFIG			Data Counter Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

GPIF_DATA_COUNT_CONFIG			Data Counter Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
INCREMENT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1							

GPIF_DATA_COUNT_CONFIG			Data Counter Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
				DOWN_UP	SW_RESET	RELOAD	ENABLE
				R/W	R/W1S	R/W	R/W
				R	R/W0C	R	R
				1	0	1	0

Configures the 32-bit data counter.

Bit	Name	Description	
7:0	INCREMENT[7:0]	8-bit quantity to be added/subtracted to the counter on each clock	
3	DOWN_UP	0	Down count
		1	Up count
2	SW_RESET	0	Hardware write 0 to signal that counter has reset
		1	Software writes one to reset/load the counter
1	RELOAD	0	Saturate on reaching the limit
		1	Reload on reaching the limit
0	ENABLE	0	This counter is not used.
		1	This counter is used.

10.7.22 GPIF_DATA_COUNT_RESET

Data Counter Reset Register

GPIF_DATA_COUNT_RESET			Data Counter Reset Register				0xE0014098
b31	b30	b29	b28	b27	b26	b25	b24
RESET_LOAD[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_DATA_COUNT_RESET			**Data Counter Reset Register**				
b23	b22	b21	b20	b19	b18	b17	b16
RESET_LOAD[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_DATA_COUNT_RESET			**Data Counter Reset Register**				
b15	b14	b13	b12	b11	b10	b9	b8
RESET_LOAD[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_DATA_COUNT_RESET			**Data Counter Reset Register**				
b7	b6	b5	b4	b3	b2	b1	b0
RESET_LOAD[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Sets the reset/reload value of the data counter.

Bit	Name	Description
31:0	**RESET_LOAD[31:0]**	Reset counter to this value. Reload to this value when limit is reached if specified.

10.7.23 GPIF_DATA_COUNT_LIMIT

Data Counter Limit Register

GPIF_DATA_COUNT_LIMIT			Data Counter Limit Register				0xE001409C
b31	b30	b29	b28	b27	b26	b25	b24
LIMIT[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GPIF_DATA_COUNT_LIMIT			Data Counter Limit Register				
b23	b22	b21	b20	b19	b18	b17	b16
LIMIT[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GPIF_DATA_COUNT_LIMIT			Data Counter Limit Register				
b15	b14	b13	b12	b11	b10	b9	b8
LIMIT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GPIF_DATA_COUNT_LIMIT			Data Counter Limit Register				
b7	b6	b5	b4	b3	b2	b1	b0
LIMIT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFF							

Sets the limit value of the data counter.

Bit	Name	Description
31:0	**LIMIT[31:0]**	Reload data counter if this limit is reached and reload is enabled.

10.7.24 GPIF_CTRL_COMP_VALUE

Control Comparator Value Register

GPIF_CTRL_COMP_VALUE			Control Comparator Value Register				0xE00140A0
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_CTRL_COMP_VALUE			Control Comparator Value Register				
b23	b22	b21	b20	b19	b18	b17	b16

GPIF_CTRL_COMP_VALUE			Control Comparator Value Register				
b15	b14	b13	b12	b11	b10	b9	b8
VALUE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_CTRL_COMP_VALUE			Control Comparator Value Register				
b7	b6	b5	b4	b3	b2	b1	b0
VALUE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Sets the target value for the 16-bit control-bus comparator.

Bit	Name	Description
15:0	**VALUE[15:0]**	Output true when CTRL bus matches this value

10.7.25 GPIF_CTRL_COMP_MASK

Control Comparator Mask Register

GPIF_CTRL_COMP_MASK			Control Comparator Mask Register				0xE00140A4
b31	b30	b29	b28	b27	b26	b25	b24
GPIF_CTRL_COMP_MASK			**Control Comparator Mask Register**				
b23	b22	b21	b20	b19	b18	b17	b16
GPIF_CTRL_COMP_MASK			**Control Comparator Mask Register**				
b15	b14	b13	b12	b11	b10	b9	b8
MASK[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_CTRL_COMP_MASK			**Control Comparator Mask Register**				
b7	b6	b5	b4	b3	b2	b1	b0
MASK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Sets the comparison mask for the 16-bit control-bus comparator.

Bit	Name	Description	
15:0	**MASK[15:0]**	0	Bit at this bit position is a don't-care for comparison
		1	Bit at this bit position in the CTRL bus is to be used in comparison

10.7.26 GPIF_DATA_COMP_VALUE

Data Comparator Value Register

GPIF_DATA_COMP_VALUE			Data Comparator Value Register				0xE00140A8
b31	b30	b29	b28	b27	b26	b25	b24
VALUE[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_DATA_COUNT_LIMIT			**Data Counter Limit Register**				
b23	b22	b21	b20	b19	b18	b17	b16
VALUE[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_DATA_COUNT_LIMIT			**Data Counter Limit Register**				
b15	b14	b13	b12	b11	b10	b9	b8
VALUE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_DATA_COUNT_LIMIT			**Data Counter Limit Register**				
b7	b6	b5	b4	b3	b2	b1	b0
VALUE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Sets the target value for the 32-bit data comparator.

Bit	Name	Description
31:0	**VALUE[31:0]**	Output true when Data bus matches this value.

10.7.27 GPIF_DATA_COMP_MASK

Data Comparator Mask Register

GPIF_DATA_COMP_MASK			Data Comparator Mask Register				0xE00140AC
b31	b30	b29	b28	b27	b26	b25	b24
MASK[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_DATA_COMP_MASK			**Data Comparator Mask Register**				
b23	b22	b21	b20	b19	b18	b17	b16
MASK[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_DATA_COMP_MASK			**Data Comparator Mask Register**				
b15	b14	b13	b12	b11	b10	b9	b8
MASK[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_DATA_COMP_MASK			**Data Comparator Mask Register**				
b7	b6	b5	b4	b3	b2	b1	b0
MASK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Sets the comparison mask for the 32-bit data comparator.

Bit	Name	Description	
31:0	**MASK[31:0]**	0	Bit at this bit position is a don't-care for comparison
		1	Bit at this bit position in the Data bus is to be used in comparison

10.7.28 GPIF_ADDR_COMP_VALUE

Address Comparator Value Register

GPIF_ADDR_COMP_VALUE			Address Comparator Value Register				0xE00140B0
b31	b30	b29	b28	b27	b26	b25	b24
VALUE[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_ADDR_COMP_VALUE			**Address Comparator Value Register**				
b23	b22	b21	b20	b19	b18	b17	b16
VALUE[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_ADDR_COMP_VALUE			**Address Comparator Value Register**				
b15	b14	b13	b12	b11	b10	b9	b8
VALUE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_ADDR_COMP_VALUE			**Address Comparator Value Register**				
b7	b6	b5	b4	b3	b2	b1	b0
VALUE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Sets the target value for the 32-bit address comparator.

Bit	Name	Description
31:0	**VALUE[31:0]**	Output true when Data bus matches this value.

10.7.29 GPIF_ADDR_COMP_MASK

Address Comparator Mask Register

GPIF_ADDR_COMP_MASK			Address Comparator Mask Register				0xE00140B4
b31	b30	b29	b28	b27	b26	b25	b24
MASK[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_ADDR_COMP_MASK			**Address Comparator Mask Register**				
b23	b22	b21	b20	b19	b18	b17	b16
MASK[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_ADDR_COMP_MASK			**Address Comparator Mask Register**				
b15	b14	b13	b12	b11	b10	b9	b8
MASK[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_ADDR_COMP_MASK			**Address Comparator Mask Register**				
b7	b6	b5	b4	b3	b2	b1	b0
MASK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Sets the comparison mask for the 32-bit data comparator.

Bit	Name	Description	
31:0	**MASK[31:0]**	0	Bit at this bit position is a don't-care for comparison
		1	Bit at this bit position in the CTRL bus is to be used in comparison

10.7.30 GPIF_DATA_CTRL

Data Control Register

GPIF_DATA_CTRL			Data Control Register				0xE00140B8
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_DATA_CTRL			Data Control Register				
b23	b22	b21	b20	b19	b18	b17	b16

GPIF_DATA_CTRL			Data Control Register				
b15	b14	b13	b12	b11	b10	b9	b8
EG_ADDR_VALID[3:0]				IN_ADDR_VALID[3:0]			
R/W1S	R/W1S	R/W1S	R/W1S	R/W1C	R/W1C	R/W1C	R/W1C
R/W0C	R/W0C	R/W0C	R/W0C	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

GPIF_DATA_CTRL			Data Control Register				
b7	b6	b5	b4	b3	b2	b1	b0
EG_DATA_VALID[3:0]				IN_DATA_VALID[3:0]			
R/W1S	R/W1S	R/W1S	R/W1S	R/W1C	R/W1C	R/W1C	R/W1C
R/W0C	R/W0C	R/W0C	R/W0C	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	1	0	1	0

Configures and controls flow of data through GPIF_INGRESS_DATA/EGRESS_DATA registers. Provides valid flags for the data registers associated with the four threads.

Bit	Name	Description
15:12	**EG_ADDR_VALID[3:0]**	Software writes 1 to indicate a valid word is present in the address register. Hardware writes 0 to indicate that the data is used and new word can be written.
11:8	**IN_ADDR_VALID[3:0]**	Indicates address available in INGRESS_ADDRESS. Cleared by software when address processed.
7:4	**EG_DATA_VALID[3:0]**	Software writes 1 to indicate a valid word is present in the address register. Hardware writes 0 to indicate that the data is used and new word can be written.
3:0	**IN_DATA_VALID[3:0]**	Indicates data available in INGRESS_DATA. Cleared by software when data processed.

10.7.31 GPIF_INGRESS_DATA

Socket Ingress Data Register

There are four GPIF_INGRESS_DATA registers. The address of each is calculated as GPIF_INGRESS_DATA(x) = 0xE00140BC + (x*0x4). Hence GPIF_INGRESS_DATA(0) is at address 0xE00140BC, GPIF_INGRESS_DATA(1) is at address 0xE00140BC + 0x4 and so on. The definition of each of these is the same.

GPIF_INGRESS_DATA			Socket Ingress Data Register				0xE00140BC
b31	b30	b29	b28	b27	b26	b25	b24
DATA[31:24]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

GPIF_INGRESS_DATA			Socket Ingress Data Register				
b23	b22	b21	b20	b19	b18	b17	b16
DATA[23:16]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

GPIF_INGRESS_DATA			Socket Ingress Data Register				
b15	b14	b13	b12	b11	b10	b9	b8
DATA[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

GPIF_INGRESS_DATA			Socket Ingress Data Register				
b7	b6	b5	b4	b3	b2	b1	b0
DATA[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Holds ingress data for the active socket in a thread.

Bit	Name	Description
31:0	**DATA[31:0]**	Ingress Data. This register will hold only one word of GPIF_BUS_CONFIG.BUS_WIDTH. No packing/ unpacking is done. MSBs will be 0.

10.7.32 GPIF_EGRESS_DATA

Socket Egress Data Register

There are four GPIF_EGRESS_DATA registers. The address of each is calculated as GPIF_EGRESS_DATA(x) = 0xE00140CC + (x*0x4). Hence GPIF_EGRESS_DATA(0) is at address 0xE00140CC, GPIF_EGRESS_DATA(1) is at address 0xE00140CC + 0x4 and so on. The definition of each of these is the same.

GPIF_EGRESS_DATA			Socket Egress Data Register				0xE00140CC
b31	b30	b29	b28	b27	b26	b25	b24
DATA[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_EGRESS_DATA			Socket Egress Data Register				
b23	b22	b21	b20	b19	b18	b17	b16
DATA[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_EGRESS_DATA			Socket Egress Data Register				
b15	b14	b13	b12	b11	b10	b9	b8
DATA[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_EGRESS_DATA			Socket Egress Data Register				
b7	b6	b5	b4	b3	b2	b1	b0
DATA[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Holds egress data for the active socket in a thread.

Bit	Name	Description
31:0	**DATA[31:0]**	Egress data. This register will hold only one word of GPIF_BUS_CONFIG.BUS_WIDTH. No packing/ unpacking is done. MSBs are ignored.

10.7.33 GPIF_INGRESS_ADDRESS

Thread Ingress Address Register

There are four GPIF_INGRESS_ADDRESS registers. The address of each is calculated as GPIF_INGRESS_ADDRESS(x) = 0xE00140DC + (x*0x4). Hence GPIF_INGRESS_ADDRESS(0) is at address 0xE00140DC, GPIF_INGRESS_ADDRESS(1) is at address 0xE00140DC + 0x4 and so on. The definition of each of these is the same.

GPIF_INGRESS_ADDRESS			Thread Ingress Address Register				0xE00140DC
b31	b30	b29	b28	b27	b26	b25	b24
ADDRESS[31:24]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

GPIF_INGRESS_ADDRESS			Thread Ingress Address Register				
b23	b22	b21	b20	b19	b18	b17	b16
ADDRESS[23:16]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

GPIF_INGRESS_ADDRESS			Thread Ingress Address Register				
b15	b14	b13	b12	b11	b10	b9	b8
ADDRESS[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

GPIF_INGRESS_ADDRESS			Thread Ingress Address Register				
b7	b6	b5	b4	b3	b2	b1	b0
ADDRESS[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Holds ingress address for the active socket in a thread.

Bit	Name	Description
31:0	**ADDRESS[31:0]**	Ingress address

10.7.34 GPIF_EGRESS_ADDRESS

Thread Egress Address Register

There are four GPIF_EGRESS_ADDRESS registers. The address of each is calculated as GPIF_EGRESS_ADDRESS(x) = 0xE00140EC + (x*0x4). Hence GPIF_EGRESS_ADDRESS(0) is at address 0xE00140EC, GPIF_EGRESS_ADDRESS(1) is at address 0xE00140EC + 0x4 and so on. The definition of each of these is the same.

GPIF_EGRESS_ADDRESS			Thread Egress Address Register				0xE00140EC
b31	b30	b29	b28	b27	b26	b25	b24
ADDRESS[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_EGRESS_ADDRESS			Thread Egress Address Register				
b23	b22	b21	b20	b19	b18	b17	b16
ADDRESS[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_EGRESS_ADDRESS			Thread Egress Address Register				
b15	b14	b13	b12	b11	b10	b9	b8
ADDRESS[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_EGRESS_ADDRESS			Thread Egress Address Register				
b7	b6	b5	b4	b3	b2	b1	b0
ADDRESS[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Holds egress address for the active socket in a thread.

Bit	Name	Description
31:0	**ADDRESS[31:0]**	Egress address

10.7.35 GPIF_THREAD_CONFIG

Thread Configuration Register

There are four GPIF_THREAD_CONFIG registers. The address of each is calculated as GPIF_THREAD_CONFIG(x) = 0xE00140FC + (x*0x4). Hence GPIF_THREAD_CONFIG(0) is at address 0xE00140FC, GPIF_THREAD_CONFIG(1) is at address 0xE00140FC + 0x4 and so on. The definition of each of these is the same.

GPIF_THREAD_CONFIG			Thread Configuration Register				0xE00140FC
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE		**WATERMARK[13:8]**					
R/W		R/W	R/W	R/W	R/W	R/W	R/W
R/W		R	R	R	R	R	R
0							

GPIF_THREAD_CONFIG			Thread Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16
WATERMARK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1							

GPIF_THREAD_CONFIG			Thread Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
				BURST_SIZE[3:0]			
				R/W	R/W	R/W	R/W
				R	R	R	R
				4			

GPIF_THREAD_CONFIG			Thread Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
WM_CFG			**THREAD_SOCK[4:0]**				
R/W			R/W	R/W	R/W	R/W	R/W
R			R/W	R/W	R/W	R/W	R/W
0			0	1	0	1	0

Configures the active socket and thread watermark. (All sockets behind the thread share the watermark). The hardware gets the status and direction of the sockets from the adapter.The watermark signal is the limit hit indicator of a counter that counts read/write accesses that happen on the interface. The limit of this counter is programmed by the hardware to “end of usable space” and is reset to 0 by PIB hardware as dma_ready (available flag) goes from 0->1.

Bit	Name	Description
31	**ENABLE**	Enables the thread controller for operation. Can be set by firmware after initializing THREAD_SOCK and other fields. Will be set by hardware when THREAD_SOCK is written to by hardware.
29:16	**WATERMARK[13:0]**	Watermark position. Indicates number words that would be subtracted from the end of usable space/ data. Watermark needs to be programmed to a value greater than the round trip flag latency of the system. This latency is a sum of three quantities namely (1) FX3 latency between seeing the end of the last burst that completely fills the buffer to the time the full/empty flag updates (can be calculated from generic gpif params), (2). Expected time of arrival of a flow control signal that would prevent the AP from issuing the next burst (as measured from the end of the burst) (3). Any additional group latency between the APs dma controller logic and the interface pins in both directions.

continued on next page

10.7.35 GPIF_THREAD_CONFIG *(continued)*

		"end of usable space/data" is calculated using one of the equations depending on the context. (1) For normal write transfers in either PP modes, this is the size of the buffer to be produced into. (2) For normal read transfers in either PP modes, this is the byte count of the buffer to be consumed. (3) For partial buffer read or write transfers in PP mode 1, this is the value of written by AP into the PP_DMA_SIZE register rounded up to an integral number of BURSTSIZE quanta.
11:8	**BURST_SIZE[3:0]**	Log2 of burst size (1: 2 words, 2: 4 words, etc). Programmed to support systems that work with fixed burst sizes. A burst is defined as a portion of transfer that unconditionally completes after it is initiated. The system must always transfer an entire burst before responding to a change in a partial flag. In transfers that involve short packet, the PIB hardware will automatically append zeros/truncate data to do its part in preserving the above mentioned definition of burst. Burst size is a power of 2 and must be programmed to a value greater than watermark when partial flag is used.Buffer sizes used should be integral multiple of the burst size. Maximum value is 14. When value is >0, any socket switching must occur on 8 byte boundaries.Besides, this field needs to be programmed to a non-zero value to support bandwidth>200MBps on P-Port.
7	**WM_CFG**	0 Assert partial flag when number of samples (remaining available for reading/writing) is less than or equal to the watermark. 1 Assert when samples are more than the watermark.
4:0	**THREAD_SOCK[4:0]**	Active Socket Number for this thread. Can be written by software for fixed socket assignment.(all threads) Can be modified by hardware as result of PP_DMA_XFER accesses (only for thread 0) Can be modified by hardware as result of alpha 'sample AIN' (all threads. Hardware can only modify bits [4:2] of this field)

10.7.36 GPIF_LAMBDA_STAT

Lambda Status Register

GPIF_LAMBDA_STAT			Lambda Status Register				0xE001410C
b31	b30	b29	b28	b27	b26	b25	b24
LAMBDA[31:24]							
R	R	R	R	R	R	R	R
R/w	R/w	R/w	R/w	R/w	R/w	R/w	R/w
GPIF_LAMBDA_STAT			Lambda Status Register				
b23	b22	b21	b20	b19	b18	b17	b16
LAMBDA[23:16]							
R	R	R	R	R	R	R	R
R/w	R/w	R/w	R/w	R/w	R/w	R/w	R/w
GPIF_LAMBDA_STAT			Lambda Status Register				
b15	b14	b13	b12	b11	b10	b9	b8
LAMBDA[15:8]							
R	R	R	R	R	R	R	R
R/w	R/w	R/w	R/w	R/w	R/w	R/w	R/w
GPIF_LAMBDA_STAT			Lambda Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
LAMBDA[7:0]							
R	R	R	R	R	R	R	R
R/w	R/w	R/w	R/w	R/w	R/w	R/w	R/w
0x10000000							

Provides the current state of the 32 Lambdas (inputs).

Bit	Name	Description
31:0	**LAMBDA[31:0]**	Current value of the Lambda inputs

10.7.37 GPIF_ALPHA_STAT

Alpha Status Register

GPIF_ALPHA_STAT			Alpha Status Register				0xE0014110
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_ALPHA_STAT			Alpha Status Register				
b23	b22	b21	b20	b19	b18	b17	b16

GPIF_ALPHA_STAT			Alpha Status Register				
b15	b14	b13	b12	b11	b10	b9	b8

GPIF_ALPHA_STAT			Alpha Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
ALPHA[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	1	0	1	0

Provides the current state of the 8 Alphas (state machine early outputs).

Bit	Name	Description
7:0	**ALPHA[7:0]**	Current value of the Alpha signals

10.7.38 GPIF_BETA_STAT

Beta Status Register

GPIF_BETA_STAT			Beta Status Register				0xE0014114
b31	b30	b29	b28	b27	b26	b25	b24
BETA[31:24]							
R	R	R	R	R	R	R	R
R/w	R/w	R/w	R/w	R/w	R/w	R/w	R/w
0	0	0	0	0	0	0	0
GPIF_BETA_STAT			Beta Status Register				
b23	b22	b21	b20	b19	b18	b17	b16
BETA[23:16]							
R	R	R	R	R	R	R	R
R/w	R/w	R/w	R/w	R/w	R/w	R/w	R/w
0	0	0	0	0	0	0	0
GPIF_BETA_STAT			Beta Status Register				
b15	b14	b13	b12	b11	b10	b9	b8
BETA[15:8]							
R	R	R	R	R	R	R	R
R/w	R/w	R/w	R/w	R/w	R/w	R/w	R/w
0	0	0	0	0	0	0	0
GPIF_BETA_STAT			Beta Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
BETA[7:0]							
R	R	R	R	R	R	R	R
R/w	R/w	R/w	R/w	R/w	R/w	R/w	R/w
0	0	0	0	0	0	0	0

Provides the current state of the 32 Betas (state machine late outputs).

Bit	Name	Description
31:0	**BETA[31:0]**	Current value of the Beta signals

10.7.39 GPIF_WAVEFORM_CTRL_STAT

Waveform Program Control Register

GPIF_WAVEFORM_CTRL_STAT			Waveform Program Control Register				0xE0014118
b31	b30	b29	b28	b27	b26	b25	b24
CURRENT_STATE[7:0]							
R	R	R	R	R	R	R	R
R/w	R/w	R/w	R/w	R/w	R/w	R/w	R/w
0	0	0	0	0	0	0	0

GPIF_DATA_CTRL			Data Control Register				
b23	b22	b21	b20	b19	b18	b17	b16
ALPHA_INIT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_DATA_CTRL			Data Control Register				
b15	b14	b13	b12	b11	b10	b9	b8
				CPU_LAMBDA	GPIF_STAT[2:0]		
				R/W	R	R	R
				R	R/w	R/w	R/w
				0	0	0	0

GPIF_DATA_CTRL			Data Control Register				
b7	b6	b5	b4	b3	b2	b1	b0
						PAUSE	WAVEFORM_VALID
						R/W	R/W
						R	R
						0	0

Offers the facility to program new waveforms to GPIF. Provides current status of GPIF.

Bit	Name	Description
31:24	**CURRENT_STATE[7:0]**	Current state of GPIF. Always updated.
23:16	**ALPHA_INIT[7:0]**	Initial values for alpha outputs. These are loaded into the alpha registers when GPIF execution starts (first WAVEFORM_SWITCH) is set.
11	**CPU_LAMBDA**	Visible to the state machine as lambda 30.
10:8	**GPIF_STAT[2:0]**	0 Waveform is not valid (Initial state or WAVEFORM_VALID is cleared) 1 <unused> 2 GPIF is armed (WAVEFORM_VALID is set) 3 GPIF is running (using WAVEFORM_SWITCH) 4 GPIF is done (encountered DONE_STATE) 5 GPIF is paused (PAUSE = 0) 6 GPIF is switching (waiting for timeout/terminal state) 7 An error occurred
1	**PAUSE**	Write 1 here to pause GPIF. 0 to resume where left off.
0	**WAVEFORM_VALID**	0 Waveforms are no longer valid, stop operation and return outputs to default state

1	The waveform memory is consistent and valid.

10.7.40 GPIF_WAVEFORM_SWITCH

Waveform Switch Control Register

GPIF_WAVEFORM_SWITCH			Waveform Switch Control Register				0xE001411C
b31	b30	b29	b28	b27	b26	b25	b24
DONE_STATE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_WAVEFORM_SWITCH			Waveform Switch Control Register				
b23	b22	b21	b20	b19	b18	b17	b16
DESTINATION_STATE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_WAVEFORM_SWITCH			Waveform Switch Control Register				
b15	b14	b13	b12	b11	b10	b9	b8
TERMINAL_STATE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_WAVEFORM_SWITCH			Waveform Switch Control Register				
b7	b6	b5	b4	b3	b2	b1	b0
TERMINATED	TIMEOUT_ REACHED	TIMEOUT_MODE[2:0]			SWITCH_NOW	DONE_ENABLE	WAVEFORM_ SWITCH
R	R	R/W	R/W	R/W	R/W	R/W	R/W1S
R/W	R/W	R	R	R	R	R	R/W0C
0	0	0	0	0	0	0	0

Offers the facility to program new waveforms to GPIF. Provides current status of GPIF.

Bit	Name	Description
31:24	DONE_STATE[7:0]	Signal GPIF_DONE upon reaching this state.
23:16	DESTINATION_STATE[7:0]	State to jump to, may be the initial state of the new waveform.
15:8	TERMINAL_STATE[7:0]	State from which to initiate the switch. Corresponds to idle states of waveforms.
7	TERMINATED	Indicates that the TERMINAL_STATE was reached since last WAVEFORM_SWITCH
6	TIMEOUT_REACHED	Indicates that timeout was reached since last WAVEFORM_SWITCH
5:3	TIMEOUT_MODE[2:0]	0 Timeout disable 1 Timeout for reaching TERMINAL_STATE. Interrupt on timeout 2 Timeout for reaching DONE_STATE. Interrupt on timeout 3 Timeout for reaching TERMINAL STATE. Force switch on timeout. 4 Timeout for hanging in current state. Timer resets on each transition.
2	SWITCH_NOW	1 Do not wait for TERMINAL_STATE, switch right away

continued on next page

10.7.40 GPIF_WAVEFORM_SWITCH *(continued)*

1	**DONE_ENABLE**	1	Enable checking for DONE_STATE and generation of GPIF_DONE.
0	**WAVEFORM_SWITCH**	Software sets this bit after programming the switch register. Hardware clears it after the switch is complete.	

10.7.41 GPIF_WAVEFORM_SWITCH_TIMEOUT

Waveform Timeout Register

GPIF_WAVEFORM_SWITCH_TIMEOUT			Waveform Timeout Register				0xE0014120
b31	b30	b29	b28	b27	b26	b25	b24
RESET_LOAD[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_WAVEFORM_SWITCH_TIMEOUT			Waveform Timeout Register				
b23	b22	b21	b20	b19	b18	b17	b16
RESET_LOAD[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_WAVEFORM_SWITCH_TIMEOUT			Waveform Timeout Register				
b15	b14	b13	b12	b11	b10	b9	b8
RESET_LOAD[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_WAVEFORM_SWITCH_TIMEOUT			Waveform Timeout Register				
b7	b6	b5	b4	b3	b2	b1	b0
RESET_LOAD[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Defines the timeout counter (in the number of state machine clock) for waveform switching. Effective only when TIMEOUT_ENABLE is set.

Bit	Name	Description
31:0	**RESET_LOAD[31:0]**	Timeout value

10.7.42 GPIF_CRC_CONFIG

CRC Configuration Register

GPIF_CRC_CONFIG			CRC Configuration Register				0xE0014124
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE							
R/W							
R							
0							

GPIF_CRC_CONFIG			CRC Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16
	CRC_ERROR	**BYTE_ENDIAN**	**BIT_ENDIAN**				
	R	R/W	R/W				
	R/W	R	R				
	0	0	0				

GPIF_CRC_CONFIG			CRC Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
CRC_RECEIVED[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

GPIF_CRC_CONFIG			CRC Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
CRC_RECEIVED[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Configuration of CRC computation for data stream.

Bit	Name	Description
31	**ENABLE**	Enables CRC calculation
22	**CRC_ERROR**	A CRC was loaded into CRC_RECEIVED that is different from CRC_VALUE
21	**BYTE_ENDIAN**	Indicates the order in which bytes in a 32-bit word are brought through the CRC shift register. This is independent from the endianness of the interface. 0 LSB first 1 MSB first
20	**BIT_ENDIAN**	Indicates the order in which the bits in each byte are brought through the CRC shift register. 0 LSb first 1 MSb first
15:0	**CRC_RECEIVED[15:0]**	A CRC value received on the data inputs by the state machine

10.7.43 GPIF_CRC_DATA

CRC Data Register

GPIF_CRC_DATA			CRC Data Register				0xE0014128
b31	b30	b29	b28	b27	b26	b25	b24
CRC_VALUE[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
GPIF_CRC_DATA			**CRC Data Register**				
b23	b22	b21	b20	b19	b18	b17	b16
CRC_VALUE[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
GPIF_CRC_DATA			**CRC Data Register**				
b15	b14	b13	b12	b11	b10	b9	b8
INITIAL_VALUE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
GPIF_CRC_DATA			**CRC Data Register**				
b7	b6	b5	b4	b3	b2	b1	b0
INITIAL_VALUE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

CRC initial values and results.

Bit	Name	Description
31:16	**CRC_VALUE[15:0]**	Current result of CRC calculation
15:8	**INITIAL_VALUE[15:0]**	Initial CRC value

10.7.44 GPIF_BETA_DEASSERT

Beta Deassert Register

GPIF_BETA_DEASSERT			Beta Deassert Register				0xE001412C
b31	b30	b29	b28	b27	b26	b25	b24
APPLY_DEASSERT[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GPIF_BETA_DEASSERT			Beta Deassert Register				
b23	b22	b21	b20	b19	b18	b17	b16
APPLY_DEASSERT[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GPIF_BETA_DEASSERT			Beta Deassert Register				
b15	b14	b13	b12	b11	b10	b9	b8
APPLY_DEASSERT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

GPIF_BETA_DEASSERT			Beta Deassert Register				
b7	b6	b5	b4	b3	b2	b1	b0
APPLY_DEASSERT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1							

Controls the applicability of BETA_DEASSERT descriptor field to individual betas.

Bit	Name		Description
31:0	APPLY_DEASSERT[31:0]	0	BETA_DEASSERT does not apply. Betas remain asserted throughout the state.
		1	BETA_DEASSERT from the waveform descriptor applies to this beta. This is not honored for external betas, which always behave as if apply_deassert = 0.

10.7.45 GPIF_FUNCTION

Transition Function Registers

There are 32 GPIF_FUNCTION registers. The address of each is calculated as GPIF_FUNCTION(x) = 0xE0014130 + (x*0x4). Hence GPIF_FUNCTION(0) is at address 0xE0014130, GPIF_FUNCTION(1) is at address 0xE0014130 + 0x4 and so on. The definition of each of these is the same.

GPIF_FUNCTION			Transition Function Registers				0xE0014130
b31	b30	b29	b28	b27	b26	b25	b24

GPIF_CRC_CONFIG			CRC Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

GPIF_CRC_CONFIG			CRC Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
FUNCTION[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_CRC_CONFIG			CRC Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
FUNCTION[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Contains the truth tables for user-defined transition functions.

Bit	Name	Description
15:0	FUNCTION[15:0]	Truth table for transition function. Bit position X contains output when the 4 inputs constitute the value X in binary. For example, bit 2 = 1 means in3 = 0, in2 = 0, in1 = 1 and in0 = 0 will evaluate true for this function.

10.7.46 GPIF_LEFT_WAVEFORM

Left Edge Waveform Memory Register

There are 256 GPIF_LEFT_WAVEFORM registers. The address of each is calculated as GPIF_LEFT_WAVEFORM(x) = 0xE0015000 + (x*0x10). Hence GPIF_LEFT_WAVEFORM(0) is at address 0xE0015000, GPIF_LEFT_WAVEFORM(1) is at address 0xE0015000 + 0x10, and so on. The definition of each of these is the same.

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				0xE0015000
b127	b126	b125	b124	b123	b122	b121	b120
UNUSED							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b119	b118	b117	b116	b115	b114	b113	b112
UNUSED							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b111	b110	b109	b108	b107	b106	b105	b104
UNUSED							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b103	b102	b101	b100	b99	b98	b97	b96
UNUSED							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b95	b94	b93	b92	b91	b90	b89	b88
VALID	BETA_DEASSERT	REPEAT_COUNT[7:2]					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b87	b86	b85	b84	b83	b82	b81	b80
REPEAT_COUNT[1:0]		Beta[31:26]					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b79	b78	b77	b76	b75	b74	b73	b72
Beta[25:18]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.7.46 GPIF_LEFT_WAVEFORM *(continued)*

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b71	b70	b69	b68	b67	b66	b65	b64
Beta[17:10]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b63	b62	b61	b60	b59	b58	b57	b56
Beta[9:2]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b55	b54	b53	b52	b51	b50	b49	b48
Beta[1:0]		**Alpha_Right[7:2]**					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b47	b46	b45	b44	b43	b42	b41	b40
Alpha_Right[1:0]		**Alpha_Left[7:2]**					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b39	b38	b37	b36	b35	b34	b33	b32
Alpha_Left[1:0]		**f1[4:0]**					**f0[4]**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b31	b30	b29	b28	b27	b26	b25	b24
f0[3:0]				**Fd[4:1]**			
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b23	b22	b21	b20	b19	b18	b17	b16
Fd[0]	**Fc[4:0]**					**Fb[4:3]**	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.7.46 GPIF_LEFT_WAVEFORM *(continued)*

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b15	b14	b13	b12	b11	b10	b9	b8
Fb[2:0]			Fa[4:0]				
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_LEFT_WAVEFORM			Left Edge Waveform Memory Register				
b7	b6	b5	b4	b3	b2	b1	b0
NEXT_STATE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This area stores the so-called left edge state transitions from a given state. Location 17 stores the left state transition from state 17, and so on. This is a logical register implemented in 256x96b SRAM aligned on 128b boundaries. In other words the left state transition for state 17 is stored in three 32-bit words at offset+4*17, offset+4*17+1, offset+4*17+2.

Bit	Name	Description
127:96	**UNUSED**	This entry is unused and does not map to RAM in the current implementation. Writes are ignored and reads will return 0 in all cases.
95	**VALID**	0 Entry not valid. (Not programmed or edge does not exist) 1 This entry is valid.
94	**BETA_DEASSERT**	0 Keep betas asserted throughout the state 1 Deassert after asserting for exactly one clock cycle, irrespective of how many cycles the state is active. The normal (deassert) state of user defined betas is defined in GPIF_CTRL_BUS_DEFAULT. The normal state of internal betas is fixed by hardware. This function is applied only to betas selected in GPIF_BETA_DEASSERT.
93:86	**REPEAT_COUNT[7:0]**	Number of times to stay in this state – 1
85:54	**Beta[31:0]**	32 secondary outputs
53:46	**Alpha_Right[7:0]**	For the right edge
45:38	**Alpha_Left{7:0]**	Primary outputs for the left edge of the next state.
37:33	**f1[4:0]**	Index to select the second transition function from a choice of 32 functions. Truth-tables for the 32 4-bit functions are defined using the GPIF_FUNCTION registers.
32:28	**f0[4:0]**	Index to select the first transition function from a choice of 32 functions. Truth-tables for the 32 4-bit functions are defined using the GPIF_FUNCTION registers.
27:23	**Fd[4:0]**	Fourth input index
22:18	**Fc[4:0]**	Third input index.
17:13	**Fb[4:0]**	Second input index.
12:8	**Fa[4:0]**	Index to select the first input for transition functions out of 32 choices.
7:0	**NEXT_STATE[7:0]**	Next state on left transition

10.7.47 GPIF_RIGHT_WAVEFORM

Right Edge Waveform Memory Register

There are 256 GPIF_RIGHT_WAVEFORM registers. The address of each is calculated as GPIF_RIGHT_WAVEFORM(x) = 0xE0016000 + (x*0x10). Hence GPIF_RIGHT_WAVEFORM(0) is at address 0xE0016000, GPIF_RIGHT_WAVEFORM(1) is at address 0xE0016000 + 0x10, and so on. The definition of each of these is the same.

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				0xE0016000
b127	b126	b125	b124	b123	b122	b121	b120
UNUSED							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b119	b118	b117	b116	b115	b114	b113	b112
UNUSED							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b111	b110	b109	b108	b107	b106	b105	b104
UNUSED							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b103	b102	b101	b100	b99	b98	b97	b96
UNUSED							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b95	b94	b93	b92	b91	b90	b89	b88
VALID	BETA_DEASSERT	REPEAT_COUNT[7:2]					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b87	b86	b85	b84	b83	b82	b81	b80
REPEAT_COUNT[1:0]		Beta[31:26]					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b79	b78	b77	b76	b75	b74	b73	b72
Beta[25:18]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.7.47 GPIF_LEFT_WAVEFORM *(continued)*

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b71	b70	b69	b68	b67	b66	b65	b64
Beta[17:10]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b63	b62	b61	b60	b59	b58	b57	b56
Beta[9:2]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b55	b54	b53	b52	b51	b50	b49	b48
Beta[1:0]		**Alpha_Right[7:2]**					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b47	b46	b45	b44	b43	b42	b41	b40
Alpha_Right[1:0]		**Alpha_Left[7:2]**					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b39	b38	b37	b36	b35	b34	b33	b32
Alpha_Left[1:0]		**f1[4:0]**					**f0[4]**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b31	b30	b29	b28	b27	b26	b25	b24
f0[3:0]				**Fd[4:1]**			
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b23	b22	b21	b20	b19	b18	b17	b16
Fd[0]	**Fc[4:0]**					**Fb[4:3]**	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.7.47 GPIF_LEFT_WAVEFORM *(continued)*

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b15	b14	b13	b12	b11	b10	b9	b8
Fb[2:0]			Fa[4:0]				
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

GPIF_RIGHT_WAVEFORM			Right Edge Waveform Memory Register				
b7	b6	b5	b4	b3	b2	b1	b0
NEXT_STATE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This area stores the so-called right edge state transitions from a given state. Location 17 stores the right state transition from state 17, and so on. This is a logical register implemented in 256x96b SRAM aligned on 128b boundaries. In other words the right state transition for state 17 is stored in three 32-bit words at offset+4*17, offset+4*17+1, offset+4*17+2.

Bit	Name	Description
127:96	**UNUSED**	This entry is unused and does not map to RAM in the current implementation. Writes are ignored and reads will return 0 in all cases.
95	**VALID**	0 Entry not valid. (Not programmed or edge does not exist) 1 This entry is valid.
94	**BETA_DEASSERT**	0 Keep betas asserted throughout the state 1 Deassert after asserting for exactly one clock cycle, irrespective of how many cycles the state is active. The normal (deassert) state of user defined betas is defined in GPIF_CTRL_BUS_DEFAULT. The normal state of internal betas is fixed by hardware. This function is applied only to betas selected in GPIF_BETA_DEASSERT.
93:86	**REPEAT_COUNT[7:0]**	Number of times to stay in this state – 1
85:54	**Beta[31:0]**	32 secondary outputs
53:46	**Alpha_Right[7:0]**	For the right edge
45:38	**Alpha_Left{7:0]**	Primary outputs for the left edge of the next state.
37:33	**f1[4:0]**	Index to select the second transition function from a choice of 32 functions. Truth-tables for the 32 4-bit functions are defined using the GPIF_FUNCTION registers.
32:28	**f0[4:0]**	Index to select the first transition function from a choice of 32 functions. Truth-tables for the 32 4-bit functions are defined using the GPIF_FUNCTION registers.
27:23	**Fd[4:0]**	Fourth input index
22:18	**Fc[4:0]**	Third input index.
17:13	**Fb[4:0]**	Second input index.
12:8	**Fa[4:0]**	Index to select the first input for transition functions out of 32 choices.
7:0	**NEXT_STATE[7:0]**	Next state on left transition

10.8 P-Port Registers

10.8.1 PP_ID

P-Port Device ID Register

PP_ID				P-Port Device ID Register			0xE0017E00
b15	b14	b13	b12	b11	b10	b9	b8
DEVICE_ID[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PP_ID				**P-Port Device ID Register**			
b7	b6	b5	b4	b3	b2	b1	b0
DEVICE_ID[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Provides device ID information. This must be provided by boot ROM. This register is not writable when GCTL_CONTROL.BOOTROM_EN = 0 to prevent spoofing.

Bit	Name	Description
15:0	**DEVICE_ID[15:0]**	Provides a device ID.

10.8.2 PP_INIT

P-Port Reset and Power Control Register

PP_INIT			P-Port Reset and Power Control Register				0xE0017E04
b15	b14	b13	b12	b11	b10	b9	b8
BIG_ENDIAN				**HARD_RESET_N**	**CPU_RESET_N**		
R/W				R/W0C	R/W		
R				R	R		
0				1	1		

PP_INIT			P-Port Reset and Power Control Register				
b7	b6	b5	b4	b3	b2	b1	b0
			WAKEUP_CLK	**WAKEUP_PWR**	**WDT_RESET**	**SW_RESET**	**POR**
			R	R	R	R/W	R/W
			R/W	R/W	R/W	R	R
			0	0	0	0	1

This register is used for reset and power control and determines endian orientation of the P-port.

Bit	Name	Description
15	**BIG_ENDIAN**	0 P-Port is Little Endian 1 P-Port is Big Endian
11	**HARD_RESET_N**	Software clears this bit to effect a global hard reset (all blocks, all flops). This is equivalent to toggling the RESET pin on the device. This function is also available to internal firmware in GCTL_CONTROL.
10	**CPU_RESET_N**	Software clears this bit to effect a CPU reset (aka reboot). No other blocks or registers are affected. The CPU will enter the boot ROM, that will use the WARM_BOOT flag to determine whether to reload firmware. Unlike the same bit in GCTL_CONTROL, the software needs to explicitly clear and then set this bit to bring the internal CPU out of reset. It is permissible to keep the ARM CPU in reset for an extended period of time (although not advisable).
4	**WAKEUP_CLK**	Indicates system woke up from suspend state. If firmware does not clear this bit it will stay 1 even through standby sequences. This bit is a shadow bit of GCTL_CONTROL.
3	**WAKEUP_PWR**	Indicates system woke up from standby mode. If firmware does not clear this bit it will stay 1 even through suspend sequences. This bit is a shadow bit of GCTL_CONTROL.
2	**WDT_RESET**	Indicates system woke up from a watchdog timer induced hard reset (see GCTL_WATCHDOG_CS). If firmware does not clear this bit it will stay 1 even through standby and suspend sequences. This bit is a shadow bit of GCTL_CONTROL.
1	**SW_RESET**	Indicates system woke up from a software induced hard reset sequence (from GCTL_CONTROL.HARD_RESET_N or PP_INIT.HARD_RESET_N). If firmware does not clear this bit it will stay 1 even through standby and suspend sequences. This bit is a shadow bit of GCTL_CONTROL.
0	**POR**	Indicates system woke up through a power-on-reset or RESET# pin reset sequence. If firmware does not clear this bit it will stay 1 even through software reset, standby and suspend sequences. This bit is a shadow bit of GCTL_CONTROL.

10.8.3 PP_CONFIG

P-Port Configuration Register

PP_CONFIG			P-Port Configuration Register				0xE0017E08
b15	b14	b13	b12	b11	b10	b9	b8
DACK_POLARITY	DRQ_POLARITY	DRQ_VALUE	DRQ_OVERRIDE	INTR_POLARITY	INTR_VALUE	INTR_OVERRIDE	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	
R	R	R	R	R	R	R	
0	0	0	0	0	0	0	

PP_CONFIG			P-Port Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
DRQMODE	CFGMODE			BURSTSIZE[3:0]			
R/W	R/W			R/W	R/W	R/W	R/W
R	R/W			R	R	R	R
0	1			15			

This register holds bits required to control PMMC interface.

Bit	Name	Description
15	DACK_POLARITY	0 DACK is active low 1 DACK is active high
14	DRQ_POLARITY	0 DRQ is active low 1 DRQ is active high
13	DRQ_VALUE	0 DRQ is deasserted when DRQ_OVERRIDE = 1 1 DRQ is asserted when DRQ_OVERRIDE = 1
12	DRQ_OVERRIDE	0 No override 1 DRQ signal is forced to DRQ_VALUE
11	INTR_POLARITY	0 INTR is deasserted when INTR_OVERRIDE = 1 1 INTR is asserted on override when INTR_OVERRIDE=1 This bit is used directly in hardware to generate INT signal.
10	INTR_VALUE	0 INTR is deasserted when INTR_OVERRIDE = 1 1 INTR is asserted on override when INTR_OVERRIDE = 1 This bit is used directly in hardware to generate INT signal.
9	INTR_OVERRIDE	0 No override 1 INTR signal is forced to INTR_VALUE This bit is used directly in hardware to generate INT signal.
7	DRQMODE	DMA signaling mode. See DMA section for more information. 0 Pulse mode, DRQ will deassert when DACK deasserts and will remain d-asserted for a specified time. After that DRQ may reassert depending on other settings. 1 Burst mode, DRQ will deassert when BURSTSIZE words are transferred and will not reassert until DACK is deasserted.

continued on next page

10.8.3 PP_CONFIG *(continued)*

6	**CFGMODE**	Initialization Mode 0 Normal operation mode. 1 Initialization mode. This bit is cleared to “0” by firmware, by writing 0 to PIB_CONFIG.PP_CFGMODE, after completing the initialization process. Specific usage of this bit is described in the software architecture. This bit is mirrored directly by hardware in PIB_CONFIG.
3:0	**BURSTSIZE**	Size of DMA bursts; only relevant when DRQMODE=1. 0–14 DMA burst size is 2BURSTSIZE words 15 DMA burst size is infinite (DRQ deasserts on last cycle of transfer)

10.8.4 PP_INTR_MASK

P-Port Interrupt Mask Register

PP_INTR_MASK			P-Port Interrupt Mask Register				0xE0017E1C
b15	b14	b13	b12	b11	b10	b9	b8
WAKEUP	**WR_MB_EMPTY**	**RD_MB_FULL**	**DMA_READY_EV**	**DMA_WMARK_EV**			
R/W	R/W	R/W	R/W	R/W			
R	R	R	R	R			
0	0	1	0	0			

PP_INTR_MASK			P-Port Interrupt Mask Register				
b7	b6	b5	b4	b3	b2	b1	b0
GPIF_ERR		**PIB_ERR**	**GPIF_INT**	**SOCK_AGG_BH**	**SOCK_AGG_BL**	**SOCK_AGG_AH**	**SOCK_AGG_AL**
R/W		R/W	R/W	R/W	R/W	R/W	R/W
R		R	R	R	R	R	R
0		0	0	0	0	0	0

This register holds bits required to control PMMC interface.

Bit	Name	Description	
15	**WAKEUP**	1	Forward EVENT onto INT line
14	**WR_MB_EMPTY**	1	Forward EVENT onto INT line
13	**RD_MB_FULL**	1	Forward EVENT onto INT line
12	**DMA_READY_EV**	1	Forward EVENT onto INT line
11	**DMA_WMARK_EV**	1	Forward EVENT onto INT line
7	**GPIF_ERR**	1	Forward EVENT onto INT line
5	**PIB_ERR**	1	Forward EVENT onto INT line
4	**GPIF_INT**	1	Forward EVENT onto INT line
3	**SOCK_AGG_BH**	1	Forward EVENT onto INT line
2	**SOCK_AGG_BL**	1	Forward EVENT onto INT line
1	**SOCK_AGG_AH**	1	Forward EVENT onto INT line
0	**SOCK_AGG_AL**	1	Forward EVENT onto INT line

10.8.5 PP_DRQR5_MASK

P-Port DRQ/R5 Mask Register

PP_DRQR5_MASK			P-Port DRQ/R5 Mask Register				0xE0017E20
b15	b14	b13	b12	b11	b10	b9	b8
WAKEUP	**WR_MB_EMPTY**	**RD_MB_FULL**	**DMA_READY_EV**	**DMA_WMARK_EV**			
R/W	R/W	R/W	R/W	R/W			
R	R	R	R	R			
0	0	1	0	0			

PP_DRQR5_MASK			P-Port DRQ/R5 Mask Register				
b7	b6	b5	b4	b3	b2	b1	b0
GPIF_ERR		**PIB_ERR**	**GPIF_INT**	**SOCK_AGG_BH**	**SOCK_AGG_BL**	**SOCK_AGG_AH**	**SOCK_AGG_AL**
R/W		R/W	R/W	R/W	R/W	R/W	R/W
R		R	R	R	R	R	R
0		0	0	0	0	0	0

These registers have the same layout as PP_EVENT and mask which events lead to assertion of INTR or DRQ/R5 respectively. DRQR5 is a signal that can be put on any GPIF CTRL[x] line (see GPIF_BUS_SELECT) and R5 is an MMC event notification protocol relevant only in PMMC mode.

Bit	Name		Description
15	**WAKEUP**	1	Forward EVENT onto DRQ line
14	**WR_MB_EMPTY**	1	Forward EVENT onto DRQ line
13	**RD_MB_FULL**	1	Forward EVENT onto DRQ line
12	**DMA_READY_EV**	1	Forward EVENT onto DRQ line
11	**DMA_WMARK_EV**	1	Forward EVENT onto DRQ line
7	**GPIF_ERR**	1	Forward EVENT onto DRQ line
5	**PIB_ERR**	1	Forward EVENT onto DRQ line
4	**GPIF_INT**	1	Forward EVENT onto DRQ line
3	**SOCK_AGG_BH**	1	Forward EVENT onto DRQ line
2	**SOCK_AGG_BL**	1	Forward EVENT onto DRQ line
1	**SOCK_AGG_AH**	1	Forward EVENT onto DRQ line
0	**SOCK_AGG_AL**	1	Forward EVENT onto DRQ line

10.8.6 PP_SOCK_MASK

P-Port Socket Mask Register

PP_SOCK_MASK			P-Port Socket Mask Register				0xE0017E24
b31	b30	b29	b28	b27	b26	b25	b24
SOCK_MASK[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PP_SOCK_MASK			**P-Port Socket Mask Register**				
b23	b22	b21	b20	b19	b18	b17	b16
SOCK_MASK[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PP_SOCK_MASK			**P-Port Socket Mask Register**				
b15	b14	b13	b12	b11	b10	b9	b8
SOCK_MASK[15:18]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PP_SOCK_MASK			**P-Port Socket Mask Register**				
b7	b6	b5	b4	b3	b2	b1	b0
SOCK_MASK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

These registers contain a mask that indicates which sockets affect the SOCK_AGG_A and SOCK_AGG_B values, respectively.

Bit	Name	Description
31:0	**SOCK_MASK[31:0]**	For socket <x>, bit <x> indicates: 0 Socket does not affect SOCK_AGG_A/B 1 Socket does affect SOCK_AGG_A/B

10.8.7 PP_ERROR

P-Port Error Indicator Register

PP_ERROR	P-Port Error Indicator Register						0xE0017E28
b15	b14	b13	b12	b11	b10	b9	b8
	GPIF_ERR_CODE[4:0]						
	R	R	R	R	R		
	R/W	R/W	R/W	R/W	R/W		
	0	0	0	0	0		

PP_ERROR	P-Port Error Indicator Register						
b7	b6	b5	b4	b3	b2	b1	b0
		PIB_ERR_CODE[5:0]					
		R	R	R	R	R	R
		R/W	R/W	R/W	R/W	R/W	R/W
		0	0	0	0	0	0

This register indicates the error codes associated with PIB_INTR.PIB_ERR, MMC_ERR and GPIF_ERR. The different values for these error codes will be documented in the P-Port BROS document. This register is also visible to firmware as PIB_ERROR.

Bit	Name	Description
14:10	**GPIF_ERR_CODE[4:0]**	Mirror of corresponding field in PIB_ERROR
5:0	**PIB_ERR_CODE[5:0]**	Mirror of corresponding field in PIB_ERROR

10.8.8 PP_DMA_XFER

P-Port DMA Transfer Register

PP_DMA_XFER			P-Port DMA Transfer Register				0xE0017E2C
b15	b14	b13	b12	b11	b10	b9	b8
DMA_READY	**DMA_ERROR**	**DMA_BUSY**	**SIZE_VALID**		**LONG_TRANSFER**	**DMA_DIRECTION**	**DMA_ENABLE**
R	R	R	R		R/W	R/W	R/W
W	W	W	R/W		R	R	R/W
0	0	0	0		0	0	0

PP_DMA_XFER			P-Port DMA Transfer Register				
b7	b6	b5	b4	b3	b2	b1	b0
DMA_SOCK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register is used to set up and control a DMA transfer.

Bit	Name	Description
15	**DMA_READY**	Indicates that the link controller is ready to exchange data. 0 Socket not ready for transfer 1 Socket ready for transfer; SIZE_VALID is also guaranteed 1
14	**DMA_ERROR**	0 No errors 1 DMA transfer error This bit is set when a DMA error occurs and cleared when the next transfer is started using DMA_ENABLE = 1.
13	**DMA_BUSY**	Indicates that link controller is busy processing a transfer. A zero length transfer would cause DMA_READY to never assert. 0 No DMA is in progress 1 DMA is busy
12	**SIZE_VALID**	Indicates that DMA_SIZE value is valid and corresponds to the socket selected in PP_DMA_XFER. SIZE_VALID will be 0 for a short period after PP_DMA_XFER is written into. AP will poll SIZE_VALID or DMA_READY before reading DMA_SIZE.
10	**LONG_TRANSFER**	0 Short transfer (DMA_ENABLE clears at end of buffer 1 Long Transfer (DMA_ENABLE must be cleared by AP at end of transfer)
9	**DMA_DIRECTION**	0 Read (Transfer from FX3 – Egress direction) 1 Write (Transfer to FX3 – Ingress direction)
8	**DMA_ENABLE**	0 Disable ongoing transfer. If no transfer is ongoing ignore disable 1 Enable data transfer
7:0	**DMA_SOCK[7:0]**	Processor specified socket number for data transfer

10.8.9 PP_DMA_SIZE

P-Port DMA Transfer Size Register

PP_DMA_SIZE	P-Port DMA Transfer Size Register						0xE0017E30
b15	b14	b13	b12	b11	b10	b9	b8
DMA_SIZE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PP_DMA_SIZE	P-Port DMA Transfer Size Register						
b7	b6	b5	b4	b3	b2	b1	b0
DMA_SIZE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

This register indicates the (remaining) size of the transfer. It is initialized to the number of bytes available in the buffer for egress transfers and the size of the buffer for ingress transfers. This register can be modified by the AP for shorter ingress transfers. The value read from this register is not valid unless DMA_XFER.SIZE_VALID is true.

Bit	Name	Description
15:0	**DMA_SIZE[15:0]**	Size of DMA transfer. Number of bytes available for read/write when read, number of bytes to be read/written when written.

10.8.10 PP_WR_MAILBOX

P-Port Write (Ingress) Mailbox Registers

PP_WR_MAILBOX			P-Port Write (Ingress) Mailbox Registers				0xE0017E34
b63	b62	b61	b60	b59	b58	b57	b56
WR_MAILBOX[63:56]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PP_WR_MAILBOX			**P-Port Write (Ingress) Mailbox Registers**				
b55	b54	b53	b52	b51	b50	b49	b48
WR_MAILBOX[55:48]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PP_WR_MAILBOX			**P-Port Write (Ingress) Mailbox Registers**				
b47	b46	b45	b44	b43	b42	b41	b40
WR_MAILBOX[47:40]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PP_WR_MAILBOX			**P-Port Write (Ingress) Mailbox Registers**				
b39	b38	b37	b36	b35	b34	b33	b32
WR_MAILBOX[39:32]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PP_WR_MAILBOX			**P-Port Write (Ingress) Mailbox Registers**				
b31	b30	b29	b28	b27	b26	b25	b24
WR_MAILBOX[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PP_WR_MAILBOX			**P-Port Write (Ingress) Mailbox Registers**				
b23	b22	b21	b20	b19	b18	b17	b16
WR_MAILBOX[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PP_WR_MAILBOX			**P-Port Write (Ingress) Mailbox Registers**				
b15	b14	b13	b12	b11	b10	b9	b8
WR_MAILBOX[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.8.10 PP_WR_MAILBOX *(continued)*

PP_WR_MAILBOX			P-Port Write (Ingress) Mailbox Registers				
b7	b6	b5	b4	b3	b2	b1	b0
WR_MAILBOX[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

These registers contain a message of up to 8 bytes from the Application Processor to firmware. The semantics of the possible messages is defined as part of the software specification. These registers also appear in the P-Port MMIO space as PIB_WR_MAILBOX. When the Application Processor writes data into the high word of PP_WR_MAILBOX the interrupt PIB_INTR.WR_MB_FULL is set. The expected action is that firmware reads the message and then clears PIB_INTR.WR_MB_FULL, which will set PP_EVENT.WR_MB_EMPTY to signal AP for the next message (if needed).

Bit	Name	Description
63:0	**WR_MAILBOX[63:0]**	Write mailbox message from AP

10.8.11 PP_MMIO_ADDR

P-Port MMIO Address Registers

PP_MMIO_ADDR			P-Port MMIO Address Registers				0xE0017E3C
b31	b30	b29	b28	b27	b26	b25	b24
MMIO_ADDR[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PP_MMIO_ADDR			**P-Port MMIO Address Registers**				
b23	b22	b21	b20	b19	b18	b17	b16
MMIO_ADDR[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PP_MMIO_ADDR			**P-Port MMIO Address Registers**				
b15	b14	b13	b12	b11	b10	b9	b8
MMIO_ADDR[15:18]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PP_MMIO_ADDR			**P-Port MMIO Address Registers**				
b7	b6	b5	b4	b3	b2	b1	b0
MMIO_ADDR[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

These registers together form a 32-bit address for accessing the FX3 internal MMIO space. The address can point to any internal FX3 register. The bits PP_MMIO.MMIO_RD, PP_MMIO.MMIO_WR, PP_MMIO.MMIO_DONE are used to control the operation.

Bit	Name	Description
31:0	**MMIO_ADDR[31:0]**	Address in MMIO register space to be used for access.

10.8.12 PP_MMIO_DATA

P-Port MMIO Data Registers

PP_MMIO_DATA			P-Port MMIO Data Registers				0xE0017E40
b31	b30	b29	b28	b27	b26	b25	b24
MMIO_DATA[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PP_MMIO_ADDR			**P-Port MMIO Address Registers**				
b23	b22	b21	b20	b19	b18	b17	b16
MMIO_DATA[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PP_MMIO_ADDR			**P-Port MMIO Address Registers**				
b15	b14	b13	b12	b11	b10	b9	b8
MMIO_DATA[15:18]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PP_MMIO_ADDR			**P-Port MMIO Address Registers**				
b7	b6	b5	b4	b3	b2	b1	b0
MMIO_DATA[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

These registers together form a 32-bit data word for accessing the FX3 internal MMIO space. Only full 32-bit accesses are supported to FX3 registers. The bits PP_MMIO.MMIO_RD, PP_MMIO.MMIO_WR, PP_MMIO.MMIO_DONE are used to control the operation.

Reading from these registers will return the data from the last MMIO_RD operation or the data written by Application Processor, whichever is more recent.

Bit	Name	Description
31:0	**MMIO_DATA[31:0]**	32-bit data word for read or write transaction

10.8.13 PP_MMIO

P-Port MMIO Control Registers

PP_MMIO			P-Port MMIO Control Registers				0xE0017E44
b15	b14	b13	b12	b11	b10	b9	b8

PP_MMIO			P-Port MMIO Control Registers				
b7	b6	b5	b4	b3	b2	b1	b0
				MMIO_FAIL	**MMIO_BUSY**	**MMIO_WR**	**MMIO_RD**
				R	R	R/W1S	R/W1S
				R/W	R/W	R/W0C	R/W0C
				0	0	0	0

This register controls the access to FX3's MMIO space. The process for using the PP_MMIOxxx registers is described later in a separate section.

Bit	Name	Description	
3	**MMIO_FAIL**	0	Normal
		1	MMIO operation was not executed because of security constraints (PIB_CONFIG.MMIO_ENABLE = 0)
2	**MMIO_BUSY**	0	No MMIO operation is pending
		1	MMIO operation is being executed
1	**MMIO_WR**	0	No action
		1	Initiate write of MMIO_DATA to MMIO_ADDR
0	**MMIO_RD**	0	No action
		1	Initiate read from MMIO_ADDR, place data in MMIO_DATA

10.8.14 PP_EVENT

P-Port Event Register

PP_EVENT			P-Port Event Register				0xE0017E48
b15	b14	b13	b12	b11	b10	b9	b8
WAKEUP	**WR_MB_EMPTY**	**RD_MB_FULL**	**DMA_READY_EV**	**DMA_WMARK_EV**			
R/W1C	R/W	R/W1C	R	R			
W1S	R/W1C	W1S	R/W	R/W			
0	1	0	0	0			

PP_EVENT			P-Port Event Register				
b7	b6	b5	b4	b3	b2	b1	b0
GPIF_ERR		**PIB_ERR**	**GPIF_INT**	**SOCK_AGG_BH**	**SOCK_AGG_BL**	**SOCK_AGG_AH**	**SOCK_AGG_AL**
R/W1C		R/W1C	R/W1C	R	R	R	R
W1S		W1S	W1S	R/W	R/W	R/W	R/W
0		0	0	0	0	0	0

This register indicates all types of events that can cause INTR or DRQ to assert.

Bit	Name	Description
15	**WAKEUP**	Reserved. AP should read PP_INIT register's WAKEUP_PWR or WAKEUP_CLK to detect if there was a wake up from standby or suspend.
14	**WR_MB_EMPTY**	1 WR Mailbox is empty - message can be written This field is cleared by PIB when message is written to MBX, but can also be cleared by AP when used as interrupt. This field is set by PIB only once when MBX is emptied by firmware.
13	**RD_MB_FULL**	1 RD Mailbox is full - message must be read
12	**DMA_READY_EV**	0 P-port not ready for data transfer 1 P-port ready for data transfer
11	**DMA_WMARK_EV**	0 P-Port has fewer than <watermark> words left (can be 0) 1 P-Port is ready for transfer and at least <watermark> words remain
7	**GPIF_ERR**	An error occurred in the GPIF. Firmware clears this bit after handling the error. The error code is indicated in PP_ERROR.GPIF_ERR_CODE
5	**PIB_ERR**	The socket based link controller encountered an error and needs attention. Firmware clears this bit after handling the error. The error code is indicated in PP_ERROR.PIB_ERR_CODE
4	**GPIF_INT**	1 State machine raised host interrupt
3	**SOCK_AGG_BH**	0 SOCK_STAT_B[15:8] is all zeroes 1 At least one bit set in SOCK_STAT_B[15:8]

continued on next page

2	**SOCK_AGG_BL**	0	SOCK_STAT_B[7:0] is all zeroes
		1	At least one bit set in SOCK_STAT_B[7:0]
1	**SOCK_AGG_AH**	0	SOCK_STAT_A[15:8] is all zeroes
		1	At least one bit set in SOCK_STAT_A[15:8]
0	**SOCK_AGG_AL**	0	SOCK_STAT_A[7:0] is all zeroes
		1	At least one bit set in SOCK_STAT_A[7:0]

10.8.15 PP_RD_MAILBOX

P-Port Read (Egress) Mailbox Registers

PP_RD_MAILBOX	P-Port Read (Egress) Mailbox Registers						0xE0017E4C
b63	b62	b61	b60	b59	b58	b57	b56
RD_MAILBOX[63:56]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PP_RD_MAILBOX	P-Port Read (Egress) Mailbox Registers						
b55	b54	b53	b52	b51	b50	b49	b48
RD_MAILBOX[55:48]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PP_RD_MAILBOX	P-Port Read (Egress) Mailbox Registers						
b47	b46	b45	b44	b43	b42	b41	b40
RD_MAILBOX[47:40]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PP_RD_MAILBOX	P-Port Read (Egress) Mailbox Registers						
b39	b38	b37	b36	b35	b34	b33	b32
RD_MAILBOX[39:32]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PP_RD_MAILBOX	P-Port Read (Egress) Mailbox Registers						
b31	b30	b29	b28	b27	b26	b25	b24
RD_MAILBOX[31:24]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PP_RD_MAILBOX	P-Port Read (Egress) Mailbox Registers						
b23	b22	b21	b20	b19	b18	b17	b16
RD_MAILBOX[23:16]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PP_RD_MAILBOX	P-Port Read (Egress) Mailbox Registers						
b15	b14	b13	b12	b11	b10	b9	b8
RD_MAILBOX[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

continued on next page

10.8.15 PP_RD_MAILBOX *(continued)*

PP_RD_MAILBOX			P-Port Read (Egress) Mailbox Registers				
b7	b6	b5	b4	b3	b2	b1	b0
RD_MAILBOX[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

These registers contain a message of up to 8 bytes from firmware to the Application Processor. The semantics of the possible messages will be defined as part of the software specification. These registers also appear in the P-Port MMIO space as PIB_RD_MAILBOX*. When firmware writes data into the high word of PIB_RD_MAILBOX the event PP_EVENT.RD_MB_FULL is set. The expected action is that the Application Processor reads the message and interprets it and then clears PP_EVENT.RD_MB_FULL, which sets PIB_INTR.RD_MB_EMPTY to signal firmware for the next message (if needed).

Bit	Name	Description
63:0	**RD_MAILBOX[63:0]**	Read mailbox message to AP

10.8.16 PP_SOCK_STAT

P-Port Socket Status Register

PP_SOCK_STAT			P-Port Socket Status Register				0xE0017E54
b31	b30	b29	b28	b27	b26	b25	b24
SOCK_STAT[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
PP_SOCK_STAT			P-Port Socket Status Register				
b23	b22	b21	b20	b19	b18	b17	b16
SOCK_STAT[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
PP_SOCK_STAT			P-Port Socket Status Register				
b15	b14	b13	b12	b11	b10	b9	b8
SOCK_STAT[15:18]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
PP_SOCK_STAT			P-Port Socket Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
SOCK_STAT[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

These registers contain one bit for each of the 32 sockets in the P-port, indicating the buffer availability of each socket.

Bit	Name	Description
31:0	SOCK_STAT[31:0]	For socket <x>, bit <x> indicates: 0 Socket has no active descriptor or descriptor is not available (empty for write, occupied for read) 1 Socket is available for reading or writing

10.8.17 PP_BUF_SIZE_CNT

P-Port Socket Buffer Size or Count Register

PP_BUF_SIZE_CNT			P-Port Socket Buffer Size or Count Register				0xE0017E??
b15	b14	b13	b12	b11	b10	b9	b8
SIZE_CNT[15:8]							
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PP_DMA_SIZE			P-Port DMA Transfer Size Register				
b7	b6	b5	b4	b3	b2	b1	b0
SIZE_CNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

These registers contain the buffer size or count, depending on direction, of the corresponding socket (0..31). This is equal to buffer_size/buffer_count in the corresponding DSCR_SIZE/DSCR_COUNT register. These registers are present in the PMMC P-Port space only! No registers exist in the PP space, only the socket registers are read for this from the adapter. The value of PP_BUF_SIZE_CNT is valid only if the PP_SOCK_STAT indicates ready for the corresponding socket.

Bit	Name	Description
15:0	**SIZE_CNT[15:0]**	Buffer size of the corresponding write socket (0..31)

10.9 USB Port Registers

10.9.1 UIB_INTR

USB Interrupt Register

UIB_INTR	USB Interrupt Register						0xE0030000
b31	b30	b29	b28	b27	b26	b25	b24

UIB_INTR	USB Interrupt Register						
b23	b22	b21	b20	b19	b18	b17	b16

UIB_INTR	USB Interrupt Register						
b15	b14	b13	b12	b11	b10	b9	b8
			EPM_URUN_TIMEOUT	**EPM_URUN**	**PROT_EP_INT**	**PROT_INT**	**LNK_INT**
			R/W1C	R/W1C	R	R	R
			R/W1S	R/W1S	R/W	R/W	R/W
			X	X	X	X	X

UIB_INTR	USB Interrupt Register						
b7	b6	b5	b4	b3	b2	b1	b0
CHGDET_INT	**OTG_INT**	**DEV_CTL_INT**	**DEV_EP_INT**	**OHCI_INT**	**EHCI_INT**	**HOST_EP_INT**	**HOST_INT**
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
X	X	X	X	X	X	X	X

Master interrupt vector. UIB uses hierarchical interrupt structure due to the large number of interrupt causes. This register can only be read from and not updated. Interrupts are cleared by clearing the leaf interrupt source.

Bit	Name	Description
12	**EPM_URUN_TIMEOUT**	SuperSpeed Egress EPM Interrupt
11	**EPM_URUN**	SuperSpeed Egress EPM Interrupt
10	**PROT_EP_INT**	SuperSpeed Device Endpoint Interrupt
9	**PROT_INT**	SuperSpeed Protocol Layer Interrupt
8	**LNK_INT**	SuperSpeed Link Controller Interrupt
7	**CHGDET_INT**	USB Charger Detect Interrupt
6	**OTG_INT**	USB OTG Interrupt

continued on next page

10.9.1 UIB_INTR *(continued)*

5	**DEV_CTL_INT**	Device USB Control Interrupt
4	**DEV_EP_INT**	Device EP Interrupt
3	**OHCI_INT**	OHCI Interrupt
2	**EHCI_INT**	EHCI Interrupt
1	**HOST_EP_INT**	Host EP Interrupt
0	**HOST_INT**	Host INT Status Register Interrupt

10.9.2 UIB_INTR_MASK

USB Interrupt Mask Register

UIB_INTR_MASK			USB Interrupt Mask Register				0xE0030004
b31	b30	b29	b28	b27	b26	b25	b24

UIB_INTR_MASK			USB Interrupt Mask Register				
b23	b22	b21	b20	b19	b18	b17	b16

UIB_INTR_MASK			USB Interrupt Mask Register				
b15	b14	b13	b12	b11	b10	b9	b8
			EPM_URUN_TIMEOUT	**EPM_URUN**	**PROT_EP_INT**	**PROT_INT**	**LNK_INT**
			R/W1C	R/W1C	R	R	R
			R/W1S	R/W1S	R/W	R/W	R/W
			X	X	X	X	X

UIB_INTR_MASK			USB Interrupt Mask Register				
b7	b6	b5	b4	b3	b2	b1	b0
CHGDET_INT	**OTG_INT**	**DEV_CTL_INT**	**DEV_EP_INT**	**OHCI_INT**	**EHCI_INT**	**HOST_EP_INT**	**HOST_INT**
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
X	X	X	X	X	X	X	X

Mask for UIB_INTR. Does not affect error logging, only reporting to VIC.

Bit	Name	Description
12	**EPM_URUN_TIMEOUT**	SuperSpeed Egress EPM Interrupt
11	**EPM_URUN**	SuperSpeed Egress EPM Interrupt
10	**PROT_EP_INT**	SuperSpeed Device Endpoint Interrupt
9	**PROT_INT**	SuperSpeed Protocol Layer Interrupt
8	**LNK_INT**	SuperSpeed Link Controller Interrupt
7	**CHGDET_INT**	USB Charger Detect Interrupt
6	**OTG_INT**	USB OTG Interrupt
5	**DEV_CTL_INT**	Device USB Control Interrupt
4	**DEV_EP_INT**	Device EP Interrupt

continued on next page

10.9.2 UIB_INTR_MASK *(continued)*

3	**OHCI_INT**	OHCI Interrupt
2	**EHCI_INT**	EHCI Interrupt
1	**HOST_EP_INT**	Host EP Interrupt
0	**HOST_INT**	Host INT Status Register Interrupt

10.9.3 UIB_ID

Block Identification and Version Number Register

UIB_ID	Block Identification and Version Number Register						0xE0037F00
b31	b30	b29	b28	b27	b26	b25	b24
BLOCK_VERSION[15:8]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R

UIB_ID	Block Identification and Version Number Register						
b23	b22	b21	b20	b19	b18	b17	b16
BLOCK_VERSION[7:0]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0x0001							

UIB_ID	Block Identification and Version Number Register						
b15	b14	b13	b12	b11	b10	b9	b8
BLOCK_ID[15:8]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R

UIB_ID	Block Identification and Version Number Register						
b7	b6	b5	b4	b3	b2	b1	b0
BLOCK_ID[7:0]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0x0003							

Every IP block will implement a few MMIO registers at offset 0 in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31:16	**BLOCK_VERSION[15:0]**	Version number for the IP
15:8	**BLOCK_ID[15:0]**	A unique number identifying the IP in the memory space

10.9.4 UIB_POWER

Power, Clock, and Reset Control Registers

UIB_POWER			Power, Clock, and Reset Control Registers				0xE0037F04
b31	b30	b29	b28	b27	b26	b25	b24
RESETN							
R							
W							
0							

UIB_POWER			Power, Clock, and Reset Control Registers				
b23	b22	b21	b20	b19	b18	b17	b16

UIB_POWER			Power, Clock, and Reset Control Registers				
b15	b14	b13	b12	b11	b10	b9	b8

UIB_POWER			Power, Clock, and Reset Control Registers				
b7	b6	b5	b4	b3	b2	b1	b0
							ACTIVE
							R/W
							R
							0

Every IP block will implement a few MMIO registers at offset 0x7F00 in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31	**RESETN**	Active LOW reset signal for all logic in the block. Note that reset is active on all flops in the block when either system reset is asserted (RESET# pin or SYSTEM_POWER.RESETN is asserted) or this signal is active. After setting this bit to 1, firmware will poll and wait for the ‘active’ bit to assert. Reading ‘1’ from ‘resetn’ does not indicate the block is out of reset – this may take some time depending on initialization tasks and clock frequencies.
0	**ACTIVE**	For blocks that must perform initialization after reset before becoming operational, this signal will remain deasserted until initialization is complete. In other words, reading active = 1 indicates block is initialized and ready for operation.

10.10 USB2 HS/FS/LS PHY Registers

10.10.1 PHY_CLK_AND_TEST

USB PHY Clocks and Testability Configuration Register

H

PHY_CLK_AND_TEST	USB PHY Clocks and Testability Configuration Register						0xE0031008
b31	b30	b29	b28	b27	b26	b25	b24
	ON_DCD	**SUSPEND_N**		**RESET**		**VDATSRCEEN**	**CHGRDET**
	R/W	R/W		R/W		R/W	R
	R	R		R		R	R/W
	0	0		1		0	0

PHY_CLK_AND_TEST	USB PHY Clocks and Testability Configuration Register						
b23	b22	b21	b20	b19	b18	b17	b16
CHGRMODE	**CHGRDETEN**	**CHGRDETON**	**IDDIG**	**IDPULLUP**			
R/W	R/W	R/W	R	R/W			
R	R	R	R/W	R			
1	0	0	N	0			

PHY_CLK_AND_TEST	USB PHY Clocks and Testability Configuration Register						
b15	b14	b13	b12	b11	b10	b9	b8

PHY_CLK_AND_TEST	USB PHY Clocks and Testability Configuration Register						
b7	b6	b5	b4	b3	b2	b1	b0
			VLOAD			**ONCLOCK**	**DATABUS16_8**
			R/W			R/W	R/W
			R			R	R
			1			0	1

Bit	Name	Description
30	**ON_DCD**	MIPS PHY Enable Data Contact Detect Circuitry
29	**SUSPEND_N**	Suspend value applied to USB2 PHY (active low)
27	**RESET**	Reset value applied to USB2 PHY
25	**VDATSRCEEN**	Vdat source enable for charger detect
24	**CHGRDET**	MIPS PHY Charger Detector Output (0 = host detected, 1 = charger detected)
23	**CHGRMODE**	MIPS PHY Charger Detector Mode

continued on next page

10.10.1 PHY_CLK_AND_TEST *(continued)*

22	**CHGRDETEN**	MIPS PHY Charger Detector Enable (not tested, used charger detection in CHGDET_CTRL instead)
21	**CHGRDETON**	MIPS PHY Charger Detector Power On Control (not tested)
20	**IDDIG**	MIPS PHY Digital Value of ID line
19	**IDPULLUP**	MIPS PHY Enable Sampling of ID line by PHY
4	**VLOAD**	Vendor Control Register Load Active Low
1	**ONCLOCK**	Enable 480-MHz Clock Output in Suspend
0	**DATABUS16_8**	Data Bus Size 0 RSVD 1 16-bit (only 16-bit mode is tested)

10.10.2 PHY_CONF

USB PHY Programmability and Serial Interface Register

H

PHY_CONF	USB PHY Programmability and Serial Interface Register						0xE003100C
b31	b30	b29	b28	b27	b26	b25	b24

PHY_CONF	USB PHY Programmability and Serial Interface Register						
b23	b22	b21	b20	b19	b18	b17	b16
PREEMDEPTH	**ENPRE**	**FSRFTSEL[1:0]**		**LSRFTSEL[1:0]**		**ICPCTRL[1:0]**	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	1		1		0	0

PHY_CONF	USB PHY Programmability and Serial Interface Register						
b15	b14	b13	b12	b11	b10	b9	b8
HSTEDVSEL[1:0]		**FSTUNEVSEL[2:0]**			**HSDEDVSEL[1:0]**		**HSDRVSLOPE[3]**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
2		4			2		0

PHY_CONF	USB PHY Programmability and Serial Interface Register						
b7	b6	b5	b4	b3	b2	b1	b0
HSDRVSLOPE[2:0]			**HSDRVAMPLITUDE[1:0]**		**HSDRVTIMINGN[1:0]**		**HSDRVTIMINGP**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Bit	Name	Description
23	**PREEMDEPTH**	HS Driver Pre-emphasis Depth
22	**ENPRE**	HS Driver Pre-emphasis Enable
21:20	**FSRFTSEL[1:0]**	FS Driver Rise/Fall Time Control
19:18	**LSRFTSEL[1:0]**	LS Driver Rise/Fall Time Control
17:16	**ICPCTRL[1:0]**	PLL Charge Pump Current Control
15:14	**HSTEDVSEL[1:0]**	Reference Voltage for High Speed Transmission

continued on next page

10.10.2 PHY_CONF *(continued)*

13:11	**FSTUNEVSEL[2:0]**	Reference Voltage Control for Calibration Circuit
10:9	**HSDEDVSEL[1:0]**	Reference Voltage for High Speed Disconnect
8:5	**HSDRVSLOPE[3:0]**	HS Driver Slope Control
4:3	**HSDRVAMPLITUDE[1:0]**	HS Driver Amplitude Control
2:1	**HSDRVTIMINGN[1:0]**	HS NMOS Driver Timing Control
0	**HSDRVTIMINGP**	HS PMOS Driver Timing Control

10.10.3 PHY_CHIRP

USB PHY Chirp Control Register

PHY_CHIRP			USB PHY Chirp Control Register				0xE0031014
b31	b30	b29	b28	b27	b26	b25	b24
	OVERRIDE_FSM						
	R/W						
	R						
	0						

PHY_CHIRP			USB PHY Chirp Control Register				
b23	b22	b21	b20	b19	b18	b17	b16

PHY_CHIRP			USB PHY Chirp Control Register				
b15	b14	b13	b12	b11	b10	b9	b8

PHY_CHIRP			USB PHY Chirp Control Register				
b7	b6	b5	b4	b3	b2	b1	b0
			CHIRP_STATE[4:0]				
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	0	0	0

Bit	Name	Description
30	**OVERRIDE_FSM**	Override chirp state machine
4:0	**CHIRP_STATE[4:0]**	Set chirp state machine state (if OVERRIDE_FSM == 1) 5'h00 FULL_SPEED 5'h01 FULL_SPEED_SUSPEND 5'h02 SWITCH_XCVR_TO_HSPD 5'h03 CHIRP 5'h04 LOOK_FOR_K1 5'h05 LOOK_FOR_J1 5'h06 LOOK_FOR_K2 5'h07 LOOK_FOR_J2 5'h08 LOOK_FOR_K3 5'h09 LOOK_FOR_J3 5'h0A SWITCH_XCVR_TO_FSPD 5'h0B WAIT_END_RESET_FSPD 5'h0D HIGH_SPEED 5'h0E SWITCH_XCVR_TO_FSPD_CHK_RESET 5'h0F CHECK_RESET 5'h10 HIGH_SPEED_SUSPEND 5'h11 WAIT_END_OF_RESUME 5'h12 WAIT_PP_AFTER_RESUME

10.11 USB2 Device Controller Registers

10.11.1 DEV_CS

Device Controller Master Control and Status Register

DEV_CS			Device Controller Master Control and Status Register				0xE0031400
b31	b30	b29	b28	b27	b26	b25	b24
NAKALL					SETUP_CLR_BUSY	TEST_MODE[2:1]	
R/W					R/W1C	R/W	R/W
R					R/W1S	R	R
0					0	0	0

DEV_CS			Device Controller Master Control and Status Register				
b23	b22	b21	b20	b19	b18	b17	b16
TEST_MODE[0]	DEVICEADDR[6:0]						
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_CS			Device Controller Master Control and Status Register				
b15	b14	b13	b12	b11	b10	b9	b8
COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_CS			Device Controller Master Control and Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
ERR_LIMIT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
4							

The hardware automatically retries the USB packet transfer; error recovery is transparent to the firmware. However (ERR_LIMIT,COUNT) count can be used to check for the USB bus errors—CRC, bit stuff, etc., and it can trigger an interrupt when the programmed limit is reached.

NAKALL can be used to temporarily NAK all transactions on the bus while modifying the EP configuration registers.

Bit	Name	Description
31	**NAKALL**	Set this bit to '1', the hardware will NAK all transfers from the host in all endpoint1-31.
26	**SETUP_CLR_BUSY**	Allow device to ACK SETUP data/status phase packets
25:23	**TEST_MODE[2:0]**	USB Test Mode 000 Normal operation 001 Test_J 010 Test_K 011 Test_SE0_NAK 100 Test_Packet [USB 2.0, §7.1.20, p 169; §9.4.9, Table 9−7, p 259]

continued on next page

10.11.1 DEV_CS *(continued)*

22:16	**DEVICEADDR[6:0]**	During the USB enumeration process, the host sends a device a unique 7-bit address, which the USB core copies into this register. The USB Core will automatically respond only to its assigned address. During the USB RESET, this register will be cleared to zero.
15:8	**COUNT[7:0]**	Number of errors detected To clear the error count write 0 to these bits.
7:0	**ERR_LIMIT[7:0]**	Error interrupt limit (COUNT ≥ERR_LIMIT will cause UIB_ERR_INTR.ERRLIMIT interrupt)

10.11.2 DEV_FRAMECNT

FRAMECNT Register

DEV_FRAMECNT			FRAMECNT Register				0xE0031404
b31	b30	b29	b28	b27	b26	b25	b24
DEV_FRAMECNT			**FRAMECNT Register**				
b23	b22	b21	b20	b19	b18	b17	b16
DEV_FRAMECNT			**FRAMECNT Register**				
b15	b14	b13	b12	b11	b10	b9	b8
		FRAMECNT[10:5]					
		R	R	R	R	R	R
		R/W	R/W	R/W	R/W	R/W	R/W
		N/A	N/A	N/A	N/A	N/A	N/A
DEV_FRAMECNT			**FRAMECNT Register**				
b7	b6	b5	b4	b3	b2	b1	b0
FRAMECNT[4:0]					**MICROFRAME[2:0]**		
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

USB SOF and uSOF frame counts USB.

Bit	Name	Description
13:3	**FRAMECNT[10:0]**	Every millisecond the host sends a SOF token indicating “Start Of Frame,” along with an 11-bit incrementing frame count. FX3 copies the frame count into these registers at every SOF. One use of the frame count is to respond to the USB SYNC_FRAME Request. If the USB core detects a missing or garbled SOF, it generates an internal SOF and increments USBFRAMEL-USBRAMEH.
2:0	**MICROFRAME[2:0]**	MICROFRAME contains a count 0-7 which indicates which of the eight 125-microsecond microframes last occurred. This register is active only when FX3 is operating at High Speed (480 Mbps).

10.11.3 DEV_PWR_CS

Power Management Control and Status Register

DEV_PWR_CS	Power Management Control and Status Register						0xE0031408
b31	b30	b29	b28	b27	b26	b25	b24

DEV_PWR_CS	Power Management Control and Status Register						
b23	b22	b21	b20	b19	b18	b17	b16

DEV_PWR_CS	Power Management Control and Status Register						
b15	b14	b13	b12	b11	b10	b9	b8

DEV_PWR_CS	Power Management Control and Status Register						
b7	b6	b5	b4	b3	b2	b1	b0
HSM	**FORCE_FS**		**DEV_SUSPEND**	**DISCON**	**NOSYNSOF**		**SIGRSUME**
R	R/W		R/W	R/W	R/W		R/W
R/W	R		R	R	R		R
0	0		0	1	0		0

Bit	Name	Description
7	**HSM**	If HSM=1, the SIE is operating in High Speed Mode 0-1 transition of this bit causes a HSGRANT interrupt request.
6	**FORCE_FS**	Forces the device controller to enumerate as Full Speed device.
4	**DEV_SUSPEND**	Puts the USB device controller and PHY into suspend mode (pull up connected, drivers, PLLs etc turned off).
3	**DISCON**	Setting this bit to “1” will disconnect hardware from the USB bus by removing the internal 1.5 K pull-up resistor from the D+
2	**NOSYNSOF**	If set to 1, disable synthesizing missing SOFs.
0	**SIGRSUME**	Set SIGRSUME = 1 to drive the “K” state onto the USB bus. This should be done only by a device that is capable of remote wakeup, and then only during the SUSPEND state. To signal RESUME, set SIGRSUME = 1, waits 10-15 ms, then set SIGRSUME = 0.

10.11.4 DEV_SETUPDAT

SETUPDAT0/1 Registers

DEV_SETUPDAT			SETUPDAT0/1 Registers				0xE003140C
b63	b62	b61	b60	b59	b58	b57	b56
SETUP_LENGTH[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_SETUPDAT			SETUPDAT0/1 Registers				
b55	b54	b53	b52	b51	b50	b49	b48
SETUP_LENGTH[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_SETUPDAT			SETUPDAT0/1 Registers				
b47	b46	b45	b44	b43	b42	b41	b40
SETUP_INDEX[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_SETUPDAT			SETUPDAT0/1 Registers				
b39	b38	b37	b36	b35	b34	b33	b32
SETUP_INDEX[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_SETUPDAT			SETUPDAT0/1 Registers				
b31	b30	b29	b28	b27	b26	b25	b24
SETUP_VALUE[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_SETUPDAT			SETUPDAT0/1 Registers				
b23	b22	b21	b20	b19	b18	b17	b16
SETUP_VALUE[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_SETUPDAT			SETUPDAT0/1 Registers				
b15	b14	b13	b12	b11	b10	b9	b8
SETUP_REQUEST[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

continued on next page

10.11.4 DEV_SETUPDAT *(continued)*

DEV_SETUPDAT			SETUPDAT0/1 Registers				
b7	b6	b5	b4	b3	b2	b1	b0
SETUP_REQUEST_TYPE[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

8-byte (2-long word) Storage for USB SETUP data on endpoint0.

Bit	Name	Description
63:48	**SETUP_LENGTH[15:0]**	Setup data field
47:32	**SETUP_INDEX[15:0]**	Setup data field
31:16	**SETUP_VALUE[15:0]**	Setup data field
15:8	**SETUP_REQUEST[7:0]**	Setup data field
7:0	**SETUP_REQUEST_TYPE[7:0]**	Setup data field

10.11.5 DEV_TOGGLE

Data Toggle for Endpoints Register

DEV_TOGGLE			Data Toggle for Endpoints Register				0xE0031414
b31	b30	b29	b28	b27	b26	b25	b24

DEV_TOGGLE			Data Toggle for Endpoints Register				
b23	b22	b21	b20	b19	b18	b17	b16

DEV_TOGGLE			Data Toggle for Endpoints Register				
b15	b14	b13	b12	b11	b10	b9	b8
							TOGGLE_VALID
							R/W0C
							R/W1S
							1

DEV_TOGGLE			Data Toggle for Endpoints Register				
b7	b6	b5	b4	b3	b2	b1	b0
Q	**S**	**R**	**IO**	**ENDPOINT[3:0]**			
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Data toggles when all endpoints reset to ‘0’, on power-up, or on USB reset. In general, hardware maintains all the data toggle bits, which are toggled between data packet transfers. The firmware might need to reset or set data toggle bit only following conditions:

- After a configuration changes (i.e., after the host issues a Set Configuration request).
- After an interface’s alternate setting changes (i.e., after the host issues a Set Interface request).
- After the host sends a Clear Feature - Endpoint Stall request to an endpoint.

To change the value of the toggle bit, software must first write ENDPOINT and IO, then poll for TOGGLE_VALID=1, then write to R/S, then poll again for TOGGLE_VALID=1.

Bit	Name	Description
8	**TOGGLE_VALID**	Indicates Q is valid for selected endpoint, may be polled in software. After writing to R/S, indicates write completion. This bit must be cleared by software to initiate an operation.
7	**Q**	Current value of toggle bit for EP selected in IO/ENDPOINT
6	**S**	Write ‘1’ to set data toggle to ‘1’. When both R and S are set, behavior is undefined.
5	**R**	Write ‘1’ to reset data toggle to ‘0’. When both R and S are set, behavior is undefined.

continued on next page

10.11.5 DEV_TOGGLE *(continued)*

4	**IO**	0	OUT
		1	IN
3:0	**ENDPOINT[3:0]**	Endpoint	

10.11.6 DEV_EPI_CS

IN Endpoint Control and Status Register

There are 16 DEV_EPI_CS registers corresponding to each of the possible IN endpoints. The address of each is calculated as DEV_EPI_CS(x) = 0xE0031418 + (x*0x4). Hence DEV_EPI_CS(0) is at address 0xE0031418, DEV_EPI_CS(1) is at address 0xE0031418 + 0x4 and so on. The definition of each of these is the same.

DEV_EPI_CS			IN Endpoint Control and Status Register				0xE0031418
b31	b30	b29	b28	b27	b26	b25	b24
ISOERR_MASK	SHORT_MASK	ZERO_MASK	DONE_MASK	BNAK_MASK	COMMIT_MASK		
R/W	R/W	R/W	R/W	R/W	R/W		
R	R	R	R	R	R		
0	0	0	0	0	0		

DEV_EPI_CS			IN Endpoint Control and Status Register				
b23	b22	b21	b20	b19	b18	b17	b16
ISOERR	SHORT	ZERO	DONE	BNAK	COMMIT		STALL
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C		R/W
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S		R
0	0	0	0	0	0		0

DEV_EPI_CS				IN Endpoint Control and Status Register			
b15	b14	b13	b12	b11	b10	b9	b8
NAK	VALID	ISOINPKS[1:0]		TYPE[1:0]		PAYLOAD[9:8]	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	1	0	0	0	0		

DEV_EPI_CS				IN Endpoint Control and Status Register			
b7	b6	b5	b4	b3	b2	b1	b0
PAYLOAD[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x40							

Endpoint IN Control and Status:

- Set up USB Package buffering, ISO/BULK/INT, IN/OUT and enable/disable endpoint
- Power up default the payload is 64-byte (Max payload count for Full Speed). In High Speed, the payload can be up to 512 for BULK and 1024 for ISO.

Bit	Name	Description
31	**ISOERR_MASK**	Interrupt mask for ISOERR bit
30	**SHORT_MASK**	Interrupt mask for SHORT bit
29	**ZERO_MASK**	Interrupt mask for ZERO bit
28	**DONE_MASK**	Interrupt mask for DONE bit
27	**BNAK_MASK**	Interrupt mask for BNAK bit
26	**COMMIT_MASK**	Interrupt mask for COMMIT bit

continued on next page

10.11.6 DEV_EPI_CS *(continued)*

23	**ISOERR**	The ISOERR is set when ISO data PIDs arrive out of sequence (applies to high speed only), or when an ISO packet was dropped because no data was available (FS or HS).
22	**SHORT**	Indicates a shorter-than-maxsize packet was received, but UIB_EPI_XFER_CNT did not reach 0.
21	**ZERO**	Indicates a zero-length packet was returned to the host in an IN transaction. Must be cleared by software.
20	**DONE**	Indicates transfer is done (UIB_EPI_XFER_CNT = 0). This bit must be cleared by software.
19	**BNAK**	When the host sends an IN token to any Bulk IN endpoint which does not have data to send, the FX3 automatically NAKs the IN token and asserts this interrupt. Note that this bit will not be set if either the Endpoint NAK or global NAK_ALL bits are set when the NAK is transmitted
18	**COMMIT**	Set whenever an IN token was ACKed by the host.
16	**STALL**	Set this bit to “1” to stall an endpoint, and to “0” to clear a stall.
15	**NAK**	Setting this bit causes NAK on IN transactions.
14	**VALID**	Set VALID = 1 to activate an endpoint, and VALID = 0 to deactivate it. All USB endpoints default to valid. An endpoint whose VALID bit is 0 does not respond to any USB traffic.
13:12	**ISOINPKS[1:0]**	Number of packets to be sent per microframe (aka high-bandwidth mode ISO). For this implementation only EP3 and EP7 support values other than 1. EP3 and EP7 support values 1..3. This field must be 0 for non-ISO endpoints.
11:10	**TYPE[1:0]**	The End Point Type (Control on EP0 only) 00 Control 01 Isochronous 10 Bulk 11 Interrupt
9:0	**PAYLOAD[9:0]**	Max number of bytes transferred for each token 0 Value 0 means 1024. Power up default value is 64.

10.11.7 DEV_EPI_XFER_CNT

IN Endpoint Remaining Transfer Length Register

There are 16 DEV_EPI_XFER_CNT registers. The address of each is calculated as DEV_EPI_XFER_CNT(x) = 0xE0031458 + (x*0x4). Hence DEV_EPI_XFER_CNT(0) is at address 0xE0031458, DEV_EPI_XFER_CNT(1) is at address 0xE0031458 + 0x4 and so on. The definition of each of these is the same.

DEV_EPI_XFER_CNT			IN Endpoint Remaining Transfer Length Register				0xE0031458
b31	b30	b29	b28	b27	b26	b25	b24

DEV_EPI_XFER_CNT			IN Endpoint Remaining Transfer Length Register				
b23	b22	b21	b20	b19	b18	b17	b16
BYTES_REMAINING[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_EPI_XFER_CNT			IN Endpoint Remaining Transfer Length Register				
b15	b14	b13	b12	b11	b10	b9	b8
BYTES_REMAINING[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	1

DEV_EPI_XFER_CNT			IN Endpoint Remaining Transfer Length Register				
b7	b6	b5	b4	b3	b2	b1	b0
BYTES_REMAINING[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

This register counts down the remaining bytes left in the transfer.

Bit	Name	Description
23:0	BYTES_REMAINING[23:0]	Number of bytes remaining in the transfer. This value will never go negative (if more bytes are transferred than remaining in counter, counter value stays at 0). The value in this register is used only for generating the DONE interrupt for the endpoint, and does not affect the actual data transfers.

10.11.8 DEV_EPO_CS

OUT Endpoint Control and Status Register

There are 16 DEV_EPO_CS registers. The address of each is calculated as DEV_EPO_CS(x) = 0xE0031498 + (x*0x4). Hence DEV_EPO_CS(0) is at address 0xE0031498, DEV_EPO_CS(1) is at address 0xE0031498 + 0x4 and so on. The definition of each of these is the same.

DEV_EPO_CS	OUT Endpoint Control and Status Register						0xE0031498
b31	b30	b29	b28	b27	b26	b25	b24
ISOERR_MASK	**SHORT_MASK**	**ZERO_MASK**	**DONE_MASK**	**BNAK_MASK**	**COMMIT_MASK**	**OVF_MASK**	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	
R	R	R	R	R	R	R	
0	0	0	0	0	0	0	

DEV_EPO_CS	OUT Endpoint Control and Status Register						
b23	b22	b21	b20	b19	b18	b17	b16
ISOERR	**SHORT**	**ZERO**	**DONE**	**BNAK**	**COMMIT**	**OVF**	**STALL**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R
0	0	0	0	0	0	0	0

DEV_EPO_CS	OUT Endpoint Control and Status Register							
b15	b14	b13	b12	b11	b10	b9	b8
NAK	**VALID**	**ISOINPKS[1:0]**		**TYPE[1:0]**		**PAYLOAD[9:8]**	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	1	0	0	0	0		

DEV_EPO_CS	OUT Endpoint Control and Status Register							
b7	b6	b5	b4	b3	b2	b1	b0
PAYLOAD[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x40							

Endpoint OUT Control and Status:

- Set up USB Package buffering, ISO/BULK/INT, IN/OUT and enable/disable endpoint
- Power up default the payload is 64-byte (Max payload count for Full Speed). In High Speed, the payload can be up to 512 for BULK and 1024 for ISO.

Bit	Name	Description
31	**ISOERR_MASK**	Interrupt mask for ISOERR bit
30	**SHORT_MASK**	Interrupt mask for SHORT bit
29	**ZERO_MASK**	Interrupt mask for ZERO bit
28	**DONE_MASK**	Interrupt mask for DONE bit
27	**BNAK_MASK**	Interrupt mask for BNAK bit
26	**COMMIT_MASK**	Interrupt mask for COMMIT bit

continued on next page

10.11.8 DEV_EPO_CS *(continued)*

25	**OVF_MASK**	Interrupt mask for OVF bit
23	**ISOERR**	The ISO_ERR is set when ISO data PIDs arrive out of sequence (applies to high speed only), or when an ISO packet was dropped because no data was available (FS or HS).
22	**SHORT**	Indicates a shorter-than-maxsize packet was received, but UIB_EPI_XFER_CNT did not reach 0).
21	**ZERO**	Indicates a zero-length packet was returned to the host in an IN transaction. Must be cleared by software.
20	**DONE**	Indicates transfer is done (UIB_EPI_XFER_CNT = 0). This bit must be cleared by software.
19	**BNAK**	When the host sends a PING/OUT token to any Bulk OUT endpoint, which does not have an empty buffer, the FX3 automatically NAKs the token and asserts this interrupt. Note that this bit will be set if there is no empty buffer at the receipt of the OUT Packet and if neither the Endpoint NAK or global NAK_ALL bits are set when the NAK is transmitted.
18	**COMMIT**	Set whenever device controller ACKs an OUT token.
17	**OVF**	Indicates a packet was received in an OUT token with more bytes than PAYLOAD.
16	**STALL**	Set this bit to “1” to stall an endpoint, and to “0” to clear a stall.
15	**NAK**	Setting this bit causes NAK on OUT and PING transactions.
14	**VALID**	Set VALID = 1 to activate an endpoint, and VALID = 0 to deactivate it. All USB endpoints default to valid. An endpoint whose VALID bit is 0 does not respond to any USB traffic.
13:12	**ISOINPKS[1:0]**	Number of packets to be sent per microframe (aka high-bandwidth mode ISO). For this implementation only EP3 and EP7 support values other than 1. EP3 and EP7 support values 1..3. This field must be 0 for non-ISO endpoints.
11:10	**TYPE[1:0]**	The End Point Type (Control on EP0 only) 00 Control 01 Isochronous 10 Bulk 11 Interrupt
9:0	**PAYLOAD[9:0]**	Max number of bytes transferred for each token 0 Value 0 means 1024. Power up default value is 64.

10.11.9 DEV_EPO_XFER_CNT

OUT Endpoint Remaining Transfer Length Register

There are 16 DEV_EPO_XFER_CNT registers. The address of each is calculated as DEV_EPO_XFER_CNT(x) = 0xE00314D8 + (x*0x4). Hence DEV_EPO_XFER_CNT(0) is at address 0xE00314D8, DEV_EPO_XFER_CNT(1) is at address 0xE00314D8 + 0x4 and so on. The definition of each of these is the same.

DEV_EPO_XFER_CNT			OUT Endpoint Remaining Transfer Length Register				0xE00314D8
b31	b30	b29	b28	b27	b26	b25	b24

DEV_EPO_XFER_CNT			OUT Endpoint Remaining Transfer Length Register				
b23	b22	b21	b20	b19	b18	b17	b16
BYTES_REMAINING[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_EPO_XFER_CNT			OUT Endpoint Remaining Transfer Length Register				
b15	b14	b13	b12	b11	b10	b9	b8
BYTES_REMAINING[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_EPO_XFER_CNT			OUT Endpoint Remaining Transfer Length Register				
b7	b6	b5	b4	b3	b2	b1	b0
BYTES_REMAINING[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

This register counts down the remaining bytes left in the transfer.

Bit	Name	Description
23:0	BYTES_REMAINING[23:0]	Number of bytes remaining in the transfer. This value will never go negative (if more bytes are transferred than remaining in counter, counter value stays at 0).

10.11.10 DEV_CTRL_INTR_MASK

CONTROL Interrupt Mask Register

DEV_CTRL_INTR_MASK			CONTROL Interrupt Mask Register				0xE0031518
b31	b30	b29	b28	b27	b26	b25	b24

DEV_CTRL_INTR_MASK			CONTROL Interrupt Mask Register				
b23	b22	b21	b20	b19	b18	b17	b16

DEV_CTRL_INTR_MASK			CONTROL Interrupt Mask Register				
b15	b14	b13	b12	b11	b10	b9	b8
				STATUS_STAGE			**URESUME**
				R/W			R/W
				R			R
				0			0

DEV_CTRL_INTR_MASK			CONTROL Interrupt Mask Register				
b7	b6	b5	b4	b3	b2	b1	b0
ERRLIMIT	**SUDAV**	**SUTOK**	**HSGRANT**	**URESET**	**SUSP**	**SOF**	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	
R	R	R	R	R	R	R	
0	0	0	0	0	0	0	

This register is used to mask/unmask the interrupt sources from the DEV_CTRL_INTR register. Setting any of the bits enables the corresponding interrupt source.

Bit	Name	Description
11	**STATUS_STAGE**	Set when host completes Status Stage of a Control Transfer
8	**URESUME**	Set when the host has initiated USB RESUME (>2.5 µs K state on bus)
7	**ERRLIMIT**	USB Error limit detect from UIB_DEV_CS (COUNT ≥ ERR_LIMIT)
6	**SUDAV**	Set when a valid SETUP token and data is received. SETUP data is available from UIB_DEV_SETUPDAT.
5	**SUTOK**	Set whenever a (valid of invalid) SETUP token is received
4	**HSGRANT**	Set when the host grants high speed communications.
3	**URESET**	Set when the host has initiated USB RESET (2.5 µs single ended 0 on bus)
2	**SUSP**	Set when the host suspends the USB bus (USB SUSPEND)
1	**SOF**	Set whenever a SOF occurs

10.11.11 DEV_CTRL_INTR

CONTROL Interrupt Request Register

DEV_CTRL_INTR_MASK			CONTROL Interrupt Request Register				0xE003151C
b31	b30	b29	b28	b27	b26	b25	b24

DEV_CTRL_INTR_MASK			CONTROL Interrupt Request Register				
b23	b22	b21	b20	b19	b18	b17	b16

DEV_CTRL_INTR_MASK			CONTROL Interrupt Request Register				
b15	b14	b13	b12	b11	b10	b9	b8
				STATUS_STAGE			**URESUME**
				R/W1C			R/W1C
				R/W1S			R/W1S
				0			0

DEV_CTRL_INTR_MASK			CONTROL Interrupt Request Register				
b7	b6	b5	b4	b3	b2	b1	b0
ERRLIMIT	**SUDAV**	**SUTOK**	**HSGRANT**	**URESET**	**SUSP**	**SOF**	
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	
0	0	0	0	0	0	0	

This register provides the interrupt status for various USB 2.0 related interrupt sources.

Bit	Name	Description
11	**STATUS_STAGE**	Set when host completes Status Stage of a Control Transfer
8	**URESUME**	Set when the host has initiated USB RESUME (>2.5 µs K state on bus)
7	**ERRLIMIT**	USB Error limit detect from UIB_DEV_CS (COUNT ≥ ERR_LIMIT)
6	**SUDAV**	Set when a valid SETUP token and data is received. SETUP data is available from UIB_DEV_SETUPDAT.
5	**SUTOK**	Set whenever a (valid of invalid) SETUP token is received
4	**HSGRANT**	Set when the host grants high speed communications.
3	**URESET**	Set when the host has initiated USB RESET (2.5 µs single ended 0 on bus)
2	**SUSP**	Set when the host suspends the USB (USB SUSPEND)
1	**SOF**	Set whenever a SOF occurs

10.11.12 DEV_EP_INTR_MASK

USB EP Interrupt Mask Register

DEV_EP_INTR_MASK			USB EP Interrupt Mask Register				0xE0031520
b31	b30	b29	b28	b27	b26	b25	b24
EP_OUT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
DEV_EP_INTR_MASK			**USB EP Interrupt Mask Register**				
b23	b22	b21	b20	b19	b18	b17	b16
EP_OUT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
DEV_EP_INTR_MASK			**USB EP Interrupt Mask Register**				
b15	b14	b13	b12	b11	b10	b9	b8
EP_IN[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
DEV_EP_INTR_MASK			**USB EP Interrupt Mask Register**				
b7	b6	b5	b4	b3	b2	b1	b0
EP_IN[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

USB Endpoint 0-31 interrupt masks.

Bit	Name	Description
31:16	**EP_OUT[15:0]**	Bit <16+x> masks any interrupt from EPO_CS[x]. 1 Enable Interrupt 0 Mask (disable) Interrupt
15:8	**EP_IN[15:0]**	Bit <x> masks any interrupt from EPI_CS[x] 1 Enable Interrupt 0 Mask (disable) Interrupt

10.11.13 DEV_EP_INTR

USB EP Interrupt Request Register

DEV_EP_INTR			USB EP Interrupt Request Register				0xE0031524
b31	b30	b29	b28	b27	b26	b25	b24
EP_OUT[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_EP_INTR			USB EP Interrupt Request Register				
b23	b22	b21	b20	b19	b18	b17	b16
EP_OUT[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_EP_INTR			USB EP Interrupt Request Register				
b15	b14	b13	b12	b11	b10	b9	b8
EP_IN[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

DEV_EP_INTR			USB EP Interrupt Request Register				
b7	b6	b5	b4	b3	b2	b1	b0
EP_IN[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

USB Endpoint 0-31 interrupt status. The status bit indicates that at least one of the interrupts enabled by the DEV_EPI_CS/ DEV_EPO_CS register is active. The specific interrupt source should be identified by reading the DEV_EPI_CS/ DEV_EPO_CS register.

Bit	Name	Description
31:16	**EP_OUT[15:0]**	Bit <16+x> is set if any interrupts in EPO_CS[x] are active
15:8	**EP_IN[15:0]**	Bit <x> is set if any interrupt in EPI_CS[x] are active

10.12 USB Controller Miscellaneous Registers

10.12.1 CHGDET_CTRL

Charger Detect Control and Configuration Register

CHGDET_CTRL			Charger Detect Control and Configuration Register				0xE0031800
b31	b30	b29	b28	b27	b26	b25	b24
PHY_CHARGER_DETECT_EN					ACA_RTRIM[2:0]		
R/W					R/W	R/W	R/W
W					R	R	R
0					0	0	0

CHGDET_CTRL			Charger Detect Control and Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16
ACA_ADC_OUT[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
N/A							

CHGDET_CTRL			Charger Detect Control and Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8
	CARKIT	ACA_CONN_MODE	ACA_ENABLE	ACA_OTG_ID_VALUE[2:0]			
	R/W	R/W	R/W	R	R	R	
	R	R	R	R/W	R/W	R/W	
	0	0	0	N/A			

CHGDET_CTRL			Charger Detect Control and Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
		ACA_RTRIM_OVERRIDE	ACA_POLL_INTERVAL[3:0]				PHY_CHG_DETECTED
		R/W	R/W	R/W	R/W	R/W	R
		R	R	R	R	R	R/W
		0	0	0	0	0	0

Bit	Name	Description
31	**PHY_CHARGER_DETECT_EN**	PHY Charger Detection Enable
26:24	**ACA_RTRIM[2:0]**	ACA Comparison Resistor Trim
23:16	**ACA_ADC_OUT[7:0]**	ACA ADC Output

continued on next page

10.12.1 CHGDET_CTRL *(continued)*

14	**CARKIT**	Carkit Adaptor Enable 0 USB Mode (normal USB operation) 1 USB PHY UART mode In UART mode D+/D- are routed (digitally) to carkit UART signals (P-Port pads or LPP UART pads as selected by GCTL_IOMATRIX).
13	**ACA_CONN_MODE**	Charger Connection Mode 0 Standard ACA/Charger 1 Motorola Enhanced Mini-USB (EMU) Requirements
12	**ACA_ENABLE**	Enable ACA OTG ID Detection 0 Disabled 1 Enabled
11:9	**ACA_OTG_ID_VALUE[2:0]**	Decoded OTG ID Value If ACA_CONN_MODE == 0 --> Standard ACA/Charger Mode 000 OTG 1.3 B-Device (RID_GND: 0...1kΩ) 001 OTG 1.3 A-Device (RID_FLOAT_CHG: >220 kΩ) 010 ACA A-Device (RID_A_CHG: 119...13 2 kΩ) 011 ACA B-Device (RID_B_CHG: 65...72 kΩ) 100 ACA C-Device (RID_C_CHG: 35...39 kΩ) If ACA_CONN_MODE == 1 --> Motorola EMU Mode 000 OTG 1.3 B-Device (RID_GND: 0...1 kΩ) 001 OTG 1.3 A-Device (RID_FLOAT_CHG: >1000 kΩ) 010 MPX.200 VPA (RPROP_ID1: <10.1 kΩ) 011 Non-Intelligent Charging Device (RPROP_ID2: 101...103 kΩ) 100 Mid-Rate Charger (RPROP_ID3: 198...202 kΩ) 101 Fast Charger (RPROP_ID4: 435.6...444.4 kΩ) [Battery Charging Specification: Table 5-3, p 29] [Motorola Enhanced Mini-USB (EMU) Requirements: §4.1.2, Table 1, p 9; §4.2.3, Table 4, p 12; §4.2.6, p 15...16; Appendix A, p 24...25] [ll65aca25 BROS 001-47035*D: Tables 5 & 6, p 32]]
5	**ACA_RTRIM_OVERRIDE**	ACA Comparison Resistor Trim Override
4:1	**ACA_POLL_INTERVAL[3:0]**	ACA OTG ID Polling Interval in 16ms increments, 0000 = 16ms
0	**PHY_CHG_DETECTED**	PHY USB Charger Present

10.12.2 CHGDET_INTR

Charger Detect Interrupt Register

CHGDET_INTR			Charger Detect Interrupt Register				0xE0031804
b31	b30	b29	b28	b27	b26	b25	b24

CHGDET_INTR			Charger Detect Interrupt Register				
b23	b22	b21	b20	b19	b18	b17	b16

CHGDET_INTR			Charger Detect Interrupt Register				
b15	b14	b13	b12	b11	b10	b9	b8

CHGDET_INTR			Charger Detect Interrupt Register				
b7	b6	b5	b4	b3	b2	b1	b0
						CHG_DET_ CHANGE	**OTG_ID_CHANGE**
						R/W1C	R/W1C
						W1S	W1S
						0	0

Bit	Name	Description
1	**CHG_DET_CHANGE**	USB Charger Detect Change Interrupt
0	**OTG_ID_CHANGE**	OTG ID Change Interrupt Indicates that the decoded value of the USB OTG ID signal has changed.

10.12.3 CHGDET_INTR_MASK

Charger Detect Interrupt Mask Register

CHGDET_INTR_MASK	Charger Detect Interrupt Mask Register						0xE0031808
b31	b30	b29	b28	b27	b26	b25	b24

CHGDET_INTR_MASK	Charger Detect Interrupt Mask Register						
b23	b22	b21	b20	b19	b18	b17	b16

CHGDET_INTR_MASK	Charger Detect Interrupt Mask Register						
b15	b14	b13	b12	b11	b10	b9	b8

CHGDET_INTR_MASK	Charger Detect Interrupt Mask Register						
b7	b6	b5	b4	b3	b2	b1	b0
						CHG_DET_CHANGE	**OTG_ID_CHANGE**
						R/W	R/W
						R	R
						0	0

Bit	Name		Description
1	**CHG_DET_CHANGE**	0	Mask interrupt
		1	Report interrupt to higher level
0	**OTG_ID_CHANGE**	0	Mask interrupt
		1	Report interrupt to higher level

10.12.4 OTG_CTRL

OTG Control Register

OTG_CTRL			OTG Control Register				0xE003180C
b31	b30	b29	b28	b27	b26	b25	b24

OTG_CTRL			OTG Control Register				
b23	b22	b21	b20	b19	b18	b17	b16

OTG_CTRL			OTG Control Register				
b15	b14	b13	b12	b11	b10	b9	b8
SSEPM_ENABLE	**SSDEV_ENABLE**	**DEV_ENABLE**	**HOST_ENABLE**	**B_END_SESS**	**B_SESS_VALID**	**A_SESS_VALID**	**DSCHG_VBUS**
R/W	R/W	R/W	R/W	R	R	R	R/W
R	R	R	R	R/W	R/W	R/W	R
0	0	0	0	0	0	0	0

OTG_CTRL			OTG Control Register				
b7	b6	b5	b4	b3	b2	b1	b0
CHG_VBUS	**VBUS_VALID**	**DM**	**DP**	**DP_PD_EN**	**DM_PD_EN**	**DP_PU_EN**	**OTG_ENABLE**
R/W	R	R	R	R/W	R/W	R/W	R/W
R	R/W	R/W	R/W	R	R	R	R
0	0	0	0	0	0	0	0

Bit	Name	Description
15	**SSEPM_ENABLE**	EPM role select 0 EPM connected to high-speed host/device 1 EPM connected to superspeed device This bit is required because SSDEVE_ENABLE and DEV_ENABLE can be used concurrently.
14	**SSDEV_ENABLE**	Enables the Super Speed device function. Behavior depends on settings of HOST_ENABLE, and DEV_ENABLE fields and is defined in the HOST_ENABLE field.
13	**DEV_ENABLE**	Enables the USB 1.1/2.0 device function. Behavior depends on settings of HOST_ENABLE, and SSDEV_ENABLE fields and is defined in the HOST_ENABLE field.

continued on next page

10.12.4 OTG_CTRL *(continued)*

12	**HOST_ENABLE**	Host/Peripheral Role Select, Enables the host function. Complete description depends on fields: DEV_ENABLE, and SSDEV_ENABLE. Combined Behavior is as follows for the following combination of fields: **DEV_ENABLE: HOST_ENABLE: SSDEV_ENABLE** 0: 0: 0: No USB Functionality Enabled 0: 0: 1: USB 3.0 SS Device only is enabled. 0: 1: 0: OTG Host function only is enabled. Both SS and USB 1.1/2.0 device functions are disabled 0: 1: 1: ****ILLEGAL****** 1: 0: 0: USB 1.1/2.0 Device function enabled. SS Device and OTG Host function disabled. 1: 0: 1: USB 1.1/2.0 Device with USB 3.0 SS Device along with Rx Detect functions are allowed. OTG Host Function not allowed 1: 1: 0: ***** ILLEGAL except as intermediate value during switching between 010 and 100 ***** 1: 1: 1: ****ILLEGAL******
11	**B_SESS_VALID**	Vbus B Session End 0 Vbus > 0.8V 1 Vbus < 0.2V
10	**B_SESS_VALID**	Vbus Valid for B Session 0 Vbus < 0.8V 1 Vbus > 4.0V
9	**A_SESS_VALID**	bus Valid for A Session 0 Vbus < 0.8V 1 Vbus > 2.0V
8	**DSCHG_VBUS**	Discharge Vbus
7	**CHG_VBUS**	Charge Vbus
6	**VBUS_VALID**	Vbus Valid 0 Vbus < 4.4V 1 Vbus > 4.7V
5	**DM**	D– line state
4	**DP**	D+ line state
3	**DP_PD_EN**	D+ line state
2	**DM_PD_EN**	D– Pull-down Enable
1	**DP_PU_EN**	D+ Pull-up Enable
0	**OTG_ENABLE**	OTG Enable

10.12.5 OTG_INTR

OTG Interrupt Register

OTG_INTR			OTG Interrupt Register				0xE0031810
b31	b30	b29	b28	b27	b26	b25	b24

OTG_INTR			OTG Interrupt Register				
b23	b22	b21	b20	b19	b18	b17	b16

OTG_INTR			OTG Interrupt Register				
b15	b14	b13	b12	b11	b10	b9	b8

OTG_INTR		OTG Interrupt Register					
b7	b6	b5	b4	b3	b2	b1	b0
		OTG_TIMER_TIMEOUT	**SRP_VBUS_INT**	**SRP_DP_INT**	**B_END_SESS_INT**	**B_SESS_VALID_INT**	**A_SESS_VALID_INT**
		R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
		R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
		0	0	0	0	0	0

Bit	Name	Description
5	**OTG_TIMER_TIMEOUT**	OTG Timer Timeout Interrupt The OTG Timer has reached its terminal count of 0.
4	**SRP_VBUS_INT**	VBUS SRP Interrupt A B-Device initiated SRP by pulsing the VBUS line.
3	**SRP_DP_INT**	D+ SRP Interrupt A B-Device initiated SRP by pulsing the D+ signal.
2	**B_END_SESS_INT**	B_END_SESS Interrupt Set when B_END_SESS goes active
1	**B_SESS_VALID_INT**	B_SESS_VALID Change Interrupt Set when B_SESS_VALID changes.
0	**A_SESS_VALID_INT**	A_SESS_VALID Change Interrupt Set when A_SESS_VALID changes.

10.12.6 OTG_INTR_MASK

OTG Interrupt Mask Register

OTG_INTR_MASK	OTG Interrupt Mask Register						0xE0031814
b31	b30	b29	b28	b27	b26	b25	b24

OTG_INTR_MASK	OTG Interrupt Mask Register						
b23	b22	b21	b20	b19	b18	b17	b16

OTG_INTR_MASK	OTG Interrupt Mask Register						
b15	b14	b13	b12	b11	b10	b9	b8

OTG_INTR_MASK	OTG Interrupt Mask Register						
b7	b6	b5	b4	b3	b2	b1	b0
		OTG_TIMER_TIMEOUT	**SRP_VBUS_INT**	**SRP_DP_INT**	**B_END_SESS_INT**	**B_SESS_VALID_INT**	**A_SESS_VALID_INT**
		R/W	R/W	R/W	R/W	R/W	R/W
		R	R	R	R	R	R
		0	0	0	0	0	0

Bit	Name		Description
5	**OTG_TIMER_TIMEOUT**	1	Report OTG Timer Timeout to higher level
		0	Mask OTG Timer Timeout event
4	**SRP_VBUS_INT**	1	Report VBUS SRP event to higher level
		0	Mask VBUS SRP event
3	**SRP_DP_INT**	1	Report D+ SRP Interrupt to higher level
		0	Mask D+ SRP event
2	**B_END_SESS_INT**	1	Report B_END_SESS Interrupt to higher level
		0	Mask B_END_SESS event
1	**B_SESS_VALID_INT**	1	Report B_SESS_VALID changes to higher level
		0	Mask B_SESS_VALID change event
0	**A_SESS_VALID_INT**	1	Report A_SESS_VALID changes to higher level
		0	Mask A_SESS_VALID change event

10.12.7 OTG_TIMER

OTG Timer Register

OTG_TIMER			OTG Timer Register				0xE0031818
b31	b30	b29	b28	b27	b26	b25	b24
OTG_TIMER_LOAD_VAL[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
OTG_TIMER			**OTG Timer Register**				
b23	b22	b21	b20	b19	b18	b17	b16
OTG_TIMER_LOAD_VAL[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
OTG_TIMER			**OTG Timer Register**				
b15	b14	b13	b12	b11	b10	b9	b8
OTG_TIMER_LOAD_VAL[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
OTG_TIMER			**OTG Timer Register**				
b7	b6	b5	b4	b3	b2	b1	b0
OTG_TIMER_LOAD_VAL[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

General timer register to create OTG timeouts. This counter decrements down at 32-kHz standby clock.

Bit	Name	Description
31:0	**OTG_TIMER_LOAD_VAL[31:0]**	Initial counter value. After OTG_TIMER_LOAD_VAL clocks, OTG_TIMER_TIMEOUT will trigger. Disable counter by writing 0 to this register.

10.13 USB End Point Manager Registers

10.13.1 EEPM_CS

Egress EPM Retry Buffer Status Register

EEPM_CS			Egress EPM Retry Buffer Status				0xE0031C00
b31	b30	b29	b28	b27	b26	b25	b24
EG_EPNUM[3:0]							
R	R	R	R				
R/W	R/W	R/W	R/W				
0	0	0	0				

EEPM_CS			Egress EPM Retry Buffer Status				
b23	b22	b21	b20	b19	b18	b17	b16
		URUN_EP_NUM[3:1]				URUN_REPAIR_TIMEOUT_EN	URUN_REPAIR_EN
		R	R	R	R	R/W	R/W
		R/W	R/W	R/W	R/W	R	R
		0	0	0	0	1	1

EEPM_CS			Egress EPM Retry Buffer Status				
b15	b14	b13	b12	b11	b10	b9	b8
VALID_PACKETS[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

EEPM_CS			Egress EPM Retry Buffer Status				
b7	b6	b5	b4	b3	b2	b1	b0
VALID_PACKETS[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

This register provides the status of the retry buffer in the Egress EPM for debug purposes only. In SuperSpeed mode, the retry buffer is a circular buffer of 16x1KB holding the last packets bursted out. Software is expected to 'unroll' the corresponding socket when a retry buffer is flushed by the protocol layer (based on Protocol Layer error interrupts). In High-Speed mode, there is a single 1-KB retry buffer for each endpoint. The data contained can remain in the buffer indefinitely; there is no need for a software rollback in this case.

Bit	Name	Description
31:28	**EG_EPNUM[3:0]**	Active Endpoint Number
21:18	**URUN_EP_NUM[3:0]**	The Endpoint that the underrun occurred.
17	**URUN_REPAIR_TIMEOUT_EN**	If an under run occurs and the URUN_REPAIR_EN is set and this register bit is set, then EPM will start a timer (16-bit counter). If the repair is not complete after 65535*epm clock, EPM will raise the UIB_INTR.EPM_URUN_TIMEOUT interrupt.

continued on next page

10.13.1 EEPM_CS *(continued)*

16	**URUN_REPAIR_EN**	This bit will repair the EPM whenever there is under run due to SYSTEM EPM will keep on reading the packet from SYSMEM whenever data is ready from SYSMEM. If an underrun occurs, EPM will raise the UIB_INTR.EPM_URUN interrupt.
15:0	**VALID_PACKETS**	Bit vector indicating which of the 16 retry buffer contain a valid packet. In SuperSpeed mode, this buffer functions as a circular buffer trailing packets that can be retried behind the WRITE_PTR. In High-Speed mode, each End Point has 1 retry buffer that may independently valid or invalid. These bits are cleared when the Protocol Layer 'activates' an End Point (as opposed to 'reactivating' it). In SuperSpeed mode all bits are cleared at once, in High-Speed mode only the bit for the Endpoint being activated is cleared.

10.13.2 IEPM_CS

Ingress EPM Control and Status Register

IEPM_CS			Ingress EPM Control and Status				0xE0031C04
b31	b30	b29	b28	b27	b26	b25	b24
		EPM_MUX_RESET	EPM_FLUSH	WRITE_PTR[11:8]			
		R/w	R/w	R	R	R	R
		R	R	R/W	R/W	R/W	R/W
		0	0	0	0	0	0

IEPM_CS			Ingress EPM Control and Status				
b23	b22	b21	b20	b19	b18	b17	b16
WRITE_PTR[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

IEPM_CS			Ingress EPM Control and Status				
b15	b14	b13	b12	b11	b10	b9	b8
				READ_PTR[11:8]			
				R	R	R	R
				R/W	R/W	R/W	R/W
				0	0	0	0

IEPM_CS			Ingress EPM Control and Status				
b7	b6	b5	b4	b3	b2	b1	b0
READ_PTR[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

This register contains the read/write pointers for the ingress buffer and allows popping of entries from the FIFO. It is for debugging purposes only.

Bit	Name	Description
29	**EPM_MUX_RESET**	This will reset the EPM Mux.
28	**EPM_FLUSH**	This will flush both the Egress and Ingress EPM.
27:16	**WRITE_PTR[11:0]**	Write pointer of the ingress buffer.
11:0	**READ_PTR[11:0]**	Read pointer of the ingress buffer.

10.13.3 IEPM_MULT

Ingress EPM MULT Function Control Register

IEPM_MULT				Ingress EPM MULT Function Control Register			0xE0031C08
b31	b30	b29	b28	b27	b26	b25	b24
						MULT_THRSHOLD[10:9]	
						R/W	R/W
						R	R

IEPM_MULT				Ingress EPM MULT Function Control Register			
b23	b22	b21	b20	b19	b18	b17	b16
MULT_THRSHOLD[8:1]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

IEPM_MULT				Ingress EPM MULT Function Control Register			
b15	b14	b13	b12	b11	b10	b9	b8
MULT_ THRSHOLD[0]	MULT_EN[14:8]						
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	0	0	0	0	0	0	0

IEPM_MULT				Ingress EPM MULT Function Control Register			
b7	b6	b5	b4	b3	b2	b1	b0
MULT_EN[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register contains the MULT enables and threshold value used by all the ingress EP except EP0.

Bit	Name	Description
25:15	**MULT_THRSHOLD[10:0]**	This field is used to when evaluate the mult signal from ingress adapter. If number of packet space available in the buffer goes down by this field, then mult signal from adapter will be evaluated and if it is set the original buffer space (number of packets) is added to NUM_PACKETS in the IEPM_ENDPOINT register.
14:0	**MULT_EN[14:0]**	Mult Enable for EP1-15.

10.13.4 EEPM_ENDPOINT

Egress EPM Retry Buffer Status

There are 16 EEPM_ENDPOINT registers. The address of each is calculated as EEPM_ENDPOINT(x) = 0xE0031C40 + (x*0x4). Hence EEPM_ENDPOINT(0) is at address 0xE0031C40, EEPM_ENDPOINT(1) is at address 0xE0031C40 + 0x4 and so on. The definition of each of these is the same.

EEPM_ENDPOINT			Egress EPM Retry Buffer Status				0xE0031C40
b31	b30	b29	b28	b27	b26	b25	b24
SOCKET_FLUSH	EEPM_EP_READY			ZLP	EEPM_BYTE_COUNT[15:13]		
R/W	R			R	R	R	R
R	R/W			R/W	R/W	R/W	R/W
0	0			0	0	0	0

EEPM_ENDPOINT			Egress EPM Retry Buffer Status				
b23	b22	b21	b20	b19	b18	b17	b16
EEPM_BYTE_COUNT[12:5]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

EEPM_ENDPOINT			Egress EPM Retry Buffer Status				
b15	b14	b13	b12	b11	b10	b9	b8
EEPM_BYTE_COUNT[4:0]					PACKET_SIZE[10:8]		
R	R	R	R	R	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R	R	R
0	0	0	0	0	0	0	0

EEPM_ENDPOINT			Egress EPM Retry Buffer Status				
b7	b6	b5	b4	b3	b2	b1	b0
PACKET_SIZE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

These registers contain the status fields for each end point (or mapped stream in SuperSpeed Streams mode). The number of packets available on each end point is computed from the SIZE and BYTE_COUNT of the currently loaded descriptor and the availability of a next descriptor (using the credit counters in the DMA Adapter). If the current buffer is completely full (BYTE_COUNT==SIZE) and the next buffer is already available, the available packet count is (BYTE_COUNT/PACKET_SIZE)+1, otherwise it is round-up integral value of (BYTE_COUNT/PACKET).

Bit	Name	Description
31	SOCKET_FLUSH	This bit will flush the corresponding socket.
30	EEPM_EP_READY	The EPM condition used by USB block
27	ZLP	ZLP present in the current buffer
26:11	EEPM_BYTE_COUNT[15:0]	Number of bytes in the current buffer
10:0	PACKET_SIZE[10:0]	Maximum packet size for this end-point. Typically this value is 1024, 512, 64, 1023 (last 2 for USB2 only).

10.13.5 IEPM_ENDPOINT

Ingress EPM Per Endpoint Control and Status Register

IEPM_ENDPOINT			Ingress EPM Per Endpoint Control and Status Register				0xE0031C80
b31	b30	b29	b28	b27	b26	b25	b24
SOCKET_FLUSH	EOT_EOP						
R/W	R/W						
R	R						
0	0						

IEPM_ENDPOINT			Ingress EPM Per Endpoint Control and Status Register				
b23	b22	b21	b20	b19	b18	b17	b16
	EP_READY	NUM_IN_PACKETS[10:5]					
	R	R	R	R	R	R	R
	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	0	0	0	0	0	0	0

IEPM_ENDPOINT			Ingress EPM Per Endpoint Control and Status Register				
b15	b14	b13	b12	b11	b10	b9	b8
NUM_IN_PACKETS[4:0]					PACKET_SIZE[10:8]		
R	R	R	R	R	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R	R	R
0	0	0	0	0			

IEPM_ENDPOINT			Ingress EPM Per Endpoint Control and Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
PACKET_SIZE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1024							

These register contain the status fields for each end-point (or mapped stream in SuperSpeed Streams mode). The number of packets for which space is available on each end point is computed from the SIZE and BYTE_COUNT of the currently loaded descriptor and the availability of a next descriptor (using the credit counter in the DMA Adapter). If the next descriptor is available, available packet count is Round-down integral value of ((2*SIZE-BYTE_COUNT)/PACKET_SIZE), otherwise it is Round-down integral value of ((SIZE-BYTE_COUNT)/PACKET_SIZE)).

Bit	Name	Description
31	**SOCKET_FLUSH**	This bit will flush the corresponding socket.
30	**EOT_EOP**	Configure end-of-packet signalling to DMA adapter. 0 Send EOP at the end of a full packet and EOT for short/zlp packets 1 Send EOT at the end of every packet Setting this bit to 1 is useful for variable size packet endpoints only.
22	**EP_READY**	The EPM condition used by USB block.
21:11	**NUM_IN_PACKETS[10:0**	Number of packets that are guaranteed to fit in the remaining buffer space of the current and next buffers. If the computed number of packets available is larger than 16, this number will be assumed to be 16 in the protocol block.
10:0	**PACKET_SIZE[10:0]**	Maximum packet size for this end-point. Typically this value is 1024, 512, 64, 1023 (last 2 for USB2 only).

10.13.6 IEPM_FIFO

Ingress EPM FIFO Entry Register

IEPM_FIFO			Ingress EPM FIFO Entry Register				0xE0031CC0
b31	b30	b29	b28	b27	b26	b25	b24

IEPM_FIFO			Ingress EPM FIFO Entry Register				
b23	b22	b21	b20	b19	b18	b17	b16
							EP_VALID
							R
							R/W
							0

IEPM_FIFO			Ingress EPM FIFO Entry Register				
b15	b14	b13	b12	b11	b10	b9	b8
IN_EPNUM[3:0]				EOT	BYTES[10:8]		
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

IEPM_FIFO			Ingress EPM FIFO Entry Register				
b7	b6	b5	b4	b3	b2	b1	b0
BYTES[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

This register provides the status of the EP that is on the top of the queue in the Ingress EPM.

Bit	Name	Description
16	**EP_VALID**	Entry is valid
15:12	**IN_EPNUM[3:0]**	Endpoint number for this packet
11	**EOT**	End of Transfer. Set for by the protocol layer short and zero length packets; forwarded to DMA Adapter.
10:0	**BYTES[10:0]**	Number of bytes in the packet

10.14 USB2 Host Controller Registers

10.14.1 HOST_CS

Host Controller Command and Status Bits Register

HOST_CS				Host Controller Command and Status Bits Register			0xE0032000
b31	b30	b29	b28	b27	b26	b25	b24
							DEACTIVATE_ON_IN_EP_SHORT_PKT
							R/W
							R
							1

HOST_CS				Host Controller Command and Status Bits Register			
b23	b22	b21	b20	b19	b18	b17	b16
DISABLE_EHCI_HOSTERR	FRAME_FIT_OFFSET[14:8]						
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

HOST_CS				Host Controller Command and Status Bits Register			
b15	b14	b13	b12	b11	b10	b9	b8
FRAME_FIT_OFFSET[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

HOST_CS				Host Controller Command and Status Bits Register			
b7	b6	b5	b4	b3	b2	b1	b0
0	DEV_ADDR[6:0]						
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Bit	Name	Description	
24	DEACTIVATE_ON_IN_EP_SHORT_PKT	0	In a case where Device is sending a short packet and the total byte count does NOT reach zero, host does not deactivate that EP.
		1	In a case where Device is sending a short packet and the total byte count does NOT reach zero, host deactivates that EP.
23	DISABLE_EHCI_HOSTERR	0	EHCI_INTR.HOST_SYS_ERR_IE works as documented (normal)
		1	HOST_SYS_ERR functionality is disabled
22:8	FRAME_FIT_OFFSET[14:0]	This register is used for frame fit calculation in the host controller. The recommended values are listed in below: EHCI Mode: 212 OHCI (Full Speed) 190 OHCI (Low Speed) 20	
6:0	DEV_ADDR[6:0]	This register contains device address to which host wishes to communicate. Host sends this address to device using set_address command. This address also used by SIE to append this address with different tokens.	

10.14.2 HOST_EP_INTR

Host End Point Interrupt Register

HOST_EP_INTR			Host End Point Interrupt Register				0xE0032004
b31	b30	b29	b28	b27	b26	b25	b24
EPI_IRQ_TOP[15:8]							
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

HOST_EP_INTR			Host End Point Interrupt Register				
b23	b22	b21	b20	b19	b18	b17	b16
EPI_IRQ_TOP[7:0]							
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

HOST_EP_INTR			Host End Point Interrupt Register				
b15	b14	b13	b12	b11	b10	b9	b8
EPO_IRQ_TOP[15:8]							
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

HOST_EP_INTR			Host End Point Interrupt Register				
b7	b6	b5	b4	b3	b2	b1	b0
EPO_IRQ_TOP[7:0]							
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

This register contains interrupt status bit for 32 EPs. CPU reads this register to determine, which EP has triggered interrupt.

Bit	Name	Description
31:16	**EPI_IRQ_TOP[15:0]**	Interrupt Requests for IN endpoints 0..15 when the EP is deactivated by Host Controller.
15:8	**EPO_IRQ_TOP[15:0]**	Interrupt Requests for OUT endpoints 0..15 when the EP is deactivated by Host Controller.

10.14.3 HOST_EP_INTR_MASK

Host End Point Interrupt Mask Register

HOST_EP_INTR_MASK			Host End Point Interrupt Mask Register				0xE0032008
b31	b30	b29	b28	b27	b26	b25	b24
EPI_IRQ_MASK[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
HOST_EP_INTR_MASK			**Host End Point Interrupt Mask Register**				
b23	b22	b21	b20	b19	b18	b17	b16
EPI_IRQ_MASK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
HOST_EP_INTR_MASK			**Host End Point Interrupt Mask Register**				
b15	b14	b13	b12	b11	b10	b9	b8
EPO_IRQ_MASK[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
HOST_EP_INTR_MASK			**Host End Point Interrupt Mask Register**				
b7	b6	b5	b4	b3	b2	b1	b0
EPO_IRQ_MASK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register contains interrupt status bit for 32 EPs. CPU reads this register to determine, which EP has triggered interrupt.

Bit	Name	Description	
31:16	**EPI_IRQ_MASK[15:0]**	1	Report IN endpoint interrupt to CPU
15:8	**EPO_IRQ_MASK[15:0]**	1	Report OUT endpoint interrupt to CPU

10.14.4 HOST_TOGGLE

Data Toggle for Endpoints Register

HOST_TOGGLE			Data Toggle for Endpoints Register				0xE003200C
b31	b30	b29	b28	b27	b26	b25	b24

HOST_TOGGLE			Data Toggle for Endpoints Register				
b23	b22	b21	b20	b19	b18	b17	b16

HOST_TOGGLE				Data Toggle for Endpoints Register			
b15	b14	b13	b12	b11	b10	b9	b8
							VALID
							R/W0C
							R/W1S
							0

HOST_TOGGLE			Data Toggle for Endpoints Register				
b7	b6	b5	b4	b3	b2	b1	b0
Q	**S**	**R**	**IO**	**ENDPOINT[3:0]**			
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Power-up and USB Reset all the endpoints data toggle will reset to ‘0’. In general, hardware maintains all the data toggle bits, which are toggled between data packet transfers. The firmware might need to reset or set data toggle bit only following conditions:

- After a configuration changes (that is, after the host issues a Set Configuration request).
- After an interface’s alternate setting changes (that is, after the host issues a Set Interface request).
- After the host sends a Clear Feature - Endpoint Stall request to an endpoint.

To change the value of the toggle bit, software must first write ENDPOINT and IO, then poll for VALID=1, then write to R or S, then poll again for VALID=1.

Bit	Name	Description
8	**VALID**	Indicates Q is valid for selected endpoint, may be polled in software. After writing to R/S, indicates write completion. This bit must be cleared by software to initiate an operation.
7	**Q**	Current value of toggle bit for EP selected in IO/ENDPOINT
6	**S**	Write ‘1’ to set data toggle to ‘1’ When both R and S are set, behavior is undefined.
5	**R**	Write ‘1’ to reset data toggle to ‘0’. When both R and S are set, behavior is undefined.

continued on next page

10.14.4 DEV_TOGGLE *(continued)*

4	**IO**	0 1	OUT IN
3:0	**ENDPOINT[3:0]**	Endpoint	

10.14.5 HOST_SHDL_CS

Scheduler Memory Pointer Register

HOST_SHDL_CS			Scheduler Memory Pointer Register				0xE0032010
b31	b30	b29	b28	b27	b26	b25	b24

HOST_SHDL_CS			Scheduler Memory Pointer Register				
b23	b22	b21	b20	b19	b18	b17	b16
				ASYNC_SHDL_STATUS	PERI_SHDL_STATUS	PERI_SHDL_CHNG	ASYNC_SHDL_CHNG
				R	R	R/W1S	R/W1S
				R/W	R/W	R/W0C	R/W0C
				0	0	0	0

HOST_SHDL_CS			Scheduler Memory Pointer Register				
b15	b14	b13	b12	b11	b10	b9	b8
BULK_CNTRL_PTR1[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

HOST_SHDL_CS			Scheduler Memory Pointer Register				
b7	b6	b5	b4	b3	b2	b1	b0
BULK_CNTRL_PTR0[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Bit	Name	Description
19	ASYNC_SHDL_STATUS	This bit will inform the software about which portion of the memory should be used for Async/nonperiodic list update both for EHCI and OHCI. 0 Indicates that the scheduler is currently using the lower portion of the scheduler memory for processing the Async/Non-periodic list with the starting location of ASYNC_PTR0 register. 1 Indicates that the scheduler is currently using the upper portion of the scheduler memory for processing the Async/Non-periodic list with the starting location of AYSNC_PTR1 register.
18	PERI_SHDL_STATUS	This bit will inform the software about which portion of the memory should be used for periodic list update both for EHCI and OHCI. 0 Indicates that the scheduler is currently using the lower portion of the scheduler memory for processing the periodic list with the starting location at base address of lower portion of the scheduler memory. 1 Indicates that the scheduler is currently using the upper portion of the scheduler memory for processing the periodic list with the starting location at base address of upper portion of the scheduler memory.
17	PERI_SHDL_CHNG	This bit is set by software to indicate that periodic schedule is changed, and the scheduler may flip to the alternate schedule at the next frame boundary. This bit is cleared by hardware upon switching to new schedule.

continued on next page

10.14.5 HOST_SHDL_CS *(continued)*

16	**ASYNC_SHDL_CHNG**	This bit is set by software to indicate that only the Async schedule is changed, and the scheduler may flip to the alternate schedule at the next microframe boundary. This bit is cleared by hardware upon switching to new schedule.
15:8	**BULK_CNTRL_PTR1[7:0]**	Asynchronous list pointer 1. Indicates the first Async schedule entry number in schedule 1. This pointer is used for the upper portion of the scheduler memory (schedule entry location 96-191).
7:0	**BULK_CNTRL_PTR0[7:0]**	Asynchronous list pointer 0. Indicates the first Async schedule entry number in schedule 0. This pointer is used for the lower portion of the scheduler memory (schedule entry location 0-95).

10.14.6 HOST_SHDL_SLEEP

Scheduler Sleep Register

HOST_SHDL_SLEEP			Scheduler Sleep Register				0xE0032014
b31	b30	b29	b28	b27	b26	b25	b24

HOST_SHDL_SLEEP			Scheduler Sleep Register				
b23	b22	b21	b20	b19	b18	b17	b16

HOST_SHDL_SLEEP			Scheduler Sleep Register				
b15	b14	b13	b12	b11	b10	b9	b8
						ASYNC_SLEEP_TIMMER[8:7]	
						R/W	R/W
						R	R

HOST_SHDL_SLEEP			Scheduler Sleep Register				
b7	b6	b5	b4	b3	b2	b1	b0
ASYNC_SLEEP_TIMMER[6:0]							ASYNC_SLEEP_EN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x12C							

Bit	Name	Description
9:1	ASYNC_SLEEP_TIMMER[8:0]	When the Async list is empty and the ASYNC_SLEEP_EN is set then the scheduler will stop traversing the async list for the amount of ASYNC_SLEEP_TIMMER*sie clock. Per EHCI spec the sleep time should be 10 µs.
0	ASYNC_SLEEP_EN	This bit will enable the sleep feature for the EHCI.

10.14.7 HOST_RESP_BASE

Response Base Address Register

HOST_RESP_BASE				Response Base Address Register			0xE0032018
b31	b30	b29	b28	b27	b26	b25	b24
BASE_ADDRESS[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
HOST_RESP_BASE				**Response Base Address Register**			
b23	b22	b21	b20	b19	b18	b17	b16
BASE_ADDRESS[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
HOST_RESP_BASE				**Response Base Address Register**			
b15	b14	b13	b12	b11	b10	b9	b8
BASE_ADDRESS[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
HOST_RESP_BASE				**Response Base Address Register**			
b7	b6	b5	b4	b3	b2	b1	b0
BASE_ADDRESS[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Bit	Name	Description
31:0	BASE_ADDRESS[31:0]	Base address where scheduler responses are written into memory. Responses are written in order of completion, wrapping at end of buffer (see UIB_HOST_RESP_CS)

10.14.8 HOST_RESP_CS

Scheduler Response Command and Control Register

HOST_RESP_CS	Scheduler Response Command and Control Register						0xE003201C
b31	b30	b29	b28	b27	b26	b25	b24
LIM_ERROR							WR_RESP_COND
R/W0C							R/W
R/W1S							R
0							0

HOST_RESP_CS	Scheduler Response Command and Control Register						
b23	b22	b21	b20	b19	b18	b17	b16
WR_PTR[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

HOST_RESP_CS	Scheduler Response Command and Control Register						
b15	b14	b13	b12	b11	b10	b9	b8
LIMIT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

HOST_RESP_CS	Scheduler Response Command and Control Register						
b7	b6	b5	b4	b3	b2	b1	b0
MAX_ENTRY[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Bit	Name	Description
31	**LIM_ERROR**	Hardware will set this bit as a scheduler response is written with WR_PTR = LIMIT (this indicates and overflow in response memory). Software must clear this bit by writing 0 to it.
24	**WR_RESP_COND**	This is a condition used counting the packet on USB bus. The packet counter is used for the RESP_RATE specified in the scheduler memory. 0 The packet counter increments when the device does not NAK. The response from device could be: ACK, STALL, NYET, PID_ERROR, Data toggle mismatch, CRC16_ERROR, Time-Out. 1 The packet counter increments when the transaction is successful. The successful transaction definition is: IN-Token: No CRC16/PID/PHY Error, No toggle mismatch, No Babble, No STALL, No NYET, No NAK, No Timeout OUT-TOKEN: No CRC16/PID/PHY Error, No STALL, No NAK, No NYET for PING token
23:16	**WR_PTR**	Position at which next schedule response entry will be written (not itself a valid entry).
15:8	**LIMIT**	Response entry which, when written, would constitute an overflow error.
7:0	**MAX_ENTRY**	Maximum number of entries scheduler responses are written into memory. Responses are written in order of completion, wrapping at end of buffer (see UIB_HOST_RESP_CS)

10.14.9 HOST_ACTIVE_EP

Active Endpoint Register

HOST_ACTIVE_EP			Active Endpoint Register				0xE0032020
b31	b30	b29	b28	b27	b26	b25	b24
IN_EP_ACTIVE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
HOST_ACTIVE_EP			**Active Endpoint Register**				
b23	b22	b21	b20	b19	b18	b17	b16
IN_EP_ACTIVE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
HOST_ACTIVE_EP			**Active Endpoint Register**				
b15	b14	b13	b12	b11	b10	b9	b8
OUT_EP_ACTIVE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
HOST_ACTIVE_EP			**Active Endpoint Register**				
b7	b6	b5	b4	b3	b2	b1	b0
OUT_EP_ACTIVE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Bit	Name	Description
31:16	**IN_EP_ACTIVE[15:0]**	This indicates if the IN-EP is active or not. If there is a new schedule entry, this register needs to be Updated after the ASYNC_SHDL_CHNG or PERI_SHDL_CHNG is being set. Software should first clear the corresponding active bit upon HOST_EP_INTR interrupt and then read HOST_EP_DEACTIVATE to clear it.
15:8	**OUT_EP_ACTIVE[15:0]**	This indicates if the OUT-EP is active or not. If there is a new schedule entry, this register needs to be Updated after the ASYNC_SHDL_CHNG or PERI_SHDL_CHNG is being set. Software should first clear the corresponding active bit upon HOST_EP_INTR interrupt and then read HOST_EP_DEACTIVATE to clear it.

10.14.10 OHCI_REVISION

OHCI Host Controller Revision Number Register

OHCI_REVISION			OHCI Host Controller Revision Number Register				0xE0032010
b31	b30	b29	b28	b27	b26	b25	b24
OHCI_REVISION			**OHCI Host Controller Revision Number Register**				
b23	b22	b21	b20	b19	b18	b17	b16
OHCI_REVISION			**OHCI Host Controller Revision Number Register**				
b15	b14	b13	b12	b11	b10	b9	b8
OHCI_REVISION			**OHCI Host Controller Revision Number Register**				
b7	b6	b5	b4	b3	b2	b1	b0
REV[7:0]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0x10							

Bit	Name	Description
7:0	**REV[7:0]**	Revision

10.14.11 OHCI_CONTROL

Host Controller Operating Mode Control Register

OHCI_CONTROL			Host Controller Operating Mode Control Register				0xE0032028
b31	b30	b29	b28	b27	b26	b25	b24

OHCI_CONTROL			Host Controller Operating Mode Control Register				
b23	b22	b21	b20	b19	b18	b17	b16

OHCI_CONTROL			Host Controller Operating Mode Control Register				
b15	b14	b13	b12	b11	b10	b9	b8

OHCI_CONTROL			Host Controller Operating Mode Control Register				
b7	b6	b5	b4	b3	b2	b1	b0
HCFS[1:0]		**BLE**	**CLE**	**IE**	**PLE**		
R/W	R/W	R/W	R/W	R/W	R/W		
R/W	R/W	R	R	R	R		
0	0	0	0	0	0		

Bit	Name	Description
7:6	**HCFS[1:0]**	HostControllerFunctionalState
5	**BLE**	BulkListEnable
4	**CLE**	ControlListEnable
3	**IE**	IsochronousEnable **Note** PLE and IE must both be set to 1 for the periodic list to be enabled. There is no difference in behavior between these two bits.
2	**PLE**	PeriodicListEnable

10.14.12 OHCI_COMMAND_STATUS

Command and Status Register

OHCI_COMMAND_STATUS			Command and Status Register				0xE003202C
b31	b30	b29	b28	b27	b26	b25	b24

OHCI_COMMAND_STATUS			Command and Status Register				
b23	b22	b21	b20	b19	b18	b17	b16
				HC_HALTED	**RS**	**SOC[1:0]**	
				R	R/W	R	R
				R/W	R	R/W	R/W
				1	0	0	0

OHCI_COMMAND_STATUS			Command and Status Register				
b15	b14	b13	b12	b11	b10	b9	b8

OHCI_COMMAND_STATUS			Command and Status Register				
b7	b6	b5	b4	b3	b2	b1	b0

Bit	Name	Description
19	**HC_HALTED**	This bit is a zero whenever the Run/Stop bit is a one. The HC sets this bit to one after it has stopped executing as a result of the Run/Stop bit being set to 0 by software.
18	**RS**	Run/Stop (Replacing the OwnerShipChangeRequest) 0 Stop 1 Run
17:16	**SOC**	SchedulingOverrunCount

10.14.13 OHCI_INTERRUPT_STATUS

OHCI Host Controller Interrupt Status Register

OHCI_INTERRUPT_STATUS			OHCI Host Controller Interrupt Status Register				0xE0032030
b31	b30	b29	b28	b27	b26	b25	b24

OHCI_INTERRUPT_STATUS			OHCI Host Controller Interrupt Status Register				
b23	b22	b21	b20	b19	b18	b17	b16

OHCI_INTERRUPT_STATUS			OHCI Host Controller Interrupt Status Register				
b15	b14	b13	b12	b11	b10	b9	b8

OHCI_INTERRUPT_STATUS			OHCI Host Controller Interrupt Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
	RHSC	**FNO**		**SF**	**SF**		**SO**
	R	R/W1C		R/W1C	R/W1C		R/W1C
	R	R/W1S		R/W1S	R/W1S		R/W1S
	0	0		0	0		0

Bit	Name	Description
6	**RHSC**	RootHubStatusChange
5	**FNO**	FrameNumberOverflow
3	**RD**	ResumeDetected
2	**SF**	StartofFrame
0	**SO**	SchedulingOverrun

10.14.14 OHCI_INTERRUPT_ENABLE

OHCI Interrupt Enable Register

OHCI_INTERRUPT_ENABLE			OHCI Interrupt Enable Register				0xE0032034
b31	b30	b29	b28	b27	b26	b25	b24
MIE							
R/W1S							
R							
0							

OHCI_INTERRUPT_ENABLE			OHCI Interrupt Enable Register				
b23	b22	b21	b20	b19	b18	b17	b16

OHCI_INTERRUPT_ENABLE			OHCI Interrupt Enable Register				
b15	b14	b13	b12	b11	b10	b9	b8

OHCI_INTERRUPT_ENABLE			OHCI Interrupt Enable Register				
b7	b6	b5	b4	b3	b2	b1	b0
	RHSC	**FNO**		**SF**	**SF**		**SO**
	R/W1S	R/W1S		R/W1S	R/W1S		R/W1S
	R	R		R	R		r
	0	0		0	0		0

This register can be used to enable any of the OHCI interrupts. To clear an interrupt enable bit, write to UIB_OHCI_INTERRUPT_DISABLE.

Bit	Name	Description
31	**MIE**	Master Interrupt Enable
6	**RHSC**	RootHubStatusChange
5	**FNO**	FrameNumberOverflow
3	**RD**	ResumeDetected
2	**SF**	StartofFrame
0	**SO**	SchedulingOverrun

10.14.15 OHCI_INTERRUPT_DISABLE

OHCI Interrupt Disable Register

OHCI_INTERRUPT_DISABLE			OHCI Interrupt Disable Register				0xE0032038
b31	b30	b29	b28	b27	b26	b25	b24
MIE							
R/W1C							
R							
0							

OHCI_INTERRUPT_DISABLE			OHCI Interrupt Disable Register				
b23	b22	b21	b20	b19	b18	b17	b16

OHCI_INTERRUPT_DISABLE			OHCI Interrupt Disable Register				
b15	b14	b13	b12	b11	b10	b9	b8

OHCI_INTERRUPT_DISABLE			OHCI Interrupt Disable Register				
b7	b6	b5	b4	b3	b2	b1	b0
	RHSC	**FNO**		**SF**	**SF**		**SO**
	R/W1C	R/W1C		R/W1C	R/W1C		R/W1C
	R	R		R	R		R
	0	0		0	0		0

This register can be used to disable any of the OHCI interrupts. To clear an interrupt enable bit, write to UIB_OHCI_INTERRUPT_ENABLE.

Bit	Name	Description
31	**MIE**	Master Interrupt Enable
6	**RHSC**	RootHubStatusChange
5	**FNO**	FrameNumberOverflow
3	**RD**	ResumeDetected
2	**SF**	StartofFrame
0	**SO**	SchedulingOverrun

10.14.16 OHCI_FM_INTERVAL

OHCI Frame Control Information Register

OHCI_FM_INTERVAL			OHCI Frame Control Information Register				0xE003203C
b31	b30	b29	b28	b27	b26	b25	b24
FIT	**FSMPS[14:8]**						
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0							

OHCI_FM_INTERVAL			OHCI Frame Control Information Register				
b23	b22	b21	b20	b19	b18	b17	b16
FSMPS[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x27F0							

OHCI_FM_INTERVAL			OHCI Frame Control Information Register				
b15	b14	b13	b12	b11	b10	b9	b8
	FI[14:8]						
	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	R	R	R	R	R	R	R

OHCI_FM_INTERVAL			OHCI Frame Control Information Register				
b7	b6	b5	b4	b3	b2	b1	b0
FI[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x752F							

Bit	Name	Description
31	**FIT**	FrameIntervalToggle
30:16	**FSMPS[14:0]**	FSLargestDataPacket
14:0	**FI[14:0]**	FrameInterval

10.14.17 OHCI_FM_REMAINING

Current Value of Remaining Frame Count Register

OHCI_FM_REMAINING			Current Value of Remaining Frame Count Register				0xE0032040
b31	b30	b29	b28	b27	b26	b25	b24
FRT							
R							
R/W							
0							

OHCI_FM_INTERVAL			OHCI Frame Control Information Register				
b23	b22	b21	b20	b19	b18	b17	b16

OHCI_FM_INTERVAL			OHCI Frame Control Information Register				
b15	b14	b13	b12	b11	b10	b9	b8
	FR[14:8]						
	R	R	R	R	R	R	R
	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	0	0	0	0	0	0	0

OHCI_FM_INTERVAL			OHCI Frame Control Information Register				
b7	b6	b5	b4	b3	b2	b1	b0
FR[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Name	Description
31	**FRT**	FrameRemainingToggle
14:0	**FR[14:0]**	FrameRemaining

10.14.18 OHCI_FM_NUMBER

Full Speed Frame Number Register

OHCI_FM_NUMBER			Full Speed Frame Number Register				0xE0032044
b31	b30	b29	b28	b27	b26	b25	b24

OHCI_FM_NUMBER			Full Speed Frame Number Register				
b23	b22	b21	b20	b19	b18	b17	b16

OHCI_FM_NUMBER			Full Speed Frame Number Register				
b15	b14	b13	b12	b11	b10	b9	b8
FN[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

OHCI_FM_NUMBER			Full Speed Frame Number Register				
b7	b6	b5	b4	b3	b2	b1	b0
FN[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Name	Description
15:0	**FN[15:0]**	FrameNumber

10.14.19 OHCI_PERIODIC_START

Periodic Schedule Start Register

OHCI_PERIODIC_START			Periodic Schedule Start Register				0xE0032048
b31	b30	b29	b28	b27	b26	b25	b24
OHCI_PERIODIC_START			**Periodic Schedule Start Register**				
b23	b22	b21	b20	b19	b18	b17	b16
OHCI_PERIODIC_START			**Periodic Schedule Start Register**				
b15	b14	b13	b12	b11	b10	b9	b8
PS[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
OHCI_PERIODIC_START			**Periodic Schedule Start Register**				
b7	b6	b5	b4	b3	b2	b1	b0
PS[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x6977							

Value indicating time where HC should start executing periodic schedule.

Bit	Name	Description
15:0	**PS[15:0]**	PeriodicStart The Default is: 90% * FI = 90% * 0x752F = 0x6977

10.14.20 OHCI_LS_THRESHOLD

LSTHRESHOLD Register

OHCI_LS_THRESHOLD			LSTHRESHOLD Register				0xE003204C
b31	b30	b29	b28	b27	b26	b25	b24

OHCI_LS_THRESHOLD			LSTHRESHOLD Register				
b23	b22	b21	b20	b19	b18	b17	b16

OHCI_LS_THRESHOLD			LSTHRESHOLD Register				
b15	b14	b13	b12	b11	b10	b9	b8
				LST[11:8]			
				R/W	R/W	R/W	R/W
				R	R	R	R

OHCI_LS_THRESHOLD			LSTHRESHOLD Register				
b7	b6	b5	b4	b3	b2	b1	b0
LST[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x628							

Value which is compared to the FrameRemaining field prior to initiating a Low Speed.

Bit	Name	Description
11:0	**LST[11:0]**	LSThreshold

10.14.21 OHCI_RH_PORT_STATUS

Root Hub Port Status Register

OHCI_RH_PORT_STATUS			Root Hub Port Status Register				0xE0032054
b31	b30	b29	b28	b27	b26	b25	b24
PORT_RESUME_FW	**OC**						
R/W	R						
R/W	R						
0	0						

OHCI_RH_PORT_STATUS			Root Hub Port Status Register				
b23	b22	b21	b20	b19	b18	b17	b16
			PRSC		**PSSC**	**PESC**	**CSC**
			R/W1C		R/W1C	R/W1C	R/1C
			R/W1S		R/W1S	R/W1S	R/W
			0		0	0	0

OHCI_RH_PORT_STATUS			Root Hub Port Status Register				
b15	b14	b13	b12	b11	b10	b9	b8

OHCI_RH_PORT_STATUS			Root Hub Port Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
			PRS		**PSS**	**PES**	**CCS**
			R/W1S		R/W1S	R/W1S	R
			R/W0C		R/W0C	R/W	R/W
			0		0	0	0

Bit	Name	Description
31	**PORT_RESUME_FW**	Port Resume (virtual register of PSS from firmware). For an OHCI resume initiated by the host, firmware clears the PSS bit. However, this bit is not supposed to go low until the resume is complete. The PORT_RESUME_FW bit will give us a way to signal to the Host Logic that it should do a resume and then go clear the PSS bit. There is no real functionality being added, this is only trying to emulate the OHCI interface better.
20	**PRSC**	PortResetStatusChange
18	**PSSC**	PortSuspendStatusChange
17	**PESC**	PortEnableStatusChange
16	**CSC**	ConnectStatusChange

continued on next page

10.14.21 OHCI_RH_PORT_STATUS *(continued)*

4	PRS	(read) PortResetStatus (write) SetPortReset
2	PSS	(read) PortSuspendStatus (write) SetPortSuspend
1	PES	(read) PortEnableStatus (write) SetPortEnable
0	CCS	(read) CurrentConnectStatus (write) ClearPortEnable

10.14.22 OHCI_EOF

OHCI End of Frame Times Register

OHCI_EOF				OHCI End of Frame Times Register			0xE0032058
b31	b30	b29	b28	b27	b26	b25	b24
EOF2[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
OHCI_EOF				**OHCI End of Frame Times Register**			
b23	b22	b21	b20	b19	b18	b17	b16
EPF2[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x0019							
OHCI_EOF				**OHCI End of Frame Times Register**			
b15	b14	b13	b12	b11	b10	b9	b8
EOF1[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
OHCI_EOF				**OHCI End of Frame Times Register**			
b7	b6	b5	b4	b3	b2	b1	b0
EOF1[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x0050							

Bit	Name	Description
31:16	**EOF2[15:0]**	EOF2 Time (default: 10 bit times * 2.5 clocks/bit => 25)
15:0	**EOF1[15:0]**	EOF1 Time (default: 32 bit times * 2.5 clocks/bit => 80)

10.14.23 EHCI_HCCPARAMS

Multiple Mode Control Register

OHCI_RH_PORT_STATUS			Root Hub Port Status Register				0xE003205C
b31	b30	b29	b28	b27	b26	b25	b24

OHCI_RH_PORT_STATUS			Root Hub Port Status Register				
b23	b22	b21	b20	b19	b18	b17	b16

OHCI_RH_PORT_STATUS			Root Hub Port Status Register				
b15	b14	b13	b12	b11	b10	b9	b8

OHCI_RH_PORT_STATUS			Root Hub Port Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
ISO_SHDL_THR[3:0]					ASYNC_PARK_CAP		ADDR_64_BIT_CAP
R	R	R	R		R/W		R
R	R	R	R		R		R
0	0	0	0		0		0

Multiple mode control (time-base bit functionality), addressing capability.

Bit	Name	Description
7:4	**ISO_SHDL_THR[3:0]**	Isochronous Scheduling Threshold (In this implementation the scheduler will cache 1 micro-frame worth of data. Appropriate value will be programmed to ensure correct functionality).
2	**ASYNC_PARK_CAP**	Asynchronous Schedule Park Capability
0	**ADDR_64_BIT_CAP**	64-bit Addressing Capability

10.14.24 EHCI_USBCMD

EHCI Command Register

OHCI_RH_PORT_STATUS			Root Hub Port Status Register				0xE0032060
b31	b30	b29	b28	b27	b26	b25	b24

OHCI_RH_PORT_STATUS			Root Hub Port Status Register				
b23	b22	b21	b20	b19	b18	b17	b16
INT_THRESHOLD_CTRL[7:0]							
R/w	R/w	R/w	R/w	R/w	R/w	R/w	R/w
R	R	R	R	R	R	R	R
8							

OHCI_RH_PORT_STATUS			Root Hub Port Status Register				
b15	b14	b13	b12	b11	b10	b9	b8
				ASYNC_SHDL_PRK_EN		ASYNC_SHDL_PRK_CNT[1:0]	
				R/w		R/w	R/w
				R		R	R
				0		0	0

OHCI_RH_PORT_STATUS			Root Hub Port Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
		ASYNC_SHDL_EN	PER_SHDL_EN				RS
		R/w	R/w				R/W
		R	R				R/W
		0	0				0

Bit	Name	Description
23:16	INT_THRESHOLD_CTRL[7:0]	Interrupt Threshold Control
11	ASYNC_SHDL_PRK_EN	Asynchronous Schedule Park Mode Enable
9:8	ASYNC_SHDL_PRK_CNT[1:0]	Asynchronous Schedule Park Mode Count
5	ASYNC_SHDL_EN	Asynchronous Schedule Enable
4	PER_SHDL_EN	Periodic Schedule Enable
0	RS	Run/Stop

10.14.25 EHCI_USBSTS

Host Controller States and Pending Interrupts Register

EHCI_USBSTS			Host Controller States and Pending Interrupts Register				0xE0032064
b31	b30	b29	b28	b27	b26	b25	b24

EHCI_USBSTS			Host Controller States and Pending Interrupts Register				
b23	b22	b21	b20	b19	b18	b17	b16

EHCI_USBSTS			Host Controller States and Pending Interrupts Register				
b15	b14	b13	b12	b11	b10	b9	b8
ASYNC_SHDL_ST	**PER_SHDL_ST**	**RECLAMATION**	**HC_HALTED**				
R	R	R	R				
R/w	R/W	R/W	R/W				
0	0	0	1				

EHCI_USBSTS			Host Controller States and Pending Interrupts Register				
b7	b6	b5	b4	b3	b2	b1	b0
			HOST_SYS_ERR		**PORT_CHNG_DET**	**USBERRINT**	**USBINT**
			R/W1C		R/W1C	R/W1C	R/W1C
			R/W1S		R/W1S	R/W1S	R/W1S
			0		0	0	0

Pending interrupts and various states of the Host Controller

Bit	Name	Description
15	**ASYNC_SHDL_ST**	Asynchronous Schedule Status
14	**PER_SHDL_ST**	Periodic Schedule Status
13	**RECLAMATION**	Reclamation
12	**HC_HALTED**	This bit is a zero whenever the Run/Stop bit is a one. The HC sets this bit to one after it has stopped executing as a result of the Run/Stop bit being set to 0 by software.
4	**HOST_SYS_ERR**	Host System Error. EHCI over scheduling Status
2	**PORT_CHNG_DET**	Port Change Detect
1	**USBERRINT**	USB Error Interrupt
0	**USBINT**	USB Interrupt. **Note** This field does not assert for SETUP+IN(STATUS) qTDs with no data phase. The workaround is to use the HOST_EP_INTR[0] along with the transaction response that gets written into SRAM for HCD.

10.14.26 EHCI_USBINTR

EHCI Interrupt Register

EHCI_USBINTR			EHCI Interrupt Register				0xE0032068
b31	b30	b29	b28	b27	b26	b25	b24

EHCI_USBINTR			EHCI Interrupt Register				
b23	b22	b21	b20	b19	b18	b17	b16

EHCI_USBINTR			EHCI Interrupt Register				
b15	b14	b13	b12	b11	b10	b9	b8

EHCI_USBINTR			EHCI Interrupt Register				
b7	b6	b5	b4	b3	b2	b1	b0
			HOST_SYS_ERR_IE		**PORT_CHANGE_DET_IE**	**USBERRINT_IE**	**USBINT_IE**
			R/W		R/W	R/W	R/W
			R		R	R	R
			0		0	0	0

Bit	Name	Description
4	**HOST_SYS_ERR_IE**	Host System Error Interrupt Enable
2	**PORT_CHANGE_DET_IE**	Port Change Interrupt Enable
1	**USBERRINT_IE**	USB Error Interrupt Enable
0	**USBINT_IE**	USB Interrupt Enable

10.14.27 EHCI_FRINDEX

Frame Index Register

EHCI_FRINDEX			Frame Index Register				0xE003206C
b31	b30	b29	b28	b27	b26	b25	b24

EHCI_FRINDEX			Frame Index Register				
b23	b22	b21	b20	b19	b18	b17	b16

EHCI_FRINDEX			Frame Index Register				
b15	b14	b13	b12	b11	b10	b9	b8
		FRINDEX[13:8]					
		R	R	R	R	R	R
		R/W	R/W	R/W	R/W	R/W	R/W
		0	0	0	0	0	0

EHCI_FRINDEX			Frame Index Register				
b7	b6	b5	b4	b3	b2	b1	b0
FRINDEX[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Name	Description
13:0	FRINDEX[13:0]	The value indicates the Frame on which the scheduler is operating. This value is also used to achieve interrupt endpoint polling duration.

10.14.28 EHCI_CONFIGFLAG

Configure Flag Register

EHCI_CONFIGFLAG			Configure Flag Register				0xE0032070
b31	b30	b29	b28	b27	b26	b25	b24

EHCI_CONFIGFLAG			Configure Flag Register				
b23	b22	b21	b20	b19	b18	b17	b16

EHCI_CONFIGFLAG			Configure Flag Register				
b15	b14	b13	b12	b11	b10	b9	b8

EHCI_CONFIGFLAG			Configure Flag Register				
b7	b6	b5	b4	b3	b2	b1	b0
							CF
							R/W
							R
							0

Bit	Name	Description
0	**CF**	Configure Flag

10.14.29 EHCI_PORTSC

Port Status and Control Register

EHCI_PORTSC				Port Status and Control Register			0xE0032074
b31	b30	b29	b28	b27	b26	b25	b24
PORT_RESET_FW	**PORT_RESUME_HW**						
R/q	R						
R	R/W						
0	0						

EHCI_PORTSC				Port Status and Control Register			
b23	b22	b21	b20	b19	b18	b17	b16

EHCI_PORTSC				Port Status and Control Register			
b15	b14	b13	b12	b11	b10	b9	b8
		PORT_OWNER		**EHCI_LINE_STATE[1:0]**			**PORT_RESET**
		R/W		R	R		R/W1S
		R/W		R/W	R/W		R/W
		1		0	0		0

EHCI_PORTSC				Port Status and Control Register			
b7	b6	b5	b4	b3	b2	b1	b0
PORT_SUSPEND	**F_PORT_RESUME**			**PORT_EN_C**	**PORT_EN**	**PORT_CONNECT_C**	**PORT_CONNECT**
R/W	R/W			R/W1C	R/W0C	R/W1C	R
R/W	R/W			R/W1S	R/W	R/W1S	R/W
0	0			0	0	0	0

Bit	Name	Description
31	**PORT_RESET_FW**	Port Reset (virtual register of PORT_RESET from firmware)
30	**PORT_RESUME_HW**	Hardware Initiated Resume Active
13	**PORT_OWNER**	Port Owner

continued on next page

10.14.29 EHCI_PORTSC *(continued)*

11:10	**EHCI_LINE_STATE[1:0]**	Line Status
8	**PORT_RESET**	Port Reset Firmware sets this bit to initiate Port Reset and subsequently writes a 0 to it to initiate the end of Port Reset. When the Host has completed Port Reset, it will clear this bit. Firmware must poll this bit to determine the end of the Port Reset, and also whether the attached device is High-Speed (PORT_EN == 1).
7	**PORT_SUSPEND**	Suspend
6	**F_PORT_RESUME**	Force Port Resume
3	**PORT_EN_C**	Port Enable/Disable Change
2	**PORT_EN**	Port Enabled/Disabled
1	**PORT_CONNECT_C**	Connect Status Change
0	**PORT_CONNECT**	Current Connect Status

10.14.30 EHCI_EOF

EHCI End of Frame Times Register

EHCI_EOF	EHCI End of Frame Times Register							0xE0032078
b31	b30	b29	b28	b27	b26	b25	b24	
EOF2[15:8]								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
R	R	R	R	R	R	R	R	

EHCI_EOF	EHCI End of Frame Times Register							
b23	b22	b21	b20	b19	b18	b17	b16	
EPF2[7:0]								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
R	R	R	R	R	R	R	R	
0x0004								

EHCI_EOF	EHCI End of Frame Times Register							
b15	b14	b13	b12	b11	b10	b9	b8	
EOF1[15:8]								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
R	R	R	R	R	R	R	R	

EHCI_EOF	EHCI End of Frame Times Register							
b7	b6	b5	b4	b3	b2	b1	b0	
EOF1[7:0]								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
R	R	R	R	R	R	R	R	
0x0023								

Bit	Name	Description
31:16	**EOF2[15:0]**	EOF2 Time (default: 64 bit times / 16 bits/clock -> 4)
15:0	**EOF1[15:0]**	EOF1 Time (default: 560 bit times / 16 bits/clock -> 35)

10.14.31 SHDL_CHNG_TYPE

Scheduler Change Type Register

SHDL_CHNG_TYPE				Scheduler Change Type Register			0xE003207C
b31	b30	b29	b28	b27	b26	b25	b24
EPI_CHNG_TYPE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
SHDL_CHNG_TYPE				**Scheduler Change Type Register**			
b23	b22	b21	b20	b19	b18	b17	b16
EPI_CHNG_TYPE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
SHDL_CHNG_TYPE				**Scheduler Change Type Register**			
b15	b14	b13	b12	b11	b10	b9	b8
EP0_CHNG_TYPE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
SHDL_CHNG_TYPE				**Scheduler Change Type Register**			
b7	b6	b5	b4	b3	b2	b1	b0
EP0_CHNG_TYPE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This register is used along with the HOST_SHDL_CS ASYNC_SHDL_CHNG/PERI_SHDL_CHNG requests.

This register defines when the Scheduler should update its internal active bits with the HOST_ACTIVE_EP register per each EP.

Bit	Name	Description	
31:16	**EOF2[15:0]**	0	Update at Micro-frame boundary
		1	Update at Frame boundary
15:0	**EOF1[15:0]**	0	Update at Micro-frame boundary
		1	Update at Frame boundary

10.14.32 SHDL_STATE_MACHINE

Scheduler State Machine Register

SHDL_STATE_MACHINE	Scheduler State Machine Register						0xE0032080
b31	b30	b29	b28	b27	b26	b25	b24

SHDL_STATE_MACHINE	Scheduler State Machine Register						
b23	b22	b21	b20	b19	b18	b17	b16
					WAIT_EOF	**WAIT_SHDL_EN**	**SCRATCH_WRITE2**
					R	R	R
					R/W	R/W	R/W
					0	0	0

SHDL_STATE_MACHINE	Scheduler State Machine Register						
b15	b14	b13	b12	b11	b10	b9	b8
SCRATCH_WRITE1	**SHDL_WRITE**	**LAST_EVAL**	**WAIT_TP_STUPD**	**EXECUTE**	**ASYNC_SLEEP**	**FIRST_EVAL0**	**READ_EP0_1**
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

SHDL_STATE_MACHINE	Scheduler State Machine Register						
b7	b6	b5	b4	b3	b2	b1	b0
READ_EP0_0	**LOAD_SHDL_MEM**	**SCRATCH_READ2**	**SCRATCH_READ1**	**SCRATCH_READ0**	**FETCH**	**LOAD_PTR**	**IDLE**
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	1

Bit	Name	Description
18	**WAIT_EOF**	Wait for EOF state
17	**WAIT_SHDL_EN**	Wait for scheduler enable state
16	**SCRATCH_WRITE2**	Scratch Write state2
15	**SCRATCH_WRITE1**	Scratch Write state1
14	**SHDL_WRITE**	Scheduler write state
13	**LAST_EVAL**	Last Eval State
12	**WAIT_TP_STUPD**	Wait for TP status state
11	**EXECUTE**	Execute State
10	**ASYNC_SLEEP**	Async Sleep state

continued on next page

10.14.32 SHDL_STATE_MACHINE *(continued)*

9	**FIRST_EVAL**	First Eval State
8	**READ_EP0_1**	Read EP0 state1
7	**READ_EP0_0**	Read EP0 state0
6	**LOAD_SHDL_MEM**	Load scheduler memory state
5	**SCRATCH_READ2**	Scratch Read state2
4	**SCRATCH_READ1**	Scratch Read state1
3	**SCRATCH_READ0**	Scratch Read state0
2	**FETCH**	Fetch State
1	**LOAD_PTR**	Load Pointer State
0	**IDLE**	Idle

10.14.33 SHDL_INTERNAL_STATUS

Scheduler Internal Status Register

SHDL_INTERNAL_STATUS			Scheduler Internal Status Register				0xE0032084
b31	b30	b29	b28	b27	b26	b25	b24

SHDL_INTERNAL_STATUS			Scheduler Internal Status Register				
b23	b22	b21	b20	b19	b18	b17	b16
						TP_EPM_ OVERRUN	TP_EPM_ UNDERRUN
						R	R
						R/W	R/W
						0	0

SHDL_INTERNAL_STATUS			Scheduler Internal Status Register				
b15	b14	b13	b12	b11	b10	b9	b8
TP_TIMEOUT	TP_PID_ERROR	TP_BABBLE	TP_PORT_ERROR	TP_PHY_ERROR	TP_CRC16_ ERROR	TP_DT_ MISMATCH	TP_STALL
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

SHDL_INTERNAL_STATUS			Scheduler Internal Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
TP_NYET	TP_NAK	TP_ACK	EP0_IN	EP0_OUT	EP0_SETUP	FRAME_FIT	EP_DONE
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	1

Bit	Name	Description
17	TP_EPM_OVERRUN	TP EPM over-run
16	TP_EPM_UNDERRUN	TP EPM under-run
15	TP_TIMEOUT	TP timeout
14	TP_PID_ERROR	TP PID error
13	TP_BABBLE	TP Babble
12	TP_PORT_ERROR	TP Port error
11	TP_PHY_ERROR	TP PHY rxerror
10	TP_CRC16_ERROR	TP CRC16 Error
9	TP_DT_MISMATCH	TP DT mismatch

continued on next page

10.14.33 SHDL_INTERNAL_STATUS *(continued)*

8	**TP_STALL**	TP STALL
7	**TP_NYET**	TP NYET
6	**TP_NAK**	TP NAK
5	**TP_ACK**	TP ACK
4	**EP0_IN**	EP0 IN state
3	**EP0_OUT**	EP0 OUT state
2	**EP0_SETUP**	EP0 SETUP state
1	**FRAME_FIT**	Frame Fit
0	**EP_DONE**	EP Done

10.14.34 SHDL_OHCI

Scheduler Memory Register, OHCI Format

There are 64 SHDL_OHCI registers. The address of each is calculated as SHDL_OHCI(x) = 0xE0032400 + (x*0x4). Hence SHDL_OHCI(0) is at address 0xE0032400, SHDL_OHCI(1) is at address 0xE0032400 + 0x4 and so on. The definition of each of these is the same.

SHDL_OHCI			Scheduler Memory Register, OHCI Format				0xE0032400
b95	b94	b93	b92	b91	b90	b89	b88
							IOC_RATE[7]
							R/W
							R
							0

SHDL_OHCI			Scheduler Memory Register, OHCI Format				
b87	b86	b85	b84	b83	b82	b81	b80
IOC_RATE[6:0]							TRNS_MODE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_OHCI			Scheduler Memory Register, OHCI Format				
b79	b78	b77	b76	b75	b74	b73	b72
TOTAL_BYTE_COUNT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_OHCI			Scheduler Memory Register, OHCI Format				
b71	b70	b69	b68	b67	b66	b65	b64
TOTAL_BYTE_COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_OHCI			Scheduler Memory Register, OHCI Format				
b63	b62	b61	b60	b59	b58	b57	b56
EP0_CODE[1:0]		BYPASS_ERROR	MMULT[1:0]		RESP_RATE[7:5]		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_OHCI			Scheduler Memory Register, OHCI Format				
b55	b54	b53	b52	b51	b50	b49	b48
RESP_RATE[4:0]					POLLING_RATE[7:5]		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_OHCI			Scheduler Memory Register, OHCI Format				
b47	b46	b45	b44	b43	b42	b41	b40
POLLING_RATE[4:0]					MAX_PKT_SIZE[10:8]		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.14.34 SHDL_OHCI *(continued)*

SHDL_OHCI	Scheduler Memory Register, OHCI Format						
b39	b38	b37	b36	b35	b34	b33	b32
MAX_PKT_SIZE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_OHCI	Scheduler Memory Register, OHCI Format						
b31	b30	b29	b28	b27	b26	b25	b24
UFRAME_SMASK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_OHCI	Scheduler Memory Register, OHCI Format						
b23	b22	b21	b20	b19	b18	b17	b16
PING	RL[3:0]				MULT[1:0]		ISO_EPM
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_OHCI	Scheduler Memory Register, OHCI Format						
b15	b14	b13	b12	b11	b10	b9	b8
CERR[1:0]		NAK_CNT[3:0]				HALT	T
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_OHCI	Scheduler Memory Register, OHCI Format						
b7	b6	b5	b4	b3	b2	b1	b0
ZPLEN	EPT[1:0]		EPND[4:0]				
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This area exposes the scheduler memory in OHCI format. Please note that the EHCI and OHCI scheduler memories are using the same physical memory, so overwriting one will also overwrite the other. Scheduler memory consists of three sections:

1. Lower Section:
2. Upper Section:
3. Scratch Section:

This register exposes only the Lower and Upper section. The Scratch section is only used by Hardware and it is not accessible by Software. The Lower and Upper sections each consist of 32 End Points and each End Point requires 3 32-bit scheduler memory locations. In result, the MMIO address for lower section would be x400-x57C and for upper portion would be x580-x6FC. Software can update the appropriate portion of the memory based on the PERI_SHDL_STATUS/ ASYNC_SHDL_STATUS bits in the UIB_HOST_SHDL_CS register.

continued on next page

10.14.34 SHDL_OHCI *(continued)*

Bit	Name	Description
127:96	**SHDL_NOT_USED[31:0]**	Not used and not mapped to RAM, only present for alignment reasons
88:81	**IOC_RATE[7:0]**	Should be programmed to zero for OHCI. After IOC_rate*resp_rate the scheduler will issue interrupt at the next interrupt threshold (UIB_EHCI_USBCMD.INT_THRESHOLD_CTRL)
80	**TRNS_MODE**	0 Packet mode 1 Stream mode
79:64	**TOTAL_BYTE_COUNT[15:0]**	Total number of byte count for the transaction.
63:62	**EP0_CODE[1:0]**	This field represents the EP0 type of transaction: 00 Reserved 01 Setup + Data Phase(OUT) + Status Phase(IN) 10 Setup + Data Phase(IN) + Status Phase(OUT) 11 Setup + Status Phase(IN)
61	**BYPASS_ERROR**	This bit will disable any error that would cause an EP to be de-activated. This bit should be used only for Isochronous Endpoint.
60:59	**MMULT[1:0]**	This field is used to expand the MULT field to support the OHCI ControlBulkServiceRatio bandwidth allocation.
58:51	**RESP_RATE[7:0]**	After how many packet the response should be written into SRAM. The packet counter increments based on the condition specified in the UIB_HOST_RESP_CS.WR_RESP_COND.
50:43	**POLLING_RATE[7:0]**	00h Reserved 01h 1 ms (Every frame) 02h 2 ms (Every 2 frames) 04h 4 ms (Every 4 frames) 08h 8 ms (Every 8 frames) 10h 16 ms (Every 16 frames) 20h 32 ms (Every 32 frames) This will be on top of the Interrupt Schedule Mask in EHCI case.
42:32	**MAX_PKT_SIZE[10:0]**	This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). The maximum value this field may contain is 0x400 (1024).
31:24	**UFRAME_SMASK[7:0]**	Should be programmed to zero for OHCI. The host controller uses the value of the three low-order bits of the FRINDEX register as an index into a bit position in this bit vector. If the µFrame S-mask field has a one at the indexed bit position then this queue head is a candidate for transaction execution.
23	**PING**	Should be programmed to zero for OHCI. This field enables the host to issue Ping token when Direction is OUT and it is high-speed. 0 Do not issue ping token for high-speed OUT 1 Do issue ping token for high-speed OUT
22:19	**RL[3:0]**	Should be programmed to zero for OHCI. Nak Count Reload (RL). This field contains a value, which is used by the host controller to reload the Nak Counter field.

continued on next page

10.14.34 SHDL_OHCI *(continued)*

18:17	**MULT[1:0]**	High-Bandwidth Pipe Multiplier. This field is a multiplier used to key the host controller as the number of successive packets the host controller may submit to the endpoint in the current execution. This field should be programmed according to the EPM configuration for that EP. This field will get decremented only if the transfer is successful. If not successful, the original value will be reloaded. 00 Reserved. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per micro-frame 10 Two transactions to be issued for this endpoint per micro-frame 11 Three transactions to be issued for this endpoint per micro-frame
16	**ISO_EPM**	0 For ISO-OUT when EPM is empty for that EP then a ZLP will be issued by TP. For ISO-IN when EPM is full for that EP then and ISO-IN will be issued. 1 For ISO-OUT when EPM is empty for that EP then that entry will be skipped. For ISO-IN when EPM is full for that EP then that entry will be skipped.
15:14	**CERR[1:0]**	This field is a 2-bit down counter that keeps track of the number of consecutive Errors detected while executing this qTD. If this field is programmed with a nonzero value during setup, the Host Controller decrements the count and writes it back to the qTD if the transaction fails. If the counter counts from one to zero, the Host Controller marks the qTD inactive, sets the Halted bit to a one and error status bit for the error that caused CERR to decrement to zero. An interrupt will be generated if the USB Error Interrupt Enable bit in the USBINTR register is set to a one. If HCD programs this field to zero during setup, the Host Controller will not count errors for this qTD and there will be no limit on the retries of this qTD.
13:10	**NAK_CNT[1:0]**	Should be programmed to zero for OHCI. This field is a counter the host controller decrements whenever a transaction for the endpoint associated with this queue head results in a Nak or Nyet response.
9	**HALT**	This will indicate that the EP is halted. Skip this qTD and move to next qTD.
8	**T**	0 Not End of the Period/Asynchronous list 1 End of the Period/Asynchronous list This bit indicates that there are no more valid entries in the current list.
7	**ZPLEN**	This field is used only for non-ISO endpoints. 0 Scheduler will set the active bit to zero when the total byte count reaches zero. 1 Scheduler will set the active bit to zero when the total byte count is zero and a ZLP is sent/received.
6:5	**EPT[1:0]**	The End Point Type. 00 Control 01 Isochronous 10 Bulk 11 Interrupt
4:0	**EPND[4:0]**	[3:0]: EP number, Values: 0-15 [4]: EP Direction, 1: OUT, 0: IN The direction bit is not used for EP0. Scheduler uses the EP0_code to determine the direction of the EP0 transaction.

10.14.35 SHDL_EHCI

Scheduler Memory Register, EHCI Format

There are 64 SHDL_EHCI registers. The address of each is calculated as SHDL_EHCI(x) = 0xE0032800 + (x*0x4). Hence SHDL_EHCI(0) is at address 0xE0032800, SHDL_EHCI(1) is at address 0xE0032800 + 0x4 and so on. The definition of each of these is the same.

SHDL_EHCI	Scheduler Memory Register, EHCI Format						0xE0032800
b95	b94	b93	b92	b91	b90	b89	b88
							IOC_RATE[7]
							R/W
							R
							0

SHDL_EHCI	Scheduler Memory Register, EHCI Format						
b87	b86	b85	b84	b83	b82	b81	b80
IOC_RATE[6:0]							TRNS_MODE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_EHCI	Scheduler Memory Register, EHCI Format						
b79	b78	b77	b76	b75	b74	b73	b72
TOTAL_BYTE_COUNT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_EHCI	Scheduler Memory Register, EHCI Format						
b71	b70	b69	b68	b67	b66	b65	b64
TOTAL_BYTE_COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_EHCI	Scheduler Memory Register, EHCI Format						
b63	b62	b61	b60	b59	b58	b57	b56
EP0_CODE[1:0]		BYPASS_ERROR	MMULT[1:0]		RESP_RATE[7:5]		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_EHCI	Scheduler Memory Register, EHCI Format						
b55	b54	b53	b52	b51	b50	b49	b48
RESP_RATE[4:0]					POLLING_RATE[7:5]		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_EHCI	Scheduler Memory Register, EHCI Format						
b47	b46	b45	b44	b43	b42	b41	b40
POLLING_RATE[4:0]					MAX_PKT_SIZE[10:8]		
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.14.35 SHDL_OHCI *(continued)*

SHDL_EHCI	Scheduler Memory Register, EHCI Format						
b39	b38	b37	b36	b35	b34	b33	b32
MAX_PKT_SIZE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_EHCI	Scheduler Memory Register, EHCI Format						
b31	b30	b29	b28	b27	b26	b25	b24
UFRAME_SMASK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_EHCI	Scheduler Memory Register, EHCI Format						
b23	b22	b21	b20	b19	b18	b17	b16
PING	RL[3:0]				MULT[1:0]		ISO_EPM
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_EHCI	Scheduler Memory Register, EHCI Format						
b15	b14	b13	b12	b11	b10	b9	b8
CERR[1:0]		NAK_CNT[3:0]				HALT	T
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SHDL_EHCI	Scheduler Memory Register, EHCI Format						
b7	b6	b5	b4	b3	b2	b1	b0
ZPLEN	EPT[1:0]		EPND[4:0]				
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

This area exposes the scheduler memory in EHCI format. Please note that the EHCI and OHCI scheduler memories are using the same physical memory, so overwriting one will also overwrite the other. Scheduler memory consists of three sections:

1. Lower Section:
2. Upper Section:
3. Scratch Section:

This register exposes only the Lower and Upper section. The Scratch section is only used by Hardware and it is not accessible by Software. The Lower and Upper sections each consist of 32 End Points and each End Point requires 3 scheduler memory locations. In result, the MMIO address for lower section would be x800-x97C and for upper portion would be x980-xAFC. Software can update the appropriate portion of the memory based on the PERI_SHDL_STATUS/ ASYNC_SHDL_STATUS bits in the UIB_HOST_SHDL_CS register.

continued on next page

10.14.35 SHDL_OHCI *(continued)*

Bit	Name	Description
127:96	**SHDL_NOT_USED[31:0]**	Not used and not mapped to RAM, only present for alignment reasons
88:81	**IOC_RATE[7:0]**	After IOC_rate*resp_rate the scheduler will issue interrupt at the next interrupt threshold (UIB_EHCI_USBCMD.INT_THRESHOLD_CTRL)
80	**TRNS_MODE**	0 Packet mode 1 Stream mode
79:64	**TOTAL_BYTE_COUNT[15:0]**	Total number of byte count for the transaction.
63:62	**EP0_CODE[1:0]**	This field represents the EP0 type of transaction: 00 Reserved 01 Setup + Data Phase(OUT) + Status Phase(IN) 10 Setup + Data Phase(IN) + Status Phase(OUT) 11 Setup + Status Phase(IN)
61	**BYPASS_ERROR**	This bit will disable any error that would cause an EP to be de-activated. This bit should be used only for Isochronous Endpoint.
60:59	**MMULT[1:0]**	Should be programmed to zero for EHCI. This field is used to expand the MULT field to support the OHCI ControlBulkServiceRatio bandwidth allocation.
58:51	**RESP_RATE[7:0]**	After how many packet the response should be written into SRAM. The packet counter increments based on the condition specified in the UIB_HOST_RESP_CS.WR_RESP_COND.
50:43	**POLLING_RATE[7:0]**	00h Reserved 01h 1 ms (Every frame) 02h 2 ms (Every 2 frames) 04h 4 ms (Every 4 frames) 08h 8 ms (Every 8 frames) 10h 16 ms (Every 16 frames) 20h 32 ms (Every 32 frames) This will be on top of the Interrupt Schedule Mask in EHCI case.
42:32	**MAX_PKT_SIZE[10:0]**	This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). The maximum value this field may contain is 0x400 (1024).
31:24	**UFRAME_SMASK[7:0]**	The host controller uses the value of the three low-order bits of the FRINDEX register as an index into a bit position in this bit vector. If the µFrame S-mask field has a one at the indexed bit position then this queue head is a candidate for transaction execution.
23	**PING**	Should be programmed to zero for OHCI. This field enables the host to issue Ping token when Direction is OUT and it is high-speed. 0 Do not issue ping token for high-speed OUT 1 Do issue ping token for high-speed OUT
22:19	**RL[3:0]**	Nak Count Reload (RL). This field contains a value, which is used by the host controller to reload the Nak Counter field.

continued on next page

10.14.35 SHDL_OHCI *(continued)*

18:17	**MULT[1:0]**	High-Bandwidth Pipe Multiplier. This field is a multiplier used to key the host controller as the number of successive packets the host controller may submit to the endpoint in the current execution. This field should be programmed according to the EPM configuration for that EP. This field will get decremented only if the transfer is successful. If not successful, the original value will be reloaded. 00 Reserved. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per micro-frame 10 Two transactions to be issued for this endpoint per micro-frame 11 Three transactions to be issued for this endpoint per micro-frame
16	**ISO_EPM**	0 For ISO-OUT when EPM is empty for that EP then a ZLP will be issued by TP. For ISO-IN when EPM is full for that EP then and ISO-IN will be issued. 1 For ISO-OUT when EPM is empty for that EP then that entry will be skipped. For ISO-IN when EPM is full for that EP then that entry will be skipped.
15:14	**CERR[1:0]**	This field is a 2-bit down counter that keeps track of the number of consecutive Errors detected while executing this qTD. If this field is programmed with a nonzero value during setup, the Host Controller decrements the count and writes it back to the qTD if the transaction fails. If the counter counts from one to zero, the Host Controller marks the qTD inactive, sets the Halted bit to a one and error status bit for the error that caused CERR to decrement to zero. An interrupt will be generated if the USB Error Interrupt Enable bit in the USBINTR register is set to a one. If HCD programs this field to zero during setup, the Host Controller will not count errors for this qTD and there will be no limit on the retries of this qTD.
13:10	**NAK_CNT[1:0]**	This field is a counter the host controller decrements whenever a transaction for the endpoint associated with this queue head results in a Nak or Nyet response.
9	**HALT**	This will indicate that the EP is halted. Skip this qTD and move to next qTD.
8	**T**	0 Not End of the Period/Asynchronous list 1 End of the Period/Asynchronous list This bit indicates that there are no more valid entries in the current list.
7	**ZPLEN**	This field is used only for non-ISO endpoints. 0 Scheduler will set the active bit to zero when the total byte count reaches zero. 1 Scheduler will set the active bit to zero when the total byte count is zero and a ZLP is sent/received.
6:5	**EPT[1:0]**	The End Point Type. 00 Control 01 Isochronous 10 Bulk 11 Interrupt
4:0	**EPND[4:0]**	[3:0]: EP number, Values: 0-15 [4]: EP Direction, 1: OUT, 0: IN The direction bit is not used for EP0. Scheduler uses the EP0_code to determine the direction of the EP0 transaction.

10.15 USB3 Link Controller Registers

10.15.1 LNK_CONF

Link Configuration Register

LNK_CONF	Link Configuration Register						0xE0033000
b31	b30	b29	b28	b27	b26	b25	b24

LNK_CONF	Link Configuration Register						
b23	b22	b21	b20	b19	b18	b17	b16

LNK_CONF	Link Configuration Register						
b15	b14	b13	b12	b11	b10	b9	b8
EPM_FIRST_DELAY[3:0]					CREDIT_ADV_ HOLDOFF	LDN_DETECTION	FORCE_POWER_ PRESENT
R/W	R/W	R/W	R/W		R/W	R/W	R/W
R	R	R	R		R	R	R
5					0	0	0

LNK_CONF	Link Configuration Register						
b7	b6	b5	b4	b3	b2	b1	b0
	LCW_IGNORE_ RSVD						
	R/W						
	R						
	1						

Configuration Settings for Link Layer Block. (for debug and compatibility testing)

Bit	Name	Description
15:12	EPM_FIRST_DELAY[3:0]	Delay sending of first Header in a egress burst in PCLK cycles (125 MHz). This is to give the DMA network time to warm up the data pipeline.
10	CREDIT_ADV_HOLDOFF	Hold-off Credit Advertisement until Sequence Number Advertisement Received
9	LDN_DETECTION	Enable host LDN detection (see USB ECN#001)
8	FORCE_POWER_PRESENT	Force PowerPresent from PHY On
6	LCW_IGNORE_RSVD	0 Check reserved bits in Link Control Word are 0 (LCW) 1 Ignore reserved bits in Link Control Word (LCW)

10.15.2 LNK_INTR

Link Interrupt Register

LNK_INTR	Link Interrupt Register						0xE0033004
b31	b30	b29	b28	b27	b26	b25	b24

LNK_INTR	Link Interrupt Register						
b23	b22	b21	b20	b19	b18	b17	b16
						LTSSM_RESET	**LTSSM_ DISCONNECT**
						R/W1C	R/W1C
						R/W1S	R/W1S
						0	0

LNK_INTR	Link Interrupt Register						
b15	b14	b13	b12	b11	b10	b9	b8
LTSSM_ CONNECT	**U2_INACTIVITY_ TIMEOUT**	**PHY_ERROR**	**LINK_ERROR**	**BAD_LCW**	**LPMA**	**LXU**	**LAU**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

LNK_INTR	Link Interrupt Register						
b7	b6	b5	b4	b3	b2	b1	b0
LGO_U3	**LGO_U2**	**LGO_U1**	**LCRD**	**LBAD**	**LRTY**	**LGOOD**	**LTSSM_STATE_ CHG**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

[USB 3.0, §7.2.2.2, p 7−11…7−14]

Bit	Name	Description
17	**LTSSM_RESET**	LTSSM Reset Received (Hot or Warm)
16	**LTSSM_DISCONNECT**	LTSSM Transitions to SS.Disabled
15	**LTSSM_CONNECT**	LTSSM Transition to Polling — indicating successful SuperSpeed far-end receiver termination detection
14	**U2_INACTIVITY_TIMEOUT**	U2 Inactivity Timeout Interrupt
13	**PHY_ERROR**	PHY Error Count Threshold Reached
12	**LINK_ERROR**	Link Error Count Threshold Reached
11	**BAD_LCW**	Illegal LCW received (see LNK_CONTROL_WORD for details)

continued on next page

10.15.2 LNK_INTR *(continued)*

10	**LPMA**	LPMA Received Interrupt
9	**LXU**	LXU Received Interrupt
8	**LAU**	LAU Received Interrupt
7	**LGO_U3**	LGO_U3 Received Interrupt
6	**LGO_U2**	LGO_U2 Received Interrupt
5	**LGO_U1**	LGO_U1 Received Interrupt
4	**LCRD**	LCRD Received Interrupt
3	**LBAD**	LBAD Received Interrupt
2	**LRTY**	LRTY Received Interrupt
1	**LGOOD**	LGOOD Received Interrupt
0	**LTSSM_STATE_CHG**	LTSSM State Change Interrupt

10.15.3 LNK_INTR_MASK

Link Interrupt Mask Register

LNK_INTR_MASK			Link Interrupt Mask Register				0xE0033008
b31	b30	b29	b28	b27	b26	b25	b24

LNK_INTR_MASK			Link Interrupt Mask Register				
b23	b22	b21	b20	b19	b18	b17	b16
						LTSSM_RESET	**LTSSM_ DISCONNECT**
						R/W	R/W
						R	R
						0	0

LNK_INTR_MASK			Link Interrupt Mask Register				
b15	b14	b13	b12	b11	b10	b9	b8
LTSSM_ CONNECT	**U2_INACTIVITY_ TIMEOUT**	**PHY_ERROR**	**LINK_ERROR**	**BAD_LCW**	**LPMA**	**LXU**	**LAU**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

LNK_INTR_MASK			Link Interrupt Mask Register				
b7	b6	b5	b4	b3	b2	b1	b0
LGO_U3	**LGO_U2**	**LGO_U1**	**LCRD**	**LBAD**	**LRTY**	**LGOOD**	**LTSSM_STATE_ CHG**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Mask bits for each of the interrupt causes in the LNK_INTR register. Setting one of these bits to ‘1’, enables reporting of the corresponding interrupt to the higher level.

Bit	Name	Description
17	**LTSSM_RESET**	LTSSM Reset Received (Hot or Warm)
16	**LTSSM_DISCONNECT**	LTSSM Transitions to SS.Disabled
15	**LTSSM_CONNECT**	LTSSM Transition to Polling — indicating successful SuperSpeed far-end receiver termination detection
14	**U2_INACTIVITY_TIMEOUT**	U2 Inactivity Timeout Interrupt
13	**PHY_ERROR**	PHY Error Count Threshold Reached
12	**LINK_ERROR**	Link Error Count Threshold Reached
11	**BAD_LCW**	Illegal LCW received (see LNK_CONTROL_WORD for details)

continued on next page

10.15.3 LNK_INTR *(continued)*

10	**LPMA**	LPMA Received Interrupt
9	**LXU**	LXU Received Interrupt
8	**LAU**	LAU Received Interrupt
7	**LGO_U3**	LGO_U3 Received Interrupt
6	**LGO_U2**	LGO_U2 Received Interrupt
5	**LGO_U1**	LGO_U1 Received Interrupt
4	**LCRD**	LCRD Received Interrupt
3	**LBAD**	LBAD Received Interrupt
2	**LRTY**	LRTY Received Interrupt
1	**LGOOD**	LGOOD Received Interrupt
0	**LTSSM_STATE_CHG**	LTSSM State Change Interrupt

10.15.4 LNK_ERROR_CONF

Link Error Counter Configuration Register

LNK_ERROR_CONF	Link Error Counter Configuration Register						0xE003300C
b31	b30	b29	b28	b27	b26	b25	b24

LNK_ERROR_CONF	Link Error Counter Configuration Register						
b23	b22	b21	b20	b19	b18	b17	b16

LNK_ERROR_CONF	Link Error Counter Configuration Register						
b15	b14	b13	b12	b11	b10	b9	b8
	CREDIT_ADV_LGO_EN	CREDIT_ADV_HP_EN	CREDIT_ADV_TIMEOUT_EN	HDR_ADV_LGO_EN	HDR_ADV_LCRD_EN	HDR_ADV_HP_EN	HDR_ADV_TIMEOUT_EN
	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	R	R	R	R	R	R	R
	1	1	1	1	1	1	1

LNK_ERROR_CONF	Link Error Counter Configuration Register						
b7	b6	b5	b4	b3	b2	b1	b0
TX_SEQ_NUM_ERR_EN	PM_LC_TIMEOUT_EN	CREDIT_HP_TIMEOUT_EN	MISSING_LCRD_EN	MISSING_LGOOD_EN	RX_HP_FAIL_EN	RX_SEQ_NUM_ERR_EN	HP_TIMEOUT_EN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	1	1	1	1	1	1	1

Link Error Count [USB 3.0: §7.3, p 7−26…7−33, esp Table 7−11, p 7−32]

The Link Error Count keeps track of the number of errors for which the Link Layer Block had to transition to the Recovery State before resuming normal operation. Each error class can be enabled (default on) for debugging purposes.

Bit	Name	Description
14	CREDIT_ADV_LGO_EN	Rx Header Buffer Credit Advertisement LGO_Ux Received Error Count Enable LGO_Ux Link Command received during Rx Header Buffer Credit Advertisement. [USB 3.0: §7.3.7, p 7−30…7−31]
13	CREDIT_ADV_HP_EN	Rx Header Buffer Credit Advertisement HP Received Error Count Enable Header Packet received during Rx Header Buffer Credit Advertisement. [USB 3.0: §7.3.7, p 7−30…7−31]
12	CREDIT_ADV_TIMEOUT_EN	Rx Header Buffer Credit Advertisement CREDIT_HP_TIMER Timeout Count Enable CREDIT_HP_TIMER timeout before receipt of LCRD_x Link Command during Rx Header Buffer Credit Advertisement [USB 3.0: §7.3.7, p 7−30…7−31]

continued on next page

10.15.4 LNK_ERROR_CONF *(continued)*

11	**HDR_ADV_LGO_EN**	Header Sequence Number Advertisement LGO_Ux Received Error Count Enable LGO_Ux Link Command received during Header Sequence Number Advertisement [USB 3.0: §7.3.6, p 7−30]
10	**HDR_ADV_LCRD_EN**	Header Sequence Number Advertisement LCRD_x Received Error Count Enable LCRD_x Link Command received during Header Sequence Number Advertisement [USB 3.0: §7.3.6, p 7−30]
9	**HDR_ADV_HP_EN**	Header Sequence Number Advertisement HP Received Error Count Enable Header Packet received during Header Sequence Number Advertisement [USB 3.0: §7.3.6, p 7−30]
8	**HDR_ADV_TIMEOUT_EN**	Header Sequence Number Advertisement PENDING_HP_TIMER Timeout Count Enable PENDING_HP_TIMER timeout before receipt of Header Sequence Number LGOOD_n Link Command [USB 3.0: §7.3.6, p 7−30]
7	**TX_SEQ_NUM_ERR_EN**	ACK Tx Header Sequence Number Error Count Enable Received LGOOD_n does not match ACK Tx Header Sequence Number. [USB 3.0: §7.3.5, p 7−30]
6	**PM_LC_TIMEOUT_EN**	PM_LC_TIMER Timeout Count Enable This indicates that an LGO_Ux, LAU, or LXU Link Command is missed. [USB 3.0: §7.3.4, p 7−29]
5	**CREDIT_HP_TIMEOUT_EN**	CREDIT_HP_TIMER Timeout Count Enable Remote Rx Header Buffer Credit has not been received by CREDIT_HP_TIMEOUT. [USB 3.0: §7.2.4.1.10, p 7−21…7−22]
4	**MISSING_LCRD_EN**	Missing LCRD_x Detection Count Enable LCRD_x Sequence does not match what is expected. [USB 3.0: §7.3.4, p 7−29]
3	**MISSING_LGOOD_EN**	Missing LGOOD_n Detection Count Enable LGOOD_n Sequence Number does not match what is expected. [USB 3.0: §7.3.4, p 7−29]
2	**RX_HP_FAIL_EN**	Receive Header Packet Fail Count Enable Link Layer Block has failed to receive a Header Packet for three consecutive times. Failures are CRC errors or spurious K-symbols. [USB 3.0: §7.3.3.2, p 7−28]
1	**RX_SEQ_NUM_ERR_EN**	Rx Header Sequence Number Error Count Enable Received Rx Header Sequence Number does not match what is expected. [USB 3.0: §7.3.3.3, p 7−28]
0	**HP_TIMEOUT_EN**	PENDING_HP_TIMER Timeout Count Enable Header Packet acknowledgement has not been received by PENDING_HP_TIMEOUT. [USB 3.0: §7.2.4.1.10, p 7−21]

10.15.5 LNK_ERROR_STATUS

Link Error Status Register

LNK_ERROR_STATUS			Link Error Status Register				0xE0033010
b31	b30	b29	b28	b27	b26	b25	b24

LNK_ERROR_STATUS			Link Error Status Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_ERROR_STATUS			Link Error Status Register				
b15	b14	b13	b12	b11	b10	b9	b8
	CREDIT_ADV_LGO_EV	CREDIT_ADV_HP_EV	CREDIT_ADV_TIMEOUT_EV	HDR_ADV_LGO_EV	HDR_ADV_LCRD_EV	HDR_ADV_HP_EV	HDR_ADV_TIMEOUT_EV
	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
	0	0	0	0	0	0	0

LNK_ERROR_STATUS			Link Error Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
TX_SEQ_NUM_ERR_EV	PM_LC_TIMEOUT_EV	CREDIT_HP_TIMEOUT_EV	MISSING_LCRD_EV	MISSING_LGOOD_EV	RX_HP_FAIL_EV	RX_SEQ_NUM_ERR_EV	HP_TIMEOUT_EV
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

The LNK_ERROR_STATUS register indicates whether any of the USB 3.0 link error conditions is detected by the FX3 device since the corresponding status was previously cleared.

Bit	Name	Description
14	CREDIT_ADV_LGO_EV	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
13	CREDIT_ADV_HP_EV	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
12	CREDIT_ADV_TIMEOUT_EV	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
11	HDR_ADV_LGO_EV	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
10	HDR_ADV_LCRD_EV	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.

continued on next page

10.15.5 LNK_ERROR_STATUS *(continued)*

9	**HDR_ADV_HP_EV**	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
8	**HDR_ADV_TIMEOUT_EV**	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
7	**TX_SEQ_NUM_ERR_EV**	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
6	**PM_LC_TIMEOUT_EV**	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
5	**CREDIT_HP_TIMEOUT_EV**	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
4	**MISSING_LCRD_EV**	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
3	**MISSING_LGOOD_EV**	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
2	**RX_HP_FAIL_EV**	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
1	**RX_SEQ_NUM_ERR_EV**	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
0	**HP_TIMEOUT_EV**	Indicates this error (see LNK_ERROR_CONF for description) occurred since this bit was last cleared by firmware.

10.15.6 LNK_ERROR_COUNT

Error Counter Register

LNK_ERROR_COUNT			Error Counter Register				0xE0033014
b31	b30	b29	b28	b27	b26	b25	b24
PHY_ERROR_COUNT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
LNK_ERROR_COUNT			**Error Counter Register**				
b23	b22	b21	b20	b19	b18	b17	b16
PHY_ERROR_COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
LNK_ERROR_COUNT			**Error Counter Register**				
b15	b14	b13	b12	b11	b10	b9	b8
LINK_ERROR_COUNT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
LNK_ERROR_COUNT			**Error Counter Register**				
b7	b6	b5	b4	b3	b2	b1	b0
LINK_ERROR_COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Error counter register keeps track of errors from PHY and Link Controller. Only errors enabled in LNK_PHY_ERROR_CONF and LNK_LINK_ERROR_CONF are counted.

Bit	Name	Description
31:16	**PHY_ERROR_COUNT[15:0]**	Count of receive errors from the USB 3.0 PHY. This is for debug purposes.
15:0	**LINK_ERROR_COUNT[15:0]**	The Link Error Count keeps track of the number of errors for which the Link Layer Block had to transition to the Recovery State before resuming normal operation. Counting of errors in each class can be enabled through the LINK_ERROR_CONF register.

10.15.7 LNK_ERROR_COUNT_THRESHOLD

Error Count Threshold Register

LNK_ERROR_COUNT_THRESHOLD			Error Count Threshold Register				0xE0033018
b31	b30	b29	b28	b27	b26	b25	b24
PHY_ERROR_THRESHOLD[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
LNK_ERROR_COUNT_THRESHOLD			**Error Count Threshold Register**				
b23	b22	b21	b20	b19	b18	b17	b16
PHY_ERROR_THRESHOLD[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
LNK_ERROR_COUNT_THRESHOLD			**Error Count Threshold Register**				
b15	b14	b13	b12	b11	b10	b9	b8
LINK_ERROR_THRESHOLD[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
LNK_ERROR_COUNT_THRESHOLD			**Error Count Threshold Register**				
b7	b6	b5	b4	b3	b2	b1	b0
LINK_ERROR_THRESHOLD[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Threshold values for asserting Error Count Interrupts. These values are the error count limits after which the respective interrupts are generated.

Bit	Name	Description
31:16	**PHY_ERROR_THRESHOLD[15:0]**	PHY Error Count Threshold for Interrupt Generation
15:0	**LINK_ERROR_THRESHOLD[15:0]**	Link Error Count Threshold for Interrupt Generation

10.15.8 LNK_PHY_CONF

USB 3.0 PHY Configuration Register

LNK_PHY_CONF			USB 3.0 PHY Configuration Register				0xE003301C
b31	b30	b29	b28	b27	b26	b25	b24
RX_TERMINATION_ENABLE	RX_TERMINATION_OVR_VAL	RX_TERMINATION_OVR					
R/W	R/W	R/w					
R	R	R					
0	0	0					

LNK_PHY_CONF			USB 3.0 PHY Configuration Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_PHY_CONF			USB 3.0 PHY Configuration Register				
b15	b14	b13	b12	b11	b10	b9	b8

LNK_PHY_CONF			USB 3.0 PHY Configuration Register				
b7	b6	b5	b4	b3	b2	b1	b0
						PHY_MODE[1:0]	
						R	R
						R	R
						1	

[PIPE 3.0]

Bit	Name	Description
31	**RX_TERMINATION_ENABLE**	PHY Receiver Termination Enable
30	**RX_TERMINATION_OVR_VAL**	PHY Receiver Termination Override Value 0 Removed 1 Present
29	**RX_TERMINATION_OVR**	PHY Receiver Termination Override
1:0	**PHY_MODE[1:0]**	PHY Operation Mode 01 USB Super Speed

10.15.9 LNK_PHY_MPLL_STATUS

USB 3.0 PHY MPLL Status Register

LNK_PHY_MPLL_STATUS			USB 3.0 PHY MPLL Status Register				0xE003302C
b31	b30	b29	b28	b27	b26	b25	b24

LNK_PHY_MPLL_STATUS			USB 3.0 PHY MPLL Status Register				
b23	b22	b21	b20	b19	b18	b17	b16
REF_CLKREQ_N	**REF_CLKDIV2**	**REF_SSP_EN**	**SSC_REF_CLK_SEL[7:3]**				
R	R	R/W	R	R	R	R	R
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
1	0	0					

LNK_PHY_MPLL_STATUS			USB 3.0 PHY MPLL Status Register				
b15	b14	b13	b12	b11	b10	b9	b8
SSC_REF_CLK_SEL[2:0]			**SSC_RANGE[1:0]**		**SSC_EN**	**MPLL_MULTIPLIER[6:5]**	
R	R	R	R/W	R/W	R/W	R	R
R/W	R/W	R/W	R	R	R	R/W	R/W
0x88			0	0	1	H	H

LNK_PHY_MPLL_STATUS			USB 3.0 PHY MPLL Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
MPLL_MULTIPLIER[4:0]							
R	R	R	R	R			
R/W	R/W	R/W	R/W	R/W			
H	H	H	H	H			

Bit	Name	Description
23	**REF_CLKREQ_N**	PHY output signal ref_clkreq_n
22	**REF_CLKDIV2**	USB 3.0 PHY Input Reference Clock Divider Control
21	**REF_SSP_EN**	USB 3.0 PHY Reference Clock Enable for SSP PHY Enables the reference clock to the PHY prescaler. This signal must remain deasserted until the reference clock is stable.
20:13	**SSC_REF_CLK_SEL[7:0]**	Spread Spectrum Reference Clock Shifting
12:11	**SSC_RANGE[1:0]**	Spread Spectrum Clock Range
10	**SSC_EN**	Spread Spectrum Enable
9:3	**MPLL_MULTIPLIER[6:0]**	MPLL Frequency Multiplier Control Default values (based on clock crystal frequency): 19.2 MHz ⇒ 0x02 26 MHz ⇒ 0x60 38.4 MHz ⇒ 0x41 52 MHz ⇒ 0x30

10.15.10 LNK_PHY_TX_TRIM

USB 3.0 PHY Transmitter Config Register

LNK_PHY_TX_TRIM			USB 3.0 PHY Transmitter, Config Register				0xE003303C
b31	b30	b29	b28	b27	b26	b25	b24
				PCS_TX_SWING_LOW[6:3]			
				R/w	R/w	R/w	R/w
				R	R	R	R

LNK_PHY_TX_TRIM			USB 3.0 PHY Transmitter, Config Register				
b23	b22	b21	b20	b19	b18	b17	b16
PCS_TX_SWING_LOW[2:0]			PCS_TX_SWING_FULL[6:2]				
R/w	R/w	R/w	R/w	R/w	R/w	R/w	R/w
R	R	R	R	R	R	R	R
105							

LNK_PHY_TX_TRIM			USB 3.0 PHY Transmitter, Config Register				
b15	b14	b13	b12	b11	b10	b9	b8
PCS_TX_SWING_FULL[1:0]			PCS_TX_DEEMPH_6DB[5:1]				
R/w	R/w		R/W	R/W	R/W	R/W	R/W
R	R		R	R	R	R	R
105							

LNK_PHY_TX_TRIM			USB 3.0 PHY Transmitter, Config Register				
b7	b6	b5	b4	b3	b2	b1	b0
PCS_TX_DEEMPH_6DB[0]		PCS_TX_DEEMPH_3P5DB[5:0]					
R/W		R/W	R/W	R/W	R/W	R/W	R/W
R		R	R	R	R	R	R
32		21					

See Synopsys USB3 SSP PHY Databook

Bit	Name	Description
27:21	**PCS_TX_SWING_LOW[6:0]**	TX Amplitude (Low Swing Mode) in 10 mv units
20:14	**PCS_TX_SWING_FULL[6:0]**	TX Amplitude (Full Swing Mode) in 10 mv units
12:7	**PCS_TX_DEEMPH_6DB[5:0]**	TX de-emphasis at 6dB
5:0	**PCS_TX_DEEMPH_3P5DB[5:0]**	TX de-emphasis at 3.5dB

10.15.11 LNK_PHY_ERROR_CONF

PHY Error Counter Configuration Register

LNK_PHY_ERROR_CONF	PHY Error Counter Configuration Register						0xE0033040
b31	b30	b29	b28	b27	b26	b25	b24

LNK_PHY_ERROR_CONF	PHY Error Counter Configuration Register						
b23	b22	b21	b20	b19	b18	b17	b16

LNK_PHY_ERROR_CONF	PHY Error Counter Configuration Register						
b15	b14	b13	b12	b11	b10	b9	b8
							PHY_LOCK_EN
							R/W
							R
							1

LNK_PHY_ERROR_CONF	PHY Error Counter Configuration Register						
b7	b6	b5	b4	b3	b2	b1	b0
TRAINING_ ERROR_EN	RX_ERROR_ CRC32_EN	RX_ERROR_ CRC16_EN	RX_ERROR_ CRC5_EN	PHY_ERROR_DIS PARITY_EN	PHY_ERROR_EB _UND_EN	PHY_ERROR_EB _OVR_EN	PHY_ERROR_DE CODE_EN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	1	1	1	1	1	1	1

From RxStatus[2:0] [PIPE, p 22]

Configuration of receive error counter for the USB 3.0 PHY errors. This is for debug purposes.

Bit	Name	Description
8	PHY_LOCK_EN	Enable Counting of PHY Lock Loss. Lock Indicator To Be Determined
7	TRAINING_ERROR_EN	Enable Counting of Training Sequence Error
6	RX_ERROR_CRC32_EN	Enable Counting of Receive CRC-32 Error
5	RX_ERROR_CRC16_EN	Enable Counting of Receive CRC-16 Error
4	RX_ERROR_CRC5_EN	Enable Counting of Receive CRC-5 Error
3	PHY_ERROR_DISPARITY_EN	Enable Counting of Receive Disparity Error (RxStatus == 3'b111)
2	PHY_ERROR_EB_UND_EN	Enable Counting of Elastic Buffer Underflow (RxStatus == 3'b110)
1	PHY_ERROR_EB_OVR_EN	Enable Counting of Elastic Buffer Overflow (RxStatus == 3'b101)
0	PHY_ERROR_DECODE_EN	Enable Counting of 8b/10b Decode Errors (RxStatus == 3'b100)

10.15.12 LNK_PHY_ERROR_STATUS

PHY Error Status Register

LNK_PHY_ERROR_STATUS			PHY Error Status Register				0xE0033044
b31	b30	b29	b28	b27	b26	b25	b24

LNK_PHY_ERROR_STATUS			PHY Error Status Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_PHY_ERROR_STATUS			PHY Error Status Register				
b15	b14	b13	b12	b11	b10	b9	b8
							PHY_LOCK_EV
							R/W1C
							R/W1S
							0

LNK_PHY_ERROR_STATUS			PHY Error Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
TRAINING_ ERROR_EV	RX_ERROR_ CRC32_EV	RX_ERROR_ CRC16_EV	RX_ERROR_ CRC5_EV	PHY_ERROR_DIS PARITY_EV	PHY_ERROR_EB _UND_EV	PHY_ERROR_EB _OVR_EV	PHY_ERROR_DE CODE_EV
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

Indicates the occurrence of each PHY error type since it was last cleared by firmware.

Bit	Name	Description
8	PHY_LOCK_EV	Indicates this error (see LNK_PHY_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
7	TRAINING_ERROR_EV	Indicates this error (see LNK_PHY_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
6	RX_ERROR_CRC32_EV	Indicates this error (see LNK_PHY_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
5	RX_ERROR_CRC16_EV	Indicates this error (see LNK_PHY_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
4	RX_ERROR_CRC5_EV	Indicates this error (see LNK_PHY_ERROR_CONF for description) occurred since this bit was last cleared by firmware.

continued on next page

10.15.12 LNK_PHY_TEST *(continued)*

3	**PHY_ERROR_DISPARITY_EV**	Indicates this error (see LNK_PHY_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
2	**PHY_ERROR_EB_UND_EV**	Indicates this error (see LNK_PHY_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
1	**PHY_ERROR_EB_OVR_EV**	Indicates this error (see LNK_PHY_ERROR_CONF for description) occurred since this bit was last cleared by firmware.
0	**PHY_ERROR_DECODE_EV**	Indicates this error (see LNK_PHY_ERROR_CONF for description) occurred since this bit was last cleared by firmware.

10.15.13 LNK_DEVICE_POWER_CONTROL

USB 3.0 Device Power State Control Register

LNK_DEVICE_POWER_CONTROL			USB 3.0 Device Power State Control Register				0xE0033050
b31	b30	b29	b28	b27	b26	b25	b24
YES_U2	**YES_U1**	**NO_U2**	**NO_U1**	**AUTO_U2**	**AUTO_U1**		
R/W	R/W	R/W	R/W	R/W	R/W		
R	R	R	R	R	R		
0	0	0	0	0	0		

LNK_DEVICE_POWER_CONTROL			USB 3.0 Device Power State Control Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_DEVICE_POWER_CONTROL			USB 3.0 Device Power State Control Register				
b15	b14	b13	b12	b11	b10	b9	b8
						EXIT_LP	**TX_LXU**
						R/W1S	R/W1S
						R/W0C	R/W0C
						0	0

LNK_DEVICE_POWER_CONTROL			USB 3.0 Device Power State Control Register				
b7	b6	b5	b4	b3	b2	b1	b0
TX_LAU	**RX_U3**	**RX_U2**	**RX_U1**		**TX_U3**	**TX_U2**	**TX_U1**
R/W1S	R	R	R		R/W1S	R/W1S	R/W1S
R/W0C	R/W	R/W	R/W		R/W0C	R/W0C	R/W0C
0	0	0	0		0	0	0

This register controls the power state change request LCW protocol

Bit	Name	Description
31	**YES_U2**	When host requests transition to U2, automatically accept (send LAU). The interrupt RX_U2 is still raised for firmware to monitor, take additional power saving actions.
30	**YES_U1**	When host requests transition to U1, automatically accept (send LAU). The interrupt RX_U1 is still raised for firmware to monitor, take additional power saving actions.
29	**NO_U2**	When host requests transition to U2, automatically reject (send LXU). The interrupt RX_U2 is still raised for firmware to monitor, take additional actions. This bit must be cleared by firmware when FORCE_PM_ACCEPT is received from host.
28	**NO_U1**	When host requests transition to U1, automatically reject (send LXU). The interrupt RX_U1 is still raised for firmware to monitor, take additional actions. This bit must be cleared by firmware when FORCE_PM_ACCEPT is received from host.
27	**AUTO_U2**	When host requests transition to U2, automatically accept (send LAU) or rejects (send LXU) depending on pending activity. The interrupt RX_U2 is still raised for firmware to monitor, take additional power saving actions.

continued on next page

10.15.13 LNK_DEVICE_POWER_CONTROL *(continued)*

26	**AUTO_U1**	When host requests transition to U1, automatically accept (send LAU) or rejects (send LXU) depending on pending activity. The interrupt RX_U1 is still raised for firmware to monitor, take additional power saving actions.
9	**EXIT_LP**	Exit Low Power State This bit is cleared by hardware when the Link Layer has exited U1/U2/U3.
8	**TX_LXU**	Transmit LXU (NAK) in response to RX_U1/RX_U2/RX_U3. Do not transition to requested power state (LTSSM). This bit is cleared when the acknowledgement is sent.
7	**TX_LAU**	Transmit LAU (ACK) in response to RX_U1/RX_U2/RX_U3. Transition to requested power state (LTSSM). This bit is cleared when the acknowledgement is sent.
6	**RX_U3**	LGO_U3 Received - Request to go to U3 Power State, clear to NAK (send LXU). This bit is cleared by hardware concurrent with TX_LAU/TX_LXU being cleared. Note that an upstream port is not allowed to reject entry to U3. [USB 3.0: §7.2.4.2.4, p 7−25]
5	**RX_U2**	LGO_U2 Received - Request to go to U2 Power State, clear to NAK (send LXU). This bit is cleared by hardware concurrent with TX_LAU/TX_LXU being cleared.
4	**RX_U1**	LGO_U1 Received - Request to go to U1 Power State. This bit is cleared by hardware concurrent with TX_LAU/TX_LXU being cleared.
2	**TX_U3**	Transmit LGO_U3 - Request to go to U3 Power State (send LGO_U3). This bit is cleared by hardware when the LCW is transmitted. Note that an upstream port is not allowed to initiate entry to U3, so this should not be used for device mode. [USB 3.0: §7.2.4.2.4, p 7−25]
1	**TX_U2**	Transmit LGO_U2 - Request to go to U2 Power State (send LGO_U2). This bit is cleared by hardware when the LCW is transmitted.
0	**TX_U1**	Transmit LGO_U1 - Request to go to U1 Power State (send LGO_U1). This bit is cleared by hardware when the LCW is transmitted.

10.15.14 LNK_LTSSM_STATE

Link Training Status State Machine (LTSSM) State Register

LNK_LTSSM_STATE		Link Training Status State Machine (LTSSM) State Register					0xE0033054
b31	b30	b29	b28	b27	b26	b25	b24

LNK_LTSSM_STATE		Link Training Status State Machine (LTSSM) State Register					
b23	b22	b21	b20	b19	b18	b17	b16

LNK_LTSSM_STATE		Link Training Status State Machine (LTSSM) State Register					
b15	b14	b13	b12	b11	b10	b9	b8

LNK_LTSSM_STATE		Link Training Status State Machine (LTSSM) State Register					
b7	b6	b5	b4	b3	b2	b1	b0
		LTSSM_STATE[5:0]					
		R	R	R	R	R	R
		R/W	R/W	R/W	R/W	R/W	R/W
		0	0	0	0	0	0

USB 3.0 interface link layer state control and status

Bit	Name	Description
5:0	**LTSSM_STATE[5:0]**	LTSSM State. See USBLNK_LTSSM Tab in USB-RegMap.xls for more details.

10.15.15 LNK_LFPS_OBSERVE

LFPS Receiver Observability Register

LNK_LFPS_OBSERVE			LFPS Receiver Observability Register				0xE0033064
b31	b30	b29	b28	b27	b26	b25	b24

LNK_LFPS_OBSERVE			LFPS Receiver Observability Register				
b23	b22	b21	b20	b19	b18	b17	b16
POLLING_LFPS_SENT[3:0]				POLLING_LFPS_RCVD[3:0]			
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

LNK_LFPS_OBSERVE			LFPS Receiver Observability Register				
b15	b14	b13	b12	b11	b10	b9	b8

LNK_LFPS_OBSERVE			LFPS Receiver Observability Register				
b7	b6	b5	b4	b3	b2	b1	b0
	LOOPBACK_DET	U3_EXIT_DET	U2_EXIT_DET	U1_EXIT_DET	RESET_DET	PING_DET	POLLING_DET
	R/W0C	R/W0C	R/W0C	R/W0C	R/W0C	R/W0C	R/W0C
	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
	0	0	0	0	0	0	0

Debug register for observability of detected LFPS sequences.

Bit	Name	Description
23:20	**POLLING_LFPS_SENT[3:0]**	Number of LFPS Polling Bursts Sent since last Polling.LFPS entry
19:16	**POLLING_LFPS_RCVD[3:0]**	Number of LFPS Polling Bursts Received since last Polling.LFPS entry
6	**LOOPBACK_DET**	LFPS Sequence detected since last cleared by CPU
5	**U3_EXIT_DET**	LFPS Sequence detected since last cleared by CPU
4	**U2_EXIT_DET**	LFPS Sequence detected since last cleared by CPU
3	**U1_EXIT_DET**	LFPS Sequence detected since last cleared by CPU
2	**RESET_DET**	LFPS Sequence detected since last cleared by CPU
1	**PING_DET**	LFPS Sequence detected since last cleared by CPU
0	**POLLING_DET**	LFPS Sequence detected since last cleared by CPU

10.15.16 LNK_COMPLIANCE_PATTERN_0

Compliance Pattern CP0 Register

LNK_COMPLIANCE_PATTERN_0			Compliance Pattern CP0 Register				0xE0033138
b31	b30	b29	b28	b27	b26	b25	b24

LNK_COMPLIANCE_PATTERN_0			Compliance Pattern CP0 Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_COMPLIANCE_PATTERN_0			Compliance Pattern CP0 Register				
b15	b14	b13	b12	b11	b10	b9	b8
			TXONESZEROS	**LFPS**	**DEEMPHASIS**	**SCRAMBLED**	**K_D**
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	1	1	0

LNK_COMPLIANCE_PATTERN_0			Compliance Pattern CP0 Register				
b7	b6	b5	b4	b3	b2	b1	b0
CP[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x00							

This register determines the behavior of the FX3 device when sending any of the USB 3.0 compliance patterns. The default value of this register corresponds to compliance pattern 0.

Bit	Name	Description
12	**TXONESZEROS**	Enable TXONESZEROS (PIPE PHY Transmit Signal)
11	**LFPS**	LFPS On/Off
10	**DEEMPHASIS**	De-emphasis On/Off
9	**SCRAMBLED**	Scramble On/Off
8	**K_D**	Symbol Type 0 Data (D) 1 Symbol (K)
7:0	**CP[7:0]**	Compliance Pattern

10.15.17 LNK_COMPLIANCE_PATTERN_1

Compliance Pattern CP1 Register

LNK_COMPLIANCE_PATTERN_1			Compliance Pattern CP1 Register				0xE003313C
b31	b30	b29	b28	b27	b26	b25	b24

LNK_COMPLIANCE_PATTERN_1			Compliance Pattern CP1 Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_COMPLIANCE_PATTERN_1			Compliance Pattern CP1 Register				
b15	b14	b13	b12	b11	b10	b9	b8
			TXONESZEROS	LFPS	DEEMPHASIS	SCRAMBLED	K_D
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	1	0	0

LNK_COMPLIANCE_PATTERN_1			Compliance Pattern CP1 Register				
b7	b6	b5	b4	b3	b2	b1	b0
CP[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x4A							

USB 3.0 PHY configuration values for compliance pattern 1.

Bit	Name	Description
12	**TXONESZEROS**	Enable TXONESZEROS (PIPE PHY Transmit Signal)
11	**LFPS**	LFPS On/Off
10	**DEEMPHASIS**	De-emphasis On/Off
9	**SCRAMBLED**	Scramble On/Off
8	**K_D**	Symbol Type 0 Data (D) 1 Symbol (K)
7:0	**CP[7:0]**	Compliance Pattern

10.15.18 LNK_COMPLIANCE_PATTERN_2

Compliance Pattern CP2 Register

LNK_COMPLIANCE_PATTERN_2			Compliance Pattern CP2 Register				0xE0033140
b31	b30	b29	b28	b27	b26	b25	b24

LNK_COMPLIANCE_PATTERN_2			Compliance Pattern CP2 Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_COMPLIANCE_PATTERN_2			Compliance Pattern CP2 Register				
b15	b14	b13	b12	b11	b10	b9	b8
			TXONESZEROS	LFPS	DEEMPHASIS	SCRAMBLED	K_D
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	1	0	0

LNK_COMPLIANCE_PATTERN_2			Compliance Pattern CP2 Register				
b7	b6	b5	b4	b3	b2	b1	b0
CP[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x78							

USB 3.0 PHY configuration values for compliance pattern 2.

Bit	Name	Description
12	**TXONESZEROS**	Enable TXONESZEROS (PIPE PHY Transmit Signal)
11	**LFPS**	LFPS On/Off
10	**DEEMPHASIS**	De-emphasis On/Off
9	**SCRAMBLED**	Scramble On/Off
8	**K_D**	Symbol Type 0 Data (D) 1 Symbol (K)
7:0	**CP[7:0]**	Compliance Pattern

10.15.19 LNK_COMPLIANCE_PATTERN_3

Compliance Pattern CP3 Register

LNK_COMPLIANCE_PATTERN_3			Compliance Pattern CP3 Register				0xE0033144
b31	b30	b29	b28	b27	b26	b25	b24

LNK_COMPLIANCE_PATTERN_3			Compliance Pattern CP3 Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_COMPLIANCE_PATTERN_3			Compliance Pattern CP3 Register				
b15	b14	b13	b12	b11	b10	b9	b8
			TXONESZEROS	**LFPS**	**DEEMPHASIS**	**SCRAMBLED**	**K_D**
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	1	0	1

LNK_COMPLIANCE_PATTERN_3			Compliance Pattern CP3 Register				
b7	b6	b5	b4	b3	b2	b1	b0
CP[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xBC							

USB 3.0 PHY configuration values for compliance pattern 3.

Bit	Name	Description
12	**TXONESZEROS**	Enable TXONESZEROS (PIPE PHY Transmit Signal)
11	**LFPS**	LFPS On/Off
10	**DEEMPHASIS**	De-emphasis On/Off
9	**SCRAMBLED**	Scramble On/Off
8	**K_D**	Symbol Type 0 Data (D) 1 Symbol (K)
7:0	**CP[7:0]**	Compliance Pattern

10.15.20 LNK_COMPLIANCE_PATTERN_4

Compliance Pattern CP4 Register

LNK_COMPLIANCE_PATTERN_4			Compliance Pattern CP4 Register				0xE0033148
b31	b30	b29	b28	b27	b26	b25	b24

LNK_COMPLIANCE_PATTERN_4			Compliance Pattern CP4 Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_COMPLIANCE_PATTERN_4			Compliance Pattern CP4 Register				
b15	b14	b13	b12	b11	b10	b9	b8
			TXONESZEROS	**LFPS**	**DEEMPHASIS**	**SCRAMBLED**	**K_D**
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	1	1	0	0

LNK_COMPLIANCE_PATTERN_4			Compliance Pattern CP4 Register				
b7	b6	b5	b4	b3	b2	b1	b0
CP[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x00							

USB 3.0 PHY configuration values for compliance pattern 4.

Bit	Name	Description
12	**TXONESZEROS**	Enable TXONESZEROS (PIPE PHY Transmit Signal)
11	**LFPS**	LFPS On/Off
10	**DEEMPHASIS**	De-emphasis On/Off
9	**SCRAMBLED**	Scramble On/Off
8	**K_D**	Symbol Type 0 Data (D) 1 Symbol (K)
7:0	**CP[7:0]**	Compliance Pattern

10.15.21 LNK_COMPLIANCE_PATTERN_5

Compliance Pattern CP5 Register

LNK_COMPLIANCE_PATTERN_5			Compliance Pattern CP5 Register				0xE003314C
b31	b30	b29	b28	b27	b26	b25	b24

LNK_COMPLIANCE_PATTERN_5			Compliance Pattern CP5 Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_COMPLIANCE_PATTERN_5			Compliance Pattern CP5 Register				
b15	b14	b13	b12	b11	b10	b9	b8
			TXONESZEROS	**LFPS**	**DEEMPHASIS**	**SCRAMBLED**	**K_D**
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	1	0	1

LNK_COMPLIANCE_PATTERN_5			Compliance Pattern CP5 Register				
b7	b6	b5	b4	b3	b2	b1	b0
CP[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFC							

USB 3.0 PHY configuration values for compliance pattern 5.

Bit	Name	Description
12	**TXONESZEROS**	Enable TXONESZEROS (PIPE PHY Transmit Signal)
11	**LFPS**	LFPS On/Off
10	**DEEMPHASIS**	De-emphasis On/Off
9	**SCRAMBLED**	Scramble On/Off
8	**K_D**	Symbol Type 0 Data (D) 1 Symbol (K)
7:0	**CP[7:0]**	Compliance Pattern

10.15.22 LNK_COMPLIANCE_PATTERN_6

Compliance Pattern CP6 Register

LNK_COMPLIANCE_PATTERN_6			Compliance Pattern CP6 Register				0xE0033150
b31	b30	b29	b28	b27	b26	b25	b24

LNK_COMPLIANCE_PATTERN_6			Compliance Pattern CP6 Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_COMPLIANCE_PATTERN_6			Compliance Pattern CP6 Register				
b15	b14	b13	b12	b11	b10	b9	b8
			TXONESZEROS	**LFPS**	**DEEMPHASIS**	**SCRAMBLED**	**K_D**
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	0	0	1

LNK_COMPLIANCE_PATTERN_6			Compliance Pattern CP6 Register				
b7	b6	b5	b4	b3	b2	b1	b0
CP[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFC							

USB 3.0 PHY configuration values for compliance pattern 6.

Bit	Name	Description
12	**TXONESZEROS**	Enable TXONESZEROS (PIPE PHY Transmit Signal)
11	**LFPS**	LFPS On/Off
10	**DEEMPHASIS**	De-emphasis On/Off
9	**SCRAMBLED**	Scramble On/Off
8	**K_D**	Symbol Type 0 Data (D) 1 Symbol (K)
7:0	**CP[7:0]**	Compliance Pattern

10.15.23 LNK_COMPLIANCE_PATTERN_7

Compliance Pattern CP7 Register

LNK_COMPLIANCE_PATTERN_7			Compliance Pattern CP7 Register				0xE0033154
b31	b30	b29	b28	b27	b26	b25	b24

LNK_COMPLIANCE_PATTERN_7			Compliance Pattern CP7 Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_COMPLIANCE_PATTERN_7			Compliance Pattern CP7 Register				
b15	b14	b13	b12	b11	b10	b9	b8
			TXONESZEROS	**LFPS**	**DEEMPHASIS**	**SCRAMBLED**	**K_D**
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			1	0	1	0	0

LNK_COMPLIANCE_PATTERN_7			Compliance Pattern CP7 Register				
b7	b6	b5	b4	b3	b2	b1	b0
CP[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x00							

USB 3.0 PHY configuration values for compliance pattern 7.

Bit	Name	Description
12	**TXONESZEROS**	Enable TXONESZEROS (PIPE PHY Transmit Signal)
11	**LFPS**	LFPS On/Off
10	**DEEMPHASIS**	De-emphasis On/Off
9	**SCRAMBLED**	Scramble On/Off
8	**K_D**	Symbol Type 0 Data (D) 1 Symbol (K)
7:0	**CP[7:0]**	Compliance Pattern

10.15.24 LNK_COMPLIANCE_PATTERN_8

Compliance Pattern CP8 Register

LNK_COMPLIANCE_PATTERN_8			Compliance Pattern CP8 Register				0xE0033158
b31	b30	b29	b28	b27	b26	b25	b24

LNK_COMPLIANCE_PATTERN_8			Compliance Pattern CP8 Register				
b23	b22	b21	b20	b19	b18	b17	b16

LNK_COMPLIANCE_PATTERN_8			Compliance Pattern CP8 Register				
b15	b14	b13	b12	b11	b10	b9	b8
			TXONESZEROS	**LFPS**	**DEEMPHASIS**	**SCRAMBLED**	**K_D**
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			1	0	0	0	0

LNK_COMPLIANCE_PATTERN_8			Compliance Pattern CP8 Register				
b7	b6	b5	b4	b3	b2	b1	b0
CP[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x00							

USB 3.0 PHY configuration values for compliance pattern 8.

Bit	Name	Description
12	**TXONESZEROS**	Enable TXONESZEROS (PIPE PHY Transmit Signal)
11	**LFPS**	LFPS On/Off
10	**DEEMPHASIS**	De-emphasis On/Off
9	**SCRAMBLED**	Scramble On/Off
8	**K_D**	Symbol Type 0 Data (D) 1 Symbol (K)
7:0	**CP[7:0]**	Compliance Pattern

10.16 USB3 Protocol Layer Registers

10.16.1 PROT_CS

Protocol Layer Control and Status Register

PROT_CS	Protocol Layer Control and Status Register						0xE0033400
b31	b30	b29	b28	b27	b26	b25	b24
MULT_TIMER[4:0]					DISABLE_IDLE_DET	SEQ_NUM_CONFIG	PROT_HOST_RESET_RESP
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0x18					0	0	0

PROT_CS	Protocol Layer Control and Status Register						
b23	b22	b21	b20	b19	b18	b17	b16
TP_THRESHOLD[5:0]						NRDY_ALL	SETUP_CLR_BUSY
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W1C
R	R	R	R	R	R	R	R/W1S
58						0	0

PROT_CS	Protocol Layer Control and Status Register						
b15	b14	b13	b12	b11	b10	b9	b8

PROT_CS	Protocol Layer Control and Status Register						
b7	b6	b5	b4	b3	b2	b1	b0
	DEVICEADDR[6:0]						
	R	R	R	R	R	R	R
	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	0	0	0	0	0	0	0

NRDYALL can be used to temporarily NRDY all transactions on the bus while modifying the EP configuration registers.

SETUP_CLR_BUSY Is used by firmware to process the setup token/data.

Bit	Name	Description
31:27	MULT_TIMER[4:0]	This timer indicates how long protocol should wait for data (MULT is enabled) to be available by EPM before terminating a burst. Protocol will multiply the value programmed by 4.
26	DISABLE_IDLE_DET	This bit will control the idle detection logic in the protocol. 0 Logic is not disabled. 1 Logic is disabled.
25	SEQ_NUM_CONFIG	This bit indicates if the seq numbers are EP based or Stream ID (Socket) based 0 EP based 1 Stream ID (Socket) based

continued on next page

10.16.1 PROT_CS *(continued)*

24	**PROT_HOST_RESET_RESP**	This bit is used to inform the protocol of what the response to incoming TP/DPH should be after warm/host reset. This register will be used by Protocol until the LINK_INTR.LTSSM_RESET is cleared by CPU. 0 Issue NRDY 1 Ignore TP
23:18	**TP_THRESHOLD[5:0]**	Ingress TP response transmit buffer threshold for almost full flag. When buffer contains TP_THRESHOLD items or more, controller will stop issuing credits to host. This field must be larger than 0. The transmit buffer can hold up to 64 responses.
17	**NRDY_ALL**	Set this bit to '1', the hardware will send NRDY all transfers from the host in all endpoint1-31.
16	**SETUP_CLR_BUSY**	Allow device to ACK SETUP status phase packets
6:0	**DEVICEADDR[6:0]**	During the USB enumeration process, the host sends a device a unique 7-bit address, which the USB core copies into this register. The USB Core will automatically respond only to its assigned address. During the USB RESET, this register will be cleared to zero.

10.16.2 PROT_INTR

Protocol Layer Interrupt Register

PROT_INTR			Protocol Layer Interrupt Register				0xE0033404
b31	b30	b29	b28	b27	b26	b25	b24

PROT_INTR			Protocol Layer Interrupt Register				
b23	b22	b21	b20	b19	b18	b17	b16

PROT_INTR			Protocol Layer Interrupt Register				
b15	b14	b13	b12	b11	b10	b9	b8
SET_ADDR0_EV	**EP0_STALLED_EV**	**LMP_INVALID_PORT_CFG_EV**	**LMP_INVALID_PORT_CAP_EV**	**STATUS_STAGE**	**HOST_ERR_EV**	**SUTOK_EV**	**ITP_EV**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

PROT_INTR			Protocol Layer Interrupt Register				
b7	b6	b5	b4	b3	b2	b1	b0
TIMEOUT_HOST_ACK_EV	**TIMEOUT_PING_EV**	**TIMEOUT_PORT_CFG_EV**	**TIMEOUT_PORT_CAP_EV**	**LMP_PORT_CFG_EV**	**LMP_PORT_CAP_EV**	**LMP_UNKNOWN_EV**	**LMP_RCV_EV**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

Contains the miscellaneous (not endpoint related) interrupts causes from the Protocol Layer.

Bit	Name	Description
15	**SET_ADDR0_EV**	Device sets this interrupt when it receives a set address 0 command.
14	**EP0_STALLED_EV**	Device sets this interrupt based on the following conditions: 1: When Host sends more data than it is suppose to in ingress. 2: When Device sends more data that it suppose to in egress. 3: During STATUS Stage, host sends/asks to/for data from device. Device will come out of stall condition when it receives a valid SETUP(SUTOK_EV interrupt will be generated).
13	**LMP_INVALID_PORT_CFG_EV**	Set whenever a LMP port configuration is received but the Link Speed is not ‘1’.
12	**LMP_INVALID_PORT_CAP_EV**	Set whenever a LMP port capability is received but the Link Speed is not ‘1’ or Num HP buffer is not ‘4’ or bit zero of the Direction is not ‘1’.
11	**STATUS_STAGE**	Set when host completes Status Stage of a Control Transfer

continued on next page

10.16.2 PROT_INTR *(continued)*

10	**HOST_ERR_EV**	Set whenever an ACK TP is received with HE=1 [USB 3.0: section 8.5.1, Table 8-12, p 8-13]
9	**SUTOK_EV**	Set whenever a (valid of invalid) SETUP DPP is received that is not a set_address. The set_address DPP is handled entirely in hardware and does not require any firmware intervention.
8	**ITP_EV**	Set whenever an ITP(SOF) occurs
7	**TIMEOUT_HOST_ACK_EV**	The Host Ack Response Timer expired
6	**TIMEOUT_PING_EV**	The Ping Timer expired
5	**TIMEOUT_PORT_CFG_EV**	The Port ConfiguratIon LMP Timer expired
4	**TIMEOUT_PORT_CAP_EV**	The Port Capabilities LMP Timer expired
3	**LMP_PORT_CFG_EV**	A Port Configuration LMP was received. A response may have been sent automatically depending on settings for PROT_LMP_PORT_CONFIGURATION_TIMER.
2	**LMP_PORT_CAP_EV**	A Port Capabilities LMP was received. A response may have been sent automatically depending on settings for PROT_LMP_PORT_CAPABILITIES_TIMER.
1	**LMP_UNKNOWN_EV**	An unkNown LMP was received and placed in PROT_LMP_PACKET_RX. The LMP was not recognized and no response LMP was sent back.
0	**LMP_RCV_EV**	A LMP was received and placed in PROT_LMP_PACKET_RX. The LMP may have been recognized and processed as well (leading to other interrupts in this register).

10.16.3 PROT_INTR_MASK

Protocol Interrupts Mask Register

PROT_INTR_MASK			Protocol Interrupts Mask Register				0xE0033408
b31	b30	b29	b28	b27	b26	b25	b24

PROT_INTR_MASK			Protocol Interrupts Mask Register				
b23	b22	b21	b20	b19	b18	b17	b16

PROT_INTR_MASK			Protocol Interrupts Mask Register				
b15	b14	b13	b12	b11	b10	b9	b8
SET_ADDR0_EN	**EP0_STALLED_EN**	**LMP_INVALID_PORT_CFG_EN**	**LMP_INVALID_PORT_CAP_EN**	**STATUS_STAGE**	**HOST_ERR_EN**	**SUTOK_EN**	**ITP_EN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

PROT_INTR_MASK			Protocol Interrupts Mask Register				
b7	b6	b5	b4	b3	b2	b1	b0
TIMEOUT_HOST_ACK_EN	**TIMEOUT_PING_EN**	**TIMEOUT_PORT_CFG_EN**	**TIMEOUT_PORT_CAP_EN**	**LMP_PORT_CFG_EN**	**LMP_PORT_CAP_EN**	**LMP_UNKNOWN_EN**	**LMP_RCV_EN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Controls whether protocol layer interrupts are reported to the CPU.

Bit	Name	Description	
15	**SET_ADDR0_EN**	1	Report interrupt to CPU
14	**EP0_STALLED_EN**	1	Report interrupt to CPU
13	**LMP_INVALID_PORT_CFG_EN**	1	Report interrupt to CPU
12	**LMP_INVALID_PORT_CAP_EN**	1	Report interrupt to CPU
11	**STATUS_STAGE**	1	Report interrupt to CPU
10	**HOST_ERR_EN**	1	Report interrupt to CPU
9	**SUTOK_EN**	1	Report interrupt to CPU
8	**ITP_EN**	1	Report interrupt to CPU

continued on next page

10.16.3 PROT_INTR *(continued)*

7	**TIMEOUT_HOST_ACK_EN**	1	Report interrupt to CPU
6	**TIMEOUT_PING_EN**	1	Report interrupt to CPU
5	**TIMEOUT_PORT_CFG_EN**	1	Report interrupt to CPU
4	**TIMEOUT_PORT_CAP_EN**	1	Report interrupt to CPU
3	**LMP_PORT_CFG_EN**	1	Report interrupt to CPU
2	**LMP_PORT_CAP_EN**	1	Report interrupt to CPU
1	**LMP_UNKNOWN_EN**	1	Report interrupt to CPU
0	**LMP_RCV_EN**	1	Report interrupt to CPU

10.16.4 PROT_FRAMECNT

Frame Counter Register

PROT_FRAMECNT			Frame Counter Register				0xE0033428
b31	b30	b29	b28	b27	b26	b25	b24
					DELTA[12:10]		
					R	R	R
					R/W	R/W	R/W
					0	0	0

PROT_FRAMECNT			Frame Counter Register				
b23	b22	b21	b20	b19	b18	b17	b16
DELTA[9:2]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PROT_FRAMECNT			Frame Counter Register				
b15	b14	b13	b12	b11	b10	b9	b8
DELTA[1:0]		SS_MICROFRAME[13:8]					
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PROT_FRAMECNT			Frame Counter Register				
b7	b6	b5	b4	b3	b2	b1	b0
SS_MICROFRAME[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Provides micro-frame number received from host in ITP packets.

Bit	Name	Description
26:14	**DELTA[12:0]**	The delta value in the last ITP received
13:0	**SS_MICROFRAME[13:0]**	MICROFRAME counter which indicates which of the 8 125-microsecond micro-frames last occurred. This is based on ITPs received from Host

10.16.5 PROT_ITP_TIME

ITP Time Free Running Counter Register

PROT_ITP_TIME			ITP Time Free Running Counter				0xE0033430
b31	b30	b29	b28	b27	b26	b25	b24
PROT_ITP_TIME			**ITP Time Free Running Counter**				
b23	b22	b21	b20	b19	b18	b17	b16
COUNTER24[23:16]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
H	H	H	H	H	H	H	H
PROT_ITP_TIME			**ITP Time Free Running Counter**				
b15	b14	b13	b12	b11	b10	b9	b8
COUNTER24[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
H	H	H	H	H	H	H	H
PROT_ITP_TIME			**ITP Time Free Running Counter**				
b7	b6	b5	b4	b3	b2	b1	b0
COUNTER24[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
H	H	H	H	H	H	H	H

This register contains a free running counter running at 125MHz (spread clock) and is used for the PROT_ITP_TIMESTAMP time stamps.

Bit	Name	Description
23:0	**COUNTER24[23:0]**	Current counter value.

10.16.6 PROT_ITP_TIMESTAMP

ITP Time Stamp Register

PROT_ITP_TIMESTAMP			ITP Time Stamp Register				0xE0033434
b31	b30	b29	b28	b27	b26	b25	b24
MICROFRAME_LSB[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PROT_ITP_TIMESTAMP			**ITP Time Stamp Register**				
b23	b22	b21	b20	b19	b18	b17	b16
TIMESTAMP[23:16]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PROT_ITP_TIMESTAMP			**ITP Time Stamp Register**				
b15	b14	b13	b12	b11	b10	b9	b8
TIMESTAMP[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PROT_ITP_TIMESTAMP			**ITP Time Stamp Register**				
b7	b6	b5	b4	b3	b2	b1	b0
TIMESTAMP[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

This register contains a time-stamp of the last ITP received that can be used to calculate BIA messages when needed.

Bit	Name	Description
31:24	**MICROFRAME_LSB[7:0]**	LSBs of MICROFRAME field of ITP when timestamp was taken.
23:0	**TIMESTAMP[23:0]**	Timestamp from a free running counter at 125MHz of the last ITP reception.

10.16.7 PROT_SETUP_DAT

Received SETUP Packet Data Register

PROT_SETUP_DAT				Received SETUP Packet Data Register			0xE0033438
b63	b62	b61	b60	b59	b58	b57	b56
SETUP_LENGTH[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PROT_SETUP_DAT				Received SETUP Packet Data Register			
b55	b54	b53	b52	b51	b50	b49	b48
SETUP_LENGTH[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PROT_SETUP_DAT				Received SETUP Packet Data Register			
b47	b46	b45	b44	b43	b42	b41	b40
SETUP_INDEX[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PROT_SETUP_DAT				Received SETUP Packet Data Register			
b39	b38	b37	b36	b35	b34	b33	b32
SETUP_INDEX[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

PROT_SETUP_DAT				Received SETUP Packet Data Register			
b31	b30	b29	b28	b27	b26	b25	b24
SETUP_VALUE[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PROT_SETUP_DAT				Received SETUP Packet Data Register			
b23	b22	b21	b20	b19	b18	b17	b16
SETUP_VALUE[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PROT_SETUP_DAT				Received SETUP Packet Data Register			
b15	b14	b13	b12	b11	b10	b9	b8
SETUP_REQUEST[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

continued on next page

10.16.7 PROT_SETUP_DAT *(continued)*

PROT_SETUP_DAT			Received SETUP Packet Data Register				
b7	b6	b5	b4	b3	b2	b1	b0
SETUP_REQUEST_TYPE[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

8-byte (2-long word) Storage for USB SETUP data received on EP0

Bit	Name	Description
63:48	**SETUP_LENGTH[15:0]**	Setup data field
47:32	**SETUP_INDEX[15:0]**	Setup data field
31:16	**SETUP_VALUE[15:0]**	Setup data field
15:8	**SETUP_REQUEST[7:0]**	Setup data field
7:0	**SETUP_REQUEST_TYPE[7:0]**	Setup data field

10.16.8 PROT_SEQ_NUM

Sequence Number Register

PROT_SEQ_NUM			Sequence Number				0xE0033440
b31	b30	b29	b28	b27	b26	b25	b24
SEQ_VALID	**COMMAND**						
R/W0C	R/W						
R/W1S	R						
1	0						

PROT_SEQ_NUM			Sequence Number				
b23	b22	b21	b20	b19	b18	b17	b16
			LAST_COMMITTED[4:0]				
			R	R	R	R	R
			R/W	R/W	R/W	R/W	R/W
			0	0	0	0	0

PROT_SEQ_NUM			Sequence Number				
b15	b14	b13	b12	b11	b10	b9	b8
			SEQUENCE_NUMBER[4:0]				
			R/W	R/W	R/W	R/W	R/W
			R/W	R/W	R/W	R/W	R/W
			0	0	0	0	0

PROT_SEQ_NUM			Sequence Number				
b7	b6	b5	b4	b3	b2	b1	b0
			DIR	**ENDPOINT[3:0]**			
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	0	0	0

At power-up and USB Reset all the endpoints data toggle will reset to ‘0’. In general, hardware maintains all the data toggle bits, which are toggled between data packet transfers. The firmware might need to reset or set data toggle bit only following conditions:

- After a configuration changes (i.e., after the host issues a Set Configuration request).
- After an interface’s alternate setting changes (i.e., after the host issues a Set Interface request).
- After the host sends a Clear Feature - Endpoint Stall request to an endpoint.

Warm Reset

Hot Reset

Set Address 0

Disconnect

To read the sequence number field:

1. Software should poll for SEQ_VALID to be 1
2. Software must first write ENDPOINT and DIR field, and then set the COMMAND to 0 and write 0 to SEQ_VALID to initiate the read operation.
3. Then software should poll for SEQ_VALID to go 1 to and that will return the End-point’s sequence numbers.

To write the sequence number field:

1. Software should poll for SEQ_VALID to be 1

2. Software must first write ENDPOINT and DIR field, write the SEQ_NUMBER, set the COMMAND to 1 and write 0 to SEQ_VALID to initiate the write operation.
3. Then software should poll for SEQ_VALID to go 1 to confirm that write has taken place.

Bit	Name	Description
31	**SEQ_VALID**	Set by hardware when read/write operation has completed. Must be cleared by software to initiate a read/write operation.
30	**COMMAND**	0 Read 1 Write
20:16	**LAST_COMMITTED[4:0]**	Sequence number of last packet that was transmitted (can be higher than SEQUENCE NUMBER). Returned as part of a read operation.
12:8	**SEQUENCE_NUMBER[4:0]**	Packet sequence number of next packet to receive/transmit. Set by hardware if COMMAND=0, set by software when COMMAND=1.
4	**DIR**	0 OUT 1 IN
3:0	**ENDPOINT[3:0]**	Endpoint Number

10.16.9 PROT_EP_INTR

Endpoint Interrupt Register

PROT_EP_INTR				Endpoint Interrupts			0xE0033474
b31	b30	b29	b28	b27	b26	b25	b24
EP_OUT[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PROT_EP_INTR				**Endpoint Interrupts**			
b23	b22	b21	b20	b19	b18	b17	b16
EP_OUT[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PROT_EP_INTR				**Endpoint Interrupts**			
b15	b14	b13	b12	b11	b10	b9	b8
EP_IN[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PROT_EP_INTR				**Endpoint Interrupts**			
b7	b6	b5	b4	b3	b2	b1	b0
EP_IN[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

USB Endpoint 0-31 interrupt request register. This is a intermediate register in the UIB interrupt hierarchy. Interrupts are cleared by clearing the respective cause bits in EPI_CS/EPO_CS registers.

Bit	Name	Description
31:16	**EP_OUT[15:0]**	Bit <16+x> indicates an interrupt from EPO_CS[x]
15:0	**EP_IN[15:0]**	Bit <x> indicates an interrupt from EPI_CS[x]

10.16.10 PROT_EP_INTR_MASK

Endpoint Interrupt Mask Register

PROT_EP_INTR_MASK			Endpoint Interrupt Mask				0xE0033478
b31	b30	b29	b28	b27	b26	b25	b24
EP_OUT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PROT_EP_INTR_MASK			**Endpoint Interrupt Mask**				
b23	b22	b21	b20	b19	b18	b17	b16
EP_OUT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PROT_EP_INTR_MASK			**Endpoint Interrupt Mask**				
b15	b14	b13	b12	b11	b10	b9	b8
EP_IN[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
PROT_EP_INTR_MASK			**Endpoint Interrupt Mask**				
b7	b6	b5	b4	b3	b2	b1	b0
EP_IN[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Per endpoint masking of interrupt reporting.

Bit	Name	Description
31:16	**EP_OUT[15:0]**	Bit <16+x> masks any interrupt from EPO_CS[x]
15:0	**EP_IN[15:0]**	Bit <x> masks any interrupt from EPI_CS[x]

10.16.11 PROT_EPI_CS1

SuperSpeed IN Endpoint Control and Status Register

There are 16 PROT_EPI_CS1 registers. The address of each is calculated as PROT_EPI_CS1(x) = 0xE0033500 + (x*0x4). Hence PROT_EPI_CS1(0) is at address 0xE0033500, PROT_EPI_CS1(1) is at address 0xE0033500 + 0x4 and so on. The definition of each of these is the same.

PROT_EPI_CS1		SuperSpeed IN Endpoint Control and Status					0xE0033500
b31	b30	b29	b28	b27	b26	b25	b24
		FIRST_ACK_NUMP_0_MASK	**STREAM_ERROR_MASK**	**DBTERM_MASK**	**HBTERM_MASK**	**OOSERR_MASK**	**SHORT_MASK**
		R/W	R/W	R/W	R/W	R/W	R/W
		R	R	R	R	R	R
		0	0	0	0	0	0

PROT_EPI_CS1		SuperSpeed IN Endpoint Control and Status					
b23	b22	b21	b20	b19	b18	b17	b16
ZERO_MASK	**STREAMNRDY_MASK**	**FLOWCONTROL_MASK**	**RETRY_MASK**	**COMMIT_MASK**	**FIRST_ACK_NUMP_0**	**STREAM_ERROR**	**DBTERM**
R/W	R/W	R/W	R/W	R/W	R/W1C	R/W1C	R/W1C
R	R	R	R	R	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

PROT_EPI_CS1		SuperSpeed IN Endpoint Control and Status					
b15	b14	b13	b12	b11	b10	b9	b8
HBTERM	**OOSERR**	**SHORT**	**ZERO**	**STREAMNRDY**	**FLOWCONTROL**	**RETRY**	**COMMIT**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

PROT_EPI_CS1		SuperSpeed IN Endpoint Control and Status					
b7	b6	b5	b4	b3	b2	b1	b0
		STREAM_ERROR_STALL_EN	**EP_RESET**	**STREAM_EN**	**STALL**	**NRDY**	**VALID**
		R/W	R/W	R/W	R/W	R/W	R/W
		R	R	R	R	R	R
		0	0	0	0	0	0

Endpoint IN Control and Status:

- Setup USB Package buffering, ISO/BULK/INT, IN/OUT and enable/disable endpoint
- Power up default the payload is 64-byte (Max payload count for Full Speed). In High Speed the payload can be up to 512 for BULK and 1024 for ISO

Bit	Name	Description
29	**FIRST_ACK_NUMP_0_MASK**	Interrupt mask for FIRST_ACK_NUMP_0 bit
28	**STREAM_ERROR_MASK**	Interrupt mask for STREAM_ERROR bit
27	**DBTERM_MASK**	Interrupt mask for DBTERM bit
26	**HBTERM_MASK**	Interrupt mask for HBTERM bit
25	**OOSERR_MASK**	Interrupt mask for OOSERR bit

continued on next page

10.16.11 PROT_EPI_CS1 *(continued)*

24	**SHORT_MASK**	Interrupt mask for SHORT bit
23	**ZERO_MASK**	Interrupt mask for ZERO bit
22	**STREAMNRDY_MASK**	Interrupt mask for STREAMNRDY bit
21	**FLOWCONTROL_MASK**	Interrupt mask for FLOWCONTROL bit
20	**RETRY_MASK**	Interrupt mask for RETRY bit
19	**COMMIT_MASK**	Interrupt mask for COMMIT bit
18	**FIRST_ACK_NUMP_0**	The NumP for the first ACK is zero.
17	**STREAM_ERROR**	Stream Error occurred.
16	**DBTERM**	The Burst was terminated by the device when the MULT_TIMER expires.
15	**HBTERM**	The Burst Was terminated by the host.
14	**OOSERR**	Out Of Sequence Error. Anytime an ACK is received with unexpected sequence number request, the ACK will be dropped and intr will be raised
13	**SHORT**	Indicates a shorter-than-maxsize packet was received, but UIB_EPI_XFER_CNT did not reach 0).
12	**ZERO**	Indicates a zero length packet was returned to the host in an IN transaction. Must be cleared by software.
11	**STREAMNRDY**	Nrdy was sent for a bulk stream request because of EP-stream not present in the mapper.
10	**FLOWCONTROL**	EP in flow control due to EPM not being available.
9	**RETRY**	Whenever the USB3.0 does a retry it will asserts this interrupt.
8	**COMMIT**	Set whenever an IN token was ACKed by the host.
5	**STREAM_ERROR_STALL_EN**	Issue STALL whenever Stream error occurs.
4	**EP_RESET**	Per End Point Reset.
3	**STREAM_EN**	Enables bulk stream protocol handling for this EP
2	**STALL**	Set this bit to “1” to stall an endpoint, and to “0” to clear a stall.
1	**NRDY**	Setting this bit causes NRDY on IN transactions.
0	**VALID**	Set VALID=1 to activate an endpoint, and VALID=0 to de-activate it. All USB endpoints default to invalid. An endpoint whose VALID bit is 0 does not respond to any USB traffic.

10.16.12 PROT_EPI_CS2

SuperSpeed IN Endpoint Control and Status Register

There are 16 PROT_EPI_CS2 registers. The address of each is calculated as PROT_EPI_CS2(x) = 0xE0033540 + (x*0x4). Hence PROT_EPI_CS2(0) is at address 0xE0033540, PROT_EPI_CS2(1) is at address 0xE0033540 + 0x4 and so on. The definition of each of these is the same.

PROT_EPI_CS2			SuperSpeed IN Endpoint Control and Status				0xE0033540
b31	b30	b29	b28	b27	b26	b25	b24

PROT_EPI_CS2			SuperSpeed IN Endpoint Control and Status				
b23	b22	b21	b20	b19	b18	b17	b16
							BTERM_NUMP[4]
							R
							R/W
							0

PROT_EPI_CS2			SuperSpeed IN Endpoint Control and Status				
b15	b14	b13	b12	b11	b10	b9	b8
BTERM_NUMP[3:0]				MAXBURST[3:0]			
R	R	R	R	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R	R	R	R
0	0	0	0	0	0	0	0

PROT_EPI_CS2			SuperSpeed IN Endpoint Control and Status				
b7	b6	b5	b4	b3	b2	b1	b0
ISOINPKS[5:0]						TYPE[1:0]	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
16						0	0

Endpoint IN Control and Status:

- Setup USB Package buffering, ISO/BULK/INT, IN/OUT and enable/disable endpoint
- Power up default the payload is 64-byte (Max payload count for Full Speed). In High Speed the payload can be up to 512 for BULK and 1024 for ISO

Bit	Name	Description
16:12	**BTERM_NUMP[4:0]**	Number of packets that need to be cleaned up by CPU whenever there is a BTERM interrupt.
11:8	**MAXBURST[3:0]**	Maximum number of packets the endpoint can send.
7:2	**ISOINPKS[5:0]**	Number of packets to be sent per service interval. Maximum can be 48 (Max burst size* Mult field)
1:0	**TYPE[1:0]**	Endpoint type (EP0 supports CONTROL only) 0 ISO 1 INT 2 BULK 3 CONTROL (only valid for EP0)

10.16.13 PROT_EPI_UNMAPPED_STREAM

Unmapped Stream Request Register

There are 16 PROT_EPI_UNMAPPED_STREAM registers. The address of each is calculated as PROT_EPI_UNMAPPED_STREAM(x) = 0xE0033580 + (x*0x4). Hence PROT_EPI_UNMAPPED_STREAM(0) is at address 0xE0033580, PROT_EPI_UNMAPPED_STREAM(1) is at address 0xE0033580 + 0x4 and so on. The definition of each of these is the same.

PROT_EPI_UNMAPPED_STREAM			Unmapped Stream Request				0xE0033580
b31	b30	b29	b28	b27	b26	b25	b24

PROT_EPI_UNMAPPED_STREAM			Unmapped Stream Request				
b23	b22	b21	b20	b19	b18	b17	b16
				SPSM_STATE[3:0]			
				R	R	R	R
				R/W	R/W	R/W	R/W
				0	0	0	0

PROT_EPI_UNMAPPED_STREAM			Unmapped Stream Request				
b15	b14	b13	b12	b11	b10	b9	b8
STREAM_ID[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PROT_EPI_UNMAPPED_STREAM			Unmapped Stream Request				
b7	b6	b5	b4	b3	b2	b1	b0
STREAM_ID[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

A register for each Endpoint to handle a request for a StreamID that is not currently mapped to a socket.

Bit	Name	Description
19:16	SPSM_STATE[3:0]	Stream Protocol State Machine (SPSM) State for this EndPoint: 0 Not Configured 1 Disabled 2 Prime Pipe 3 DFR Prime Pipe 4 Idle 5 Start Stream 6 Move Data 7 End 8 Error
15:0	STREAM_ID[15:0]	The StreamID of the current stream activated (or requested to be activated) by the protocol layer.

10.16.14 PROT_EPI_MAPPED_STREAM

Mapped Streams Registers

There are 16 PROT_EPI_MAPPED_STREAM registers. The address of each is calculated as PROT_EPI_MAPPED_STREAM(x) = 0xE00335C0 + (x*0x4). Hence PROT_EPI_MAPPED_STREAM(0) is at address 0xE00335C0, PROT_EPI_MAPPED_STREAM(1) is at address 0xE00335C0 + 0x4 and so on. The definition of each of these is the same.

PROT_EPI_MAPPED_STREAM			Mapped Streams Register				0xE00335C0
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE	UNMAP	UNMAPPED					
R/W	R/W	R					
R	R	R/W					
0	0	0					

PROT_EPI_MAPPED_STREAM			Mapped Streams Register				
b23	b22	b21	b20	b19	b18	b17	b16
				EP_NUMBER[3:0]			
				R/W	R/W	R/W	R/W
				R	R	R	R
				0	0	0	0

PROT_EPI_MAPPED_STREAM			Mapped Streams Register				
b15	b14	b13	b12	b11	b10	b9	b8
STREAM_ID[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

PROT_EPI_MAPPED_STREAM			Mapped Streams Register				
b7	b6	b5	b4	b3	b2	b1	b0
STREAM_ID[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

A register for each socket indicating which EP and which StreamID is mapped to it.

Bit	Name	Description
31	**ENABLE**	Set by firmware if a stream is mapped to the corresponding socket. If this bit is set, the endpoint number corresponding to this socket number can no longer be used in non-streaming mode (that would create a conflict of two endpoints wanting to use the same socket).
30	**UNMAP**	Request to unmap this stream. May be cleared to revert/withdraw request.
29	**UNMAPPED**	Stream is unmapped (not in use by the corresponding EP's SPSM).
19:16	**EP_NUMBER[3:0]**	The Endpoint number of the stream connected to the corresponding socket by firmware.
15:0	**STREAM_ID[15:0]**	The StreamID of the stream connected to the corresponding socket by firmware.

10.16.15 PROT_EPO_CS1

SuperSpeed OUT Endpoint Control and Status Register

There are 16 PROT_EPO_CS1 registers. The address of each is calculated as PROT_EPO_CS1(x) = 0xE0033600 + (x*0x4). Hence PROT_EPO_CS1(0) is at address 0xE0033600, PROT_EPO_CS1(1) is at address 0xE0033600 + 0x4 and so on. The definition of each of these is the same.

PROT_EPO_CS1		SuperSpeed OUT Endpoint Control and Status					0xE0033600
b31	b30	b29	b28	b27	b26	b25	b24
		FIRST_ACK_ NUMP_0_MASK	STREAM_ERROR _MASK	DBTERM_MASK	HBTERM_MASK	OOSERR_MASK	SHORT_MASK
		R/W	R/W	R/W	R/W	R/W	R/W
		R	R	R	R	R	R
		0	0	0	0	0	0

PROT_EPO_CS1		SuperSpeed OUT Endpoint Control and Status					
b23	b22	b21	b20	b19	b18	b17	b16
ZERO_MASK	STREAMNRDY_ MASK	FLOWCONTROL_ MASK	RETRY_MASK	COMMIT_MASK	FIRST_ACK_ NUMP_0	STREAM_ERROR	DBTERM
R/W	R/W	R/W	R/W	R/W	R/W1C	R/W1C	R/W1C
R	R	R	R	R	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

PROT_EPO_CS1		SuperSpeed OUT Endpoint Control and Status					
b15	b14	b13	b12	b11	b10	b9	b8
HBTERM	OOSERR	SHORT	ZERO	STREAMNRDY	FLOWCONTROL	RETRY	COMMIT
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

PROT_EPO_CS1		SuperSpeed OUT Endpoint Control and Status					
b7	b6	b5	b4	b3	b2	b1	b0
		STREAM_ERROR _STALL_EN	EP_RESET	STREAM_EN	STALL	NRDY	VALID
		R/W	R/W	R/W	R/W	R/W	R/W
		R	R	R	R	R	R
		0	0	0	0	0	0

Endpoint OUT Control and Status

Bit	Name	Description
29	FIRST_ACK_NUMP_0_MASK	Interrupt mask for FIRST_ACK_NUMP_0 bit
28	STREAM_ERROR_MASK	Interrupt mask for STREAM_ERROR bit
27	DBTERM_MASK	Interrupt mask for DBTERM bit
26	HBTERM_MASK	Interrupt mask for HBTERM bit
25	OOSERR_MASK	Interrupt mask for OOSERR bit
24	SHORT_MASK	Interrupt mask for SHORT bit
23	ZERO_MASK	Interrupt mask for ZERO bit

continued on next page

10.16.15 PROT_EPO_CS1 *(continued)*

22	**STREAMNRDY_MASK**	Interrupt mask for STREAMNRDY bit
21	**FLOWCONTROL_MASK**	Interrupt mask for FLOWCONTROL bit
20	**RETRY_MASK**	Interrupt mask for RETRY bit
19	**COMMIT_MASK**	Interrupt mask for COMMIT bit
18	**FIRST_ACK_NUMP_0**	The NumP for the first ACK is zero.
17	**STREAM_ERROR**	Stream Error occurred.
16	**DBTERM**	The Burst was terminated by the device when the MULT_TIMER expires.
15	**HBTERM**	The Burst Was terminated by the host.
14	**OOSERR**	Out Of Sequence Error. Anytime an OUT-DATA is received with unexpected sequence number request, the data will be dropped and intr will be raised
13	**SHORT**	Indicates a shorter-than-maxsize packet was received.
12	**ZERO**	Indicates a zero length packet was received by the device in an OUT transaction. Must be cleared by software.
11	**STREAMNRDY**	Nrdy was sent for a bulk stream request because of EP-stream not present in the mapper.
10	**FLOWCONTROL**	EP in flow control due to EPM not being available.
9	**RETRY**	Whenever the USB3.0 does a retry it will asserts this interrupt.
8	**COMMIT**	Set whenever an OUT DATA was committed into the EPM.
5	**STREAM_ERROR_STALL_EN**	Issue STALL whenever Stream error occurs.
4	**EP_RESET**	Per End Point Reset.
3	**STREAM_EN**	Enables bulk stream protocol handling for this EP
2	**STALL**	Set this bit to “1” to stall an endpoint, and to “0” to clear a stall.
1	**NRDY**	Setting this bit causes NRDY on IN transactions.
0	**VALID**	Set VALID=1 to activate an endpoint, and VALID=0 to de-activate it. All USB endpoints default to invalid. An endpoint whose VALID bit is 0 does not respond to any USB traffic.

10.16.16 PROT_EPO_CS2

SuperSpeed OUT Endpoint Control and Status Register

There are 16 PROT_EPO_CS2 registers. The address of each is calculated as PROT_EPO_CS2(x) = 0xE0033640 + (x*0x4). Hence PROT_EPO_CS2(0) is at address 0xE0033640, PROT_EPO_CS2(1) is at address 0xE0033640 + 0x4 and so on. The definition of each of these is the same.

PROT_EPO_CS2			SuperSpeed OUT Endpoint Control and Status				0xE0033640
b31	b30	b29	b28	b27	b26	b25	b24

PROT_EPO_CS2			SuperSpeed OUT Endpoint Control and Status				
b23	b22	b21	b20	b19	b18	b17	b16

PROT_EPO_CS2			SuperSpeed OUT Endpoint Control and Status				
b15	b14	b13	b12	b11	b10	b9	b8
				MAXBURST[3:0]			
				R/W	R/W	R/W	R/W
				R	R	R	R
				0	0	0	0

PROT_EPO_CS2			SuperSpeed OUT Endpoint Control and Status				
b7	b6	b5	b4	b3	b2	b1	b0
ISOINPKS[5:0]						TYPE[1:0]	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
16						0	0

Endpoint IN Control and Status:

- Setup USB Package buffering, ISO/BULK/INT, IN/OUT and enable/disable endpoint
- Power up default the payload is 64-byte (Max payload count for Full Speed). In High Speed the payload can be up to 512 for BULK and 1024 for ISO

Bit	Name	Description
11:8	**MAXBURST[3:0]**	Maximum number of packets the endpoint can send. (truncated to 4b, 0 means 16)
7:2	**ISOINPKS[5:0]**	Number of packets to be sent per service interval. Maximum can be 48 (Max burst size* Mult field)
1:0	**TYPE[1:0]**	Endpoint type (EP0 supports CONTROL only) 0 ISO 1 INT 2 BULK 3 CONTROL (only valid for EP0)

10.16.17 PROT_EPO_UNMAPPED_STREAM

Unmapped Stream Request Register

There are 16 PROT_EPO_UNMAPPED_STREAM registers. The address of each is calculated as PROT_EPO_UNMAPPED_STREAM(x) = 0xE0033680 + (x*0x4). Hence PROT_EPO_UNMAPPED_STREAM(0) is at address 0xE0033680, PROT_EPO_UNMAPPED_STREAM(1) is at address 0xE0033680 + 0x4 and so on. The definition of each of these is the same.

PROT_EPO_UNMAPPED_STREAM			Unmapped Stream Request				0xE0033680
b31	b30	b29	b28	b27	b26	b25	b24

PROT_EPO_UNMAPPED_STREAM			Unmapped Stream Request				
b23	b22	b21	b20	b19	b18	b17	b16
				SPSM_STATE[3:0]			
				R	R	R	R
				R/W	R/W	R/W	R/W
				0	0	0	0

PROT_EPO_UNMAPPED_STREAM			Unmapped Stream Request				
b15	b14	b13	b12	b11	b10	b9	b8
STREAM_ID[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PROT_EPO_UNMAPPED_STREAM			Unmapped Stream Request				
b7	b6	b5	b4	b3	b2	b1	b0
STREAM_ID[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

A register for each Endpoint to handle a request for a StreamID that is not currently mapped to a socket.

Bit	Name	Description
19:16	**SPSM_STATE[3:0]**	Stream Protocol State Machine (SPSM) State for this EndPoint: 0 Not Configured 1 Disabled 2 Prime Pipe 3 DFR Prime Pipe 4 Idle 5 Start Stream 6 Move Data 7 End 8 Error
15:0	**STREAM_ID[15:0]**	The StreamID of the current stream activated (or requested to be activated) by the protocol layer.

10.16.18 PROT_EPO_MAPPED_STREAM

Mapped Streams Registers

There are 16 PROT_EPO_MAPPED_STREAM registers. The address of each is calculated as PROT_EPO_MAPPED_STREAM(x) = 0xE00336C0 + (x*0x4). Hence PROT_EPO_MAPPED_STREAM(0) is at address 0xE00336C0, PROT_EPO_MAPPED_STREAM(1) is at address 0xE00336C0 + 0x4 and so on. The definition of each of these is the same.

PROT_EPO_MAPPED_STREAM			Mapped Streams Register				0xE00336C0
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE	UNMAP	UNMAPPED					
R/W	R/W	R					
R	R	R/W					
0	0	0					

PROT_EPO_MAPPED_STREAM			Mapped Streams Register				
b23	b22	b21	b20	b19	b18	b17	b16
				EP_NUMBER[3:0]			
				R/W	R/W	R/W	R/W
				R	R	R	R
				0	0	0	0

PROT_EPO_MAPPED_STREAM			Mapped Streams Register				
b15	b14	b13	b12	b11	b10	b9	b8
STREAM_ID[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

PROT_EPO_MAPPED_STREAM			Mapped Streams Register				
b7	b6	b5	b4	b3	b2	b1	b0
STREAM_ID[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

A register for each socket indicating which EP and which StreamID is mapped to it. Before clearing ENABLE or changing STREAM_ID/EP_NUMBER while ENABLE=1, firmware must ensure the socket is not in use by setting UNMAP and checking UNMAPPED went high. If UNMAP=1 is not followed by UNMAPPED=1 within ~100ns, the firmware can withdraw the request by resetting UNMAP to 0.

Bit	Name	Description
31	**ENABLE**	Set by firmware if a stream is mapped to the corresponding socket. If this bit is set, the endpoint number corresponding to this socket number can no longer be used in non-streaming mode (that would create a conflict of two endpoints wanting to use the same socket).
30	**UNMAP**	Request to unmap this stream. May be cleared to revert/withdraw request.
29	**UNMAPPED**	Stream is unmapped (not in use by the corresponding EP's SPSM).
19:16	**EP_NUMBER[3:0]**	The Endpoint number of the stream connected to the corresponding socket by firmware.
15:0	**STREAM_ID[15:0]**	The StreamID of the stream connected to the corresponding socket by firmware.

10.17 USB Port - SuperSpeed Ingress Socket Registers

10.17.1 UIBIN_ID

Block Identification and Version Number Register

UIBIN_ID			Block Identification and Version Number Register				0xE0040000
b31	b30	b29	b28	b27	b26	b25	b24
BLOCK_VERSION[15:8]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
UIBIN_ID			**Block Identification and Version Number Register**				
b23	b22	b21	b20	b19	b18	b17	b16
BLOCK_VERSION[7:0]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0x0001							
UIBIN_ID			**Block Identification and Version Number Register**				
b15	b14	b13	b12	b11	b10	b9	b8
BLOCK_ID[15:8]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
UIBIN_ID			**Block Identification and Version Number Register**				
b7	b6	b5	b4	b3	b2	b1	b0
BLOCK_ID[7:0]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0x0004							

Every IP block will implement a few MMIO registers at offset 0 in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31:16	**BLOCK_VERSION[15:0]**	Version number for the IP
15:8	**BLOCK_ID[15:0]**	A unique number identifying the IP in the memory space

10.17.2 UIBIN_POWER

Power, Clock, and Reset Control Register

UIBIN_POWER	Power, Clock, and Reset Control Registers						0xE0040004
b31	b30	b29	b28	b27	b26	b25	b24
RESETN							
R/W							
W							
0							

UIBIN_POWER	Power, Clock, and Reset Control Registers						
b23	b22	b21	b20	b19	b18	b17	b16

UIBIN_POWER	Power, Clock, and Reset Control Registers						
b15	b14	b13	b12	b11	b10	b9	b8

UIBIN_POWER	Power, Clock, and Reset Control Registers						
b7	b6	b5	b4	b3	b2	b1	b0
							ACTIVE
							R
							W
							0

Every IP block will implement a few MMIO registers at offset 0 in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31	**RESETN**	Active low reset signal for all logic in the block. Note that reset is active on all flops in the block when either system reset is asserted (RESET# pin or SYSTEM_POWER.RESETN is asserted) or this signal is active. After setting this bit to 1, firmware will poll and wait for the 'active' bit to assert. Reading '1' from 'resetn' does not indicate the block is out of reset – this may take some time depending on initialization tasks and clock frequencies. This bit is nonfunctional for UIBIN and will not reset anything. Use UIB_POWER register instead.
0	**ACTIVE**	For blocks that must perform initialization after reset before becoming operational, this signal will remain deasserted until initialization is complete. In other words, reading active = 1 indicates block is initialized and ready for operation. This bit is a copy of UIB_POWER.active. Preferably use UIB_POWER register instead.

10.18 I2S Registers

10.18.1 I2S_CONFIG

I2S Configuration and Mode Register

I2S_CONFIG			I2S Configuration and Mode Register				0xE0000000
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE	**TX_CLEAR**						
R/W	R/W						
R	R						
0	0						

I2S_CONFIG			I2S Configuration and Mode Register				
b23	b22	b21	b20	b19	b18	b17	b16

I2S_CONFIG			I2S Configuration and Mode Register				
b15	b14	b13	b12	b11	b10	b9	b8
			MODE[1:0]		**BIT_WIDTH[2:0]**		
			R/W	R/W	R/W	R/W	R/W
			R	R	R	R	R
			0	0	1		

I2S_CONFIG			I2S Configuration and Mode Register				
b7	b6	b5	b4	b3	b2	b1	b0
	DMA_MODE	**MONO**	**FIXED_SCK**	**WSMODE**	**ENDIAN**	**MUTE**	**PAUSE**
	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	R	R	R	R	R	R	R
	0	0	0	0	0	1	0

Bit	Name	Description
31	**ENABLE**	Enable block here, but only after all the configuration is set. Do not set this bit to 1 while changing any other value in this register. This bit will be synchronized to the core clock. Setting this bit to 0 will complete transmission of current sample. When DMA_MODE=1 any remaining samples in the pipeline are discarded. When DMA_MODE=0 no samples are lost.
30	**TX_CLEAR**	Use only when ENABLE=0; behavior undefined when ENABLE=1 0 Do nothing 1 Clear transmit FIFO (After TX_CLEAR is set, software must wait for TX*_DONE before clearing it)
12:11	**MODE[1:0]**	0,3 I2S Mode 1 Left Justified Mode 2 Right Justified Mode.

continued on next page

10.18.1 I2S_CONFIG *(continued)*

10:8	**BIT_WIDTH[2:0]**	0	8-bit
		1	16-bit
		2	18-bit
		3	24-bit
		4	32-bit
		5-7	Reserved
6	**DMA_MODE**	0	Register-based transfers
		1	DMA-based transfers
5	**MONO**	0	Stereo
		1	Mono> Read samples from the left channel and send in out on both the channels.
4	**FIXED_SCK**	0	SCK = 16*WS, 32*WS or 64* WS for 8-bit, 16-bit and 32-bit width, 64*WS otherwise.
		1	SCK=64*WS
3	**WSMODE**	I2S_MODE:	
		0	WS=0 will denote the left Channel
		1	WS=0 will denote the right channel.
		In left/right justified modes:	
		1	WS=0 will denote the left Channel
		0	WS=0 will denote the right channel.
2	**ENDIAN**	0	MSB First
		1	LSB First
1	**MUTE**	Discard the value read from the DMA and transmit zeros instead. Continue to read input samples at normal rate.	
0	**PAUSE**	Pause transmission, transmit 0s. Setting this bit to 1 will not discard any samples. Later clearing this bit will resume output at exact same spot. In paused mode the I2S clock will continue to transmit 0-value samples. A small, integral, but undefined number of samples will be transmitted after this bit is set to 1 (to ensure no hanging samples). When one of the descriptors is modified in socket, no samples from the old descriptor will be output (all FIFO's will be cleared).	

10.18.2 I2S_STATUS

I2S Status Register

I2S_STATUS			I2S Status Register				0xE0000004
b31	b30	b29	b28	b27	b26	b25	b24
			BUSY	ERROR_CODE[3:0]			
			R	R	R	R	R
			W	W	W	W	W
			0	0xF			

I2S_STATUS			I2S Status Register				
b23	b22	b21	b20	b19	b18	b17	b16

I2S_STATUS							I2S Status Register
b15	b14	b13	b12	b11	b10	b9	b8
							ERROR
							R/W1C
							R/W1S
							0

I2S_STATUS				I2S Status Register			
b7	b6	b5	b4	b3	b2	b1	b0
NO_DATA	PAUSED	TXR_HALF	TXR_SPACE	TXR_DONE	TXL_HALF	TXL_SPACE	TXL_DONE
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	1	1	0	1	1	0

Status and error register. Most status bits are used to generate an interrupt on positive edge into INTR register.

Bit	Name	Description
28	**BUSY**	Indicates the block is busy transmitting data. This field may remain asserted after the block is suspended and must be polled before changing any configuration values.
27:24	**ERROR_CODE[3:0]**	Error code, only relevant when ERROR=1. ERROR logs only the FIRST error to occur and will never change value as long as ERROR=1. 11 Left TX FIFO/DMA socket underflow 12 Right TX FIFO/DMA socket underflow 13 Write to left TX FIFO when FIFO full 14 Write to right TX FIFO when FIFO full 15 No error
8	**ERROR**	An internal error has occurred with cause ERROR_CODE. Must be cleared by software. Sticky
7	**NO_DATA**	No data is currently available for output, but socket does not indicate empty. Only relevant when DMA_MODE=1. Non sticky.
6	**PAUSED**	Output is paused (PAUSE has taken effect). Non sticky

continued on next page

10.18.2 I2S_STATUS *(continued)*

5	**TXR_HALF**	Indicates that the right TX FIFO is at least half empty. This bit can be used to create burst-based interrupts. This bit is updated immediately after writes to EGRESS_DATA_RIGHT register. Only relevant when DMA_MODE=0. Non sticky.
4	**TXR_SPACE**	Indicates space is available in the right TX FIFO. This bit is updated immediately after writes to EGRESS_DATA_RIGHT register. Only relevant when DMA_MODE=0. Non sticky.
3	**TXR_DONE**	Indicates no more data is available for transmission on right channel. Non sticky. If DMA_MODE=0 this is defined as TX FIFO empty and shift register empty but will assert only when ENABLE=0. If DMA_MODE=1 this is defined as socket is end of transfer (EOT) and shift register empty. Note that this field will only assert after a transmission was started - it's power up state is 0.
2	**TXL_HALF**	Indicates that the left TX FIFO is at least half empty. This bit can be used to create burst-based interrupts. This bit is updated immediately after writes to EGRESS_DATA_LEFT register. Only relevant when DMA_MODE=0. Non sticky.
1	**TXL_SPACE**	Indicates space is available in the left TX FIFO. This bit is updated immediately after writes to EGRESS_DATA_LEFT register. Only relevant when DMA_MODE=0. Non sticky.
0	**TXL_DONE**	Indicates no more data is available for transmission on left channel. Non sticky. If DMA_MODE=0 this is defined as TX FIFO empty and shift register empty but will assert only when ENABLE=0. If DMA_MODE=1 this is defined as socket is EOT and shift register empty. Note that this field will only assert after a transmission was started - it's power up state is 0.

10.18.3 I2S_INTR

I2S Interrupt Request Register

I2S_INTR	I2S Interrupt Request Register						0xE0000008
b31	b30	b29	b28	b27	b26	b25	b24

I2S_INTR	I2S Interrupt Request Register						
b23	b22	b21	b20	b19	b18	b17	b16

I2S_INTR	I2S Interrupt Request Register						
b15	b14	b13	b12	b11	b10	b9	b8
							ERROR
							R/W1C
							R/W1S
							0

I2S_INTR	I2S Interrupt Request Register						
b7	b6	b5	b4	b3	b2	b1	b0
NO_DATA	**PAUSED**	**TXR_HALF**	**TXR_SPACE**	**TXR_DONE**	**TXL_HALF**	**TXL_SPACE**	**TXL_DONE**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

Interrupt requests. Derived from STATUS register. Interrupt requests must be cleared by CPU and are generated regardless of values in INTR_MASK register.

Bit	Name	Description
8	**ERROR**	Set by hardware when corresponding STATUS asserts, cleared by software.
7	**NO_DATA**	Set by hardware when corresponding STATUS asserts, cleared by software.
6	**PAUSED**	Set by hardware when corresponding STATUS asserts, cleared by software.
5	**TXR_HALF**	Set by hardware when corresponding STATUS asserts, cleared by software.
4	**TXR_SPACE**	Set by hardware when corresponding STATUS asserts, cleared by software.
3	**TXR_DONE**	Set by hardware when corresponding STATUS asserts, cleared by software.
2	**TXL_HALF**	Set by hardware when corresponding STATUS asserts, cleared by software.
1	**TXL_SPACE**	Set by hardware when corresponding STATUS asserts, cleared by software.
0	**TXL_DONE**	Set by hardware when corresponding STATUS asserts, cleared by software.

10.18.4 I2S_INTR_MASK

I2S Interrupt Mask Register

I2S_INTR_MASK	I2S Interrupt Mask Register						0xE000000C
b31	b30	b29	b28	b27	b26	b25	b24

I2S_INTR_MASK	I2S Interrupt Mask Register						
b23	b22	b21	b20	b19	b18	b17	b16

I2S_INTR_MASK	I2S Interrupt Mask Register						
b15	b14	b13	b12	b11	b10	b9	b8
							ERROR
							R/W
							R
							0

I2S_INTR_MASK	I2S Interrupt Mask Register						
b7	b6	b5	b4	b3	b2	b1	b0
NO_DATA	**PAUSED**	**TXR_HALF**	**TXR_SPACE**	**TXR_DONE**	**TXL_HALF**	**TXL_SPACE**	**TXL_DONE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Interrupt mask bits. Determine if INTR bits are reported to CPU. Have no effect on creation of INTR bits.

Bit	Name	Description	
8	**ERROR**	1	Report interrupt to CPU
7	**NO_DATA**	1	Report interrupt to CPU
6	**PAUSED**	1	Report interrupt to CPU
5	**TXR_HALF**	1	Report interrupt to CPU
4	**TXR_SPACE**	1	Report interrupt to CPU
3	**TXR_DONE**	1	Report interrupt to CPU
2	**TXL_HALF**	1	Report interrupt to CPU
1	**TXL_SPACE**	1	Report interrupt to CPU
0	**TXL_DONE**	1	Report interrupt to CPU

10.18.5 I2S_EGRESS_DATA_LEFT

I2S Egress Data Register (Left)

I2S_EGRESS_DATA_LEFT			I2S Egress Data Register (Left)				0xE0000010
b31	b30	b29	b28	b27	b26	b25	b24
DATA[31:24]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2S_EGRESS_DATA_LEFT			**I2S Egress Data Register (Left)**				
b23	b22	b21	b20	b19	b18	b17	b16
DATA[23:16]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2S_EGRESS_DATA_LEFT			**I2S Egress Data Register (Left)**				
b15	b14	b13	b12	b11	b10	b9	b8
DATA[15:8]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2S_EGRESS_DATA_LEFT			**I2S Egress Data Register (Left)**				
b7	b6	b5	b4	b3	b2	b1	b0
DATA[7:0]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Accepts egress data one sample at a time, LSB justified. Writing to this register adds one sample to the FIFO if the FIFO has space available. It will result in ERROR if the FIFO is full. The size of the transmit FIFO is configured at design time.

Bit	Name	Description
31:0	**DATA[31:0]**	Sample to be written to the peripheral in registered mode. Number of bits taken depends on sample size (see I2S_CONFIG), other bits are ignored.

10.18.6 I2S_EGRESS_DATA_RIGHT

I2S Egress Data Register (Right)

I2S_EGRESS_DATA_RIGHT			I2S Egress Data Register (Right)				0xE0000014
b31	b30	b29	b28	b27	b26	b25	b24
DATA[31:24]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2S_EGRESS_DATA_RIGHT			**I2S Egress Data Register (Right)**				
b23	b22	b21	b20	b19	b18	b17	b16
DATA[23:16]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2S_EGRESS_DATA_RIGHT			**I2S Egress Data Register (Right)**				
b15	b14	b13	b12	b11	b10	b9	b8
DATA[15:8]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2S_EGRESS_DATA_RIGHT			**I2S Egress Data Register (Right)**				
b7	b6	b5	b4	b3	b2	b1	b0
DATA[7:0]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Accepts egress data one sample at a time, LSB justified. Writing to this register adds one sample to the FIFO if the FIFO has space available. It will result in ERROR if the FIFO is full. The size of the transmit FIFO is configured at design time.

Bit	Name	Description
31:0	**DATA[31:0]**	Sample to be written to the peripheral in registered mode. Number of bits taken depends on sample size (see I2S_CONFIG), other bits are ignored.

10.18.7 I2S_COUNTER

I2S Sample Counter Register

I2S_COUNTER			I2S Sample Counter Register				0xE0000018
b31	b30	b29	b28	b27	b26	b25	b24
COUNTER[31:24]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
I2S_COUNTER			**I2S Sample Counter Register**				
b23	b22	b21	b20	b19	b18	b17	b16
COUNTER[23:16]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
I2S_COUNTER			**I2S Sample Counter Register**				
b15	b14	b13	b12	b11	b10	b9	b8
COUNTER[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
I2S_COUNTER			**I2S Sample Counter Register**				
b7	b6	b5	b4	b3	b2	b1	b0
COUNTER[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Counts number of samples written to output (L+R counts as one), can be used in software PLL to control audio decode thread.

Bit	Name	Description
31:0	**COUNTER[31:0]**	Counter increments by one for every sample written on output. Counts L+R as one sample in stereo mode. This counter is more reliable to implement a software PLL because of more periodic behavior. This counter will be reset to 0 when I2S_CONFIG.ENABLE=0.

10.18.8 I2S_SOCKET

I2S Socket Register

I2S_SOCKET			I2S Socket Register				0xE0000304
b31	b30	b29	b28	b27	b26	b25	b24

I2S_SOCKET			I2S Socket Register				
b23	b22	b21	b20	b19	b18	b17	b16

I2S_SOCKET			I2S Socket Register				
b15	b14	b13	b12	b11	b10	b9	b8
RIGHT_SOCKET[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

I2S_SOCKET			I2S Socket Register				
b7	b6	b5	b4	b3	b2	b1	b0
LEFT_SOCKET[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Indicates sockets used for DMA based operation. Left and right socket must be different for stereo operation.

Bit	Name	Description
15:8	**RIGHT_SOCKET[7:0]**	Socket number for right data samples 0-7 Supported 8-.. Reserved
7:0	**LEFT_SOCKET[7:0]**	Socket number for left data samples 0-7 Supported 8-.. Reserved

10.18.9 I2S_ID

Block Identification and Version Number Register

I2S_ID				Block Identification and Version Number			0xE00003F0
b31	b30	b29	b28	b27	b26	b25	b24
BLOCK_VERSION[15:8]							
R	R	R	R	R	R	R	R

I2S_ID				Block Identification and Version Number			
b23	b22	b21	b20	b19	b18	b17	b16
BLOCK_VERSION[7:0]							
R	R	R	R	R	R	R	R
0x0001							

I2S_ID				Block Identification and Version Number			
b15	b14	b13	b12	b11	b10	b9	b8
BLOCK_ID[15:8]							
R	R	R	R	R	R	R	R

I2S_ID				Block Identification and Version Number			
b7	b6	b5	b4	b3	b2	b1	b0
BLOCK_ID[7:0]							
R	R	R	R	R	R	R	R
0x0000							

Every low performance peripheral IP block will implement a few MMIO registers at a fixed offset in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31:16	**BLOCK_VERSION[15:0]**	Version number for the IP
15:0	**BLOCK_ID[15:0]**	A unique number identifying the IP in the memory space

10.18.10 I2S_POWER

Power, Clock, and Reset Control Register

I2S_POWER			Power, Clock, and Reset Control				0xE00003F4
b31	b30	b29	b28	b27	b26	b25	b24
RESETN							
R/W							
R							
0							

I2S_POWER			Power, Clock, and Reset Control				
b23	b22	b21	b20	b19	b18	b17	b16

I2S_POWER			Power, Clock, and Reset Control				
b15	b14	b13	b12	b11	b10	b9	b8

I2S_POWER			Power, Clock, and Reset Control				
b7	b6	b5	b4	b3	b2	b1	b0
							ACTIVE
							R
							W
							0

Every low performance peripheral IP block will implement a few MMIO registers at a fixed offset in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31	**RESETN**	Active LOW reset signal for all logic in the block. Note that reset is active on all flops in the block when either system reset is asserted (RESET# pin or SYSTEM_POWER.RESETN is asserted) or this signal is active. After setting this bit to 1, firmware will poll and wait for the ‘active’ bit to assert. Reading ‘1’ from ‘resetn’ does not indicate the block is out of reset – this may take some time depending on initialization tasks and clock frequencies. This bit must be asserted ('0') for at least 10 µs for effective reset.
0	**ACTIVE**	For blocks that must perform initialization after reset before becoming operational, this signal will remain deasserted until initialization is complete. In other words reading active=1 indicates block is initialized and ready for operation.

10.19 I²C Registers

10.19.1 I2C_CONFIG

I²C Configuration and Mode Register

I2C_CONFIG			I²C Configuration and Mode Register				0xE0000400
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE	**TX_CLEAR**	**RX_CLEAR**					
R/W	R/W	R/W					
R	R	R					
0	0	0					

I2C_CONFIG			I²C Configuration and Mode Register				
b23	b22	b21	b20	b19	b18	b17	b16

I2C_CONFIG			I²C Configuration and Mode Register				
b15	b14	b13	b12	b11	b10	b9	b8

I2C_CONFIG			I²C Configuration and Mode Register				
b7	b6	b5	b4	b3	b2	b1	b0
					I2C_100KHz	**CONTINUE_ON_NACK**	**DMA_MODE**
					R/W	R/W	R/W
					R	R	R
					1	0	0

Various modes of the I²C block

Bit	Name	Description
31	**ENABLE**	Enable block here, but only after all the configuration is set. Do not set this bit to 1 while changing any other configuration value in this register. This bit will be synchronized to the core clock. Disabling blocks resets all I²C controller state machine and stops all transfers at the end of current byte. When DMA_MODE=1, data hanging in the transmit pipeline may be lost. Any unread data in the ingress data register is lost.
30	**TX_CLEAR**	Use only when ENABLE=0; behavior undefined when ENABLE=1 0 Do nothing 1 Clear transmit FIFO (After TX_CLEAR is set, software must wait for TX_DONE before clearing it)
29	**RX_CLEAR**	Use only when ENABLE=0; behavior undefined when ENABLE=1 0 Do nothing 1 Clear receive FIFO (Software must wait for RX_DATA=0 before clearing this bit again)

continued on next page

10.19.1 I2C_CONFIG *(continued)*

2	**I2C_100KHz**	0	Other speeds, use 40% duty cycle while generating SCLK from 10X clock.
		1	I^2C is in 100KHz mode, use 50% duty cycle.
1	**CONTINUE_ON_NACK**	1	Continue transmission even if NAK is received. It is strongly advised to use this bit for debugging purposes only. This bit is overridden in preamble repeat feature.
0	**DMA_MODE**	0	Register-based transfers
		1	DMA-based transfers

10.19.2 I2C_STATUS

I²C Status Register

I2C_STATUS			I²C Status Register				0xE0000404
b31	b30	b29	b28	b27	b26	b25	b24
SDA_STAT	**SCL_STAT**	**BUS_BUSY**	**BUSY**	**ERROR_CODE[3:0]**			
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0xF			

I2C_STATUS			I²C Status Register				
b23	b22	b21	b20	b19	b18	b17	b16

I2C_STATUS				I²C Status Register			
b15	b14	b13	b12	b11	b10	b9	b8
							ERROR
							R/W1C
							R/W1S
							0

I2C_STATUS				I²C Status Register			
b7	b6	b5	b4	b3	b2	b1	b0
LOST_ ARBITRATION	**TIMEOUT**	**TX_HALF**	**TX_SPACE**	**TX_DONE**	**RX_HALF**	**RX_DATA**	**RX_DONE**
R/W1C	R/W1C	R	R	R	R	R	R
R/W1S	R/W1S	W	W	W	W	W	W
0	0	1	1	0	0	0	0

Status and error register. Most status bits are used to generate an interrupt on positive edge into INTR register.

Bit	Name	Description
31	**SDA_STAT**	Status of the SDA line.
30	**SCL_STAT**	Status of the SCL line.
29	**BUS_BUSY**	Asserts when the block has detected that it cannot start an operation (TX/RX) since the bus is kept busy by another master. Deasserts and resets when a stop condition is detected or when the block is disabled.
28	**BUSY**	Indicates the block is busy transmitting data. This field may remain asserted after the block is suspended and must be polled before changing any configuration values.

continued on next page

10.19.2 I2C_STATUS *(continued)*

27:24	**ERROR_CODE**	Error code, only relevant when ERROR=1. ERROR codes 0 through 7 are relevant for TIMEOUT as well and are used to pin-point when timeout happened (at pre-amble byte X). ERROR logs only the FIRST error to occur and will never change value as long as ERROR=1. Values detailed in BROS 0-7 Slave NAKed the corresponding byte in the preamble. 8 Slave NAKed in data phase. 9 Preamble Repeat exited due to NACK or ACK. 10 Preamble repeat-count reached without satisfying NACK.ACK conditions. 11 TX Underflow 12 Write to TX FIFO when FIFO full 13 Read from RX FIFO when FIFO empty 14 RX Overflow 15 No error
8	**ERROR**	An internal error has occurred with cause ERROR_CODE. Must be cleared by software. Sticky
7	**LOST_ARBITRATION**	Master lost arbitration during command. Software is responsible for resetting socket (in DMA_MODE) and reissuing command. Sticky
6	**TIMEOUT**	An I^2C bus timeout occurred (see I2C_TIMEOUT register). Sticky
5	**TX_HALF**	Indicates that the TX FIFO is at least half empty. This bit can be used to create burst-based interrupts. This bit is updated immediately after writes to EGRESS_DATA register. Non sticky.
4	**TX_SPACE**	Indicates space is available in the TX FIFO. This bit is updated immediately after writes to EGRESS_DATA register. Non sticky.
3	**TX_DONE**	Indicates no more data is available for transmission. Non sticky. If DMA_MODE=0 this is defined as TX FIFO empty and shift register empty. If DMA_MODE=1 this is defined as BYTES_TARNSFERRED=BYTE_COUNT and shift register empty. Note that this field will only assert after a transmission was started - it's power up state is 0.
2	**RX_HALF**	Indicates that the RX FIFO is at least half full (only relevant when DMA_MODE=0). This bit can be used to create burst based interrupts. This bit is updated immediately after reads from INGRESS_DATA register. Non sticky
1	**RX_DATA**	Indicates data is available in the RX FIFO (only relevant when DMA_MODE=0). This bit is updated immediately after reads from INGRESS_DATA register. Non sticky
0	**RX_DONE**	Indicates receive operation completed. Non sticky, Does not need software intervention to clear it.

10.19.3 I2C_INTR

I²C Interrupt Request Register

I2C_INTR	I²C Interrupt Request Register						0xE0000408
b31	b30	b29	b28	b27	b26	b25	b24

I2C_INTR	I²C Interrupt Request Register						
b23	b22	b21	b20	b19	b18	b17	b16

I2C_INTR	I²C Interrupt Request Register						
b15	b14	b13	b12	b11	b10	b9	b8
							ERROR
							R/W1C
							R/W1S
							0

I2C_INTR	I²C Interrupt Request Register						
b7	b6	b5	b4	b3	b2	b1	b0
LOST_ ARBITRATION	**TIMEOUT**	**TX_HALF**	**TX_SPACE**	**TX_DONE**	**RX_HALF**	**RX_DATA**	**RX_DONE**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

Interrupt requests. Derived from STATUS register. Interrupt requests must be cleared by CPU and are generated regardless of values in INTR_MASK register.

Bit	Name	Description
8	**ERROR**	Set by hardware when corresponding STATUS asserts, cleared by software.
7	**LOST_ARBITRATION**	Set by hardware when corresponding STATUS asserts, cleared by software.
6	**TIMEOUT**	Set by hardware when corresponding STATUS asserts, cleared by software.
5	**TX_HALF**	Set by hardware when corresponding STATUS asserts, cleared by software.
4	**TX_SPACE**	Set by hardware when corresponding STATUS asserts, cleared by software.
3	**TX_DONE**	Set by hardware when corresponding STATUS asserts, cleared by software.
2	**RX_HALF**	Set by hardware when corresponding STATUS asserts, cleared by software.
1	**RX_DATA**	Set by hardware when corresponding STATUS asserts, cleared by software.
0	**RX_DONE**	Set by hardware when corresponding STATUS asserts, cleared by software.

10.19.4 I2C_INTR_MASK

I²C Interrupt Mask Register

I2C_INTR_MASK			I²C Interrupt Mask Register				0xE000040C
b31	b30	b29	b28	b27	b26	b25	b24

I2C_INTR_MASK			I²C Interrupt Mask Register				
b23	b22	b21	b20	b19	b18	b17	b16

I2C_INTR_MASK				I²C Interrupt Mask Register			
b15	b14	b13	b12	b11	b10	b9	b8
							ERROR
							R/W
							R
							0

I2C_INTR_MASK				I²C Interrupt Mask Register			
b7	b6	b5	b4	b3	b2	b1	b0
LOST_ ARBITRATION	**TIMEOUT**	**TX_HALF**	**TX_SPACE**	**TX_DONE**	**RX_HALF**	**RX_DATA**	**RX_DONE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Interrupt mask bits. Determine if INTR bits are reported to CPU. Have no effect on creation of INTR bits.

Bit	Name	Description	
8	**ERROR**	1	Report interrupt to CPU
7	**LOST_ARBITRATION**	1	Report interrupt to CPU
6	**TIMEOUT**	1	Report interrupt to CPU
5	**TX_HALF**	1	Report interrupt to CPU
4	**TX_SPACE**	1	Report interrupt to CPU
3	**TX_DONE**	1	Report interrupt to CPU
2	**RX_HALF**	1	Report interrupt to CPU
1	**RX_DATA**	1	Report interrupt to CPU
0	**RX_DONE**	1	Report interrupt to CPU

10.19.5 I2C_TIMEOUT

Timeout Register

I2C_TIMEOUT			Timeout Register				0xE0000410
b31	b30	b29	b28	b27	b26	b25	b24
TIMEOUT[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

I2C_TIMEOUT			Timeout Register				
b23	b22	b21	b20	b19	b18	b17	b16
TIMEOUT[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

I2C_TIMEOUT			Timeout Register				
b15	b14	b13	b12	b11	b10	b9	b8
TIMEOUT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

I2C_TIMEOUT			Timeout Register				
b7	b6	b5	b4	b3	b2	b1	b0
TIMEOUT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFFFFFF							

Bus timeout interval. 0xFFFF_FFFFF means no timeout, otherwise, count the number of core clock cycles.

Bit	Name	Description
31:0	**TIMEOUT[31:0]**	Number of core clocks SCK can be held low by the slave byte transmission before triggering a time-out error.

10.19.6 I2C_DMA_TIMEOUT

DMA Timeout Register

I2C_DMA_TIMEOUT			DMA Timeout Register				0xE0000414
b31	b30	b29	b28	b27	b26	b25	b24

I2C_DMA_TIMEOUT			DMA Timeout Register				
b23	b22	b21	b20	b19	b18	b17	b16

I2C_DMA_TIMEOUT			DMA Timeout Register				
b15	b14	b13	b12	b11	b10	b9	b8
TIMEOUT16[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

I2C_DMA_TIMEOUT			DMA Timeout Register				
b7	b6	b5	b4	b3	b2	b1	b0
TIMEOUT16[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFF							

Bus timeout interval for DMA. 0xFFFF means no timeout, otherwise, count the number of core clock cycles of DMA not being ready before raising error.

Bit	Name	Description
15:0	**TIMEOUT16[15:0]**	Number of core clocks DMA has to be not ready before the condition is reported as error condition.

10.19.7 I2C_PREAMBLE_CTRL

I²C Preamble Control Register

I2C_PREAMBLE_CTRL			I²C Preamble Control Register				0xE0000418
b31	b30	b29	b28	b27	b26	b25	b24
I2C_PREAMBLE_CTRL			**I²C Preamble Control Register**				
b23	b22	b21	b20	b19	b18	b17	b16
I2C_PREAMBLE_CTRL			**I²C Preamble Control Register**				
b15	b14	b13	b12	b11	b10	b9	b8
STOP[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2C_PREAMBLE_CTRL			**I²C Preamble Control Register**				
b7	b6	b5	b4	b3	b2	b1	b0
START[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Indicates whether each preamble byte will be followed by a repeated start or stop/start sequence.

Bit	Name	Description
15:8	**STOP[7:0]**	If bit <x> is 1, issue a stop after byte <x> of the preamble (if both START and STOP are set, STOP will take priority).
7:0	**START[7:0]**	If bit <x> is 1, issue a start after byte <x> of the preamble.

10.19.8 I2C_PREAMBLE_DATA

I²C Preamble Data Register

I2C_PREAMBLE_DATA			I²C Preamble Data Register				0xE000041C
b63	b62	b61	b60	b59	b58	b57	b56
DATA[63:56]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2C_PREAMBLE_DATA			**I²C Preamble Data Register**				
b55	b54	b53	b52	b51	b50	b49	b48
DATA[55:48]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2C_PREAMBLE_DATA			**I²C Preamble Data Register**				
b47	b46	b45	b44	b43	b42	b41	b40
DATA[47:40]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2C_PREAMBLE_DATA			**I²C Preamble Data Register**				
b39	b38	b37	b36	b35	b34	b33	b32
DATA[39:32]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2C_PREAMBLE_DATA			**I²C Preamble Data Register**				
b31	b30	b29	b28	b27	b26	b25	b24
DATA[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2C_PREAMBLE_DATA			**I²C Preamble Data Register**				
b23	b22	b21	b20	b19	b18	b17	b16
DATA[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
I2C_PREAMBLE_DATA			**I²C Preamble Data Register**				
b15	b14	b13	b12	b11	b10	b9	b8
DATA[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

continued on next page

10.19.8 I2C_PREAMBLE_DATA *(continued)*

I2C_PREAMBLE_DATA			I²C Preamble Data Register				
b7	b6	b5	b4	b3	b2	b1	b0
DATA[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Contains the bytes that precede the data phase of the I2C command. This includes the slave address, R/W command, possible internal address and repeated start command. The number of bytes used is indicated in I2C_COMMAND.

Bit	Name	Description
63:0	**DATA[63:0]**	Command and initial data bytes.

10.19.9 I2C_PREAMBLE_RPT

I²C Preamble Repeat Register

I2C_PREAMBLE_RPT	I²C Preamble Repeat Register						0xE0000424
b31	b30	b29	b28	b27	b26	b25	b24
COUNT[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

I2C_PREAMBLE_RPT	I²C Preamble Repeat Register						
b23	b22	b21	b20	b19	b18	b17	b16
COUNT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

I2C_PREAMBLE_RPT	I²C Preamble Repeat Register						
b15	b14	b13	b12	b11	b10	b9	b8
COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0xFFFFFF							

I2C_PREAMBLE_RPT	I²C Preamble Repeat Register						
b7	b6	b5	b4	b3	b2	b1	b0
					STOP_ON_NACK	STOP_ON_ACK	RPT_ENABLE
					R/W	R/W	R/W
					R	R	R
					0	1	0

Offers the facility to reuse preamble in a programmable fashion. Useful for waiting on busy parts during time-consuming operations such as EEPROM programming. ERROR Code 9 or 10 when repeat feature exits. This register is only valid if RPT_ENABLE is true and takes precedence over other settings when valid.

Bit	Name		Description
31:8	COUNT[23:0]		Preamble stops after reaching the count or earlier if other conditions are satisfied. The maximum number of times a preamble can repeat is 0xFFFFFF.
2	STOP_ON_NACK	1	Preamble will stop repeating if the NACK is received after any byte. Unless this byte is set, continue on NACK if repeat is enabled.
1	STOP_ON_ACK	1	Preamble will stop repeating if the ACK is received after any byte.
0	RPT_ENABLE	1	Turns on preamble repeat feature. The sequence from IDLE to preamble_complete will repeat in a programmable fashion. START_FIRST will be honored. Repeating will always start at the first byte and end at the last byte. The only exception is if timeout is enabled and happens during byte transmission, in which case preamble will stop at the end of the current byte. Data phase is not entered if preamble repeat feature is enabled.

10.19.10 I2C_COMMAND

I²C Command Register

I2C_COMMAND			I²C Command Register				0xE0000428
b31	b30	b29	b28	b27	b26	b25	b24
I2C_STAT[1:0]			READ				
R	R		R/W				
W	W		R				
0	0		0				

I2C_COMMAND			I²C Command Register				
b23	b22	b21	b20	b19	b18	b17	b16

I2C_COMMAND			I²C Command Register				
b15	b14	b13	b12	b11	b10	b9	b8

I2C_COMMAND			I²C Command Register				
b7	b6	b5	b4	b3	b2	b1	b0
START_FIRST	STOP_LAST	NAK_LAST	PREAMBLE_VALID	PREAMBLE_LEN[3:0]			
R/W	R/W	R/W	R/W1S	R/W	R/W	R/W	R/W
R	R	R	R/W0C	R	R	R	R
1	0	1	0	0	0	0	0

This register initiates a read or write command on the I²C bus. If DMA_MODE=0 data is read/written one byte at a time from I2C_INGRESS_DATA/I2C_EGRESS_DATA. After all data bytes have been transferred, the command must be explicitly terminated by writing to this register, or I2C_BYTE_COUNT must be used to do so. If DMA_MODE=1 data is read/written from the relevant DMA sockets. In this case, the size of the DMA transfer in the socket determines the number of bytes transferred.

Bit	Name	Description	
31:30	**I2C_STAT[1:0]**	00	I²C is idle
		01	I²C is playing the preamble
		10	I²C is receiving data
		11	I²C is transmitting data.
28	**READ**	0	Write command (data phase)
		1	Read command (data phase)
		After command, the hardware will idle if no valid preamble exists, will play preamble if it does exist. Valid preamble is indicated by preamble valid bit. Data phase is not entered if preamble repeat feature is enabled.	

continued on next page

10.19.10 I2C_COMMAND *(continued)*

7	**START_FIRST**	0	Do nothing before the first byte of preamble
		1	Send START before the first byte of preamble
6	**STOP_LAST**	0	Send (repeated) START on last byte of data phase
		1	Send STOP on last byte of data phase
5	**NAK_LAST**	0	Send ACK on last byte of read
		1	Send NAK on last byte of read
4	**PREAMBLE_VALID**	Software sets this bit to indicate valid preamble. Hardware resets it to indicate that it has finished transmitting the preamble so that new preamble, such as one for doing restarts, can be populated.	
3:0	**PREAMBLE_LEN[3:0]**	Number of bytes in preamble. For preamble length = 1, set this to 1 etc. From 1 through 8. 0 and values > 8 are not supported.	

10.19.11 I2C_EGRESS_DATA

I²C Egress Data Register

I2C_EGRESS_DATA			I²C Egress Data Register				0xE000042C
b31	b30	b29	b28	b27	b26	b25	b24
I2C_EGRESS_DATA			**I²C Egress Data Register**				
b23	b22	b21	b20	b19	b18	b17	b16
I2C_EGRESS_DATA			**I²C Egress Data Register**				
b15	b14	b13	b12	b11	b10	b9	b8
I2C_EGRESS_DATA			**I²C Egress Data Register**				
b7	b6	b5	b4	b3	b2	b1	b0
DATA[7:0]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Accepts egress data one byte at a time. Writing to this register adds one byte to the FIFO if the FIFO has space available. It will result in ERROR if the FIFO is full. The size of the transmit FIFO is configured at design time.

Bit	Name	Description
7:0	**DATA[7:0]**	Data byte to be written to the peripheral in registered mode.

10.19.12 I2C_INGRESS_DATA

I²C Ingress Data Register

I2C_INGRESS_DATA			I²C Ingress Data Register				0xE0000430
b31	b30	b29	b28	b27	b26	b25	b24

I2C_INGRESS_DATA			I²C Ingress Data Register				
b23	b22	b21	b20	b19	b18	b17	b16

I2C_INGRESS_DATA			I²C Ingress Data Register				
b15	b14	b13	b12	b11	b10	b9	b8

I2C_INGRESS_DATA			I²C Ingress Data Register				
b7	b6	b5	b4	b3	b2	b1	b0
DATA[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Presents ingress data 1 byte at a time. Reading from this register removes one byte from the FIFO if the FIFO has data available. It will result in ERROR if the FIFO is empty. The size of the receive FIFO is configured at design time.

Bit	Name	Description
7:0	DATA[7:0]	Data byte read from the peripheral when DMA_MODE=0

10.19.13 I2C_CLOCK_LOW_COUNT

I²C Clock Low Count Register

I2C_CLOCK_LOW_COUNT			I²C Clock Low Count Register				0xE0000434
b31	b30	b29	b28	b27	b26	b25	b24
CLOCK_LOW_COUNT[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
I2C_CLOCK_LOW_COUNT			**I²C Clock Low Count Register**				
b23	b22	b21	b20	b19	b18	b17	b16
CLOCK_LOW_COUNT[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
I2C_CLOCK_LOW_COUNT			**I²C Clock Low Count Register**				
b15	b14	b13	b12	b11	b10	b9	b8
CLOCK_LOW_COUNT[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
I2C_CLOCK_LOW_COUNT			**I²C Clock Low Count Register**				
b7	b6	b5	b4	b3	b2	b1	b0
CLOCK_LOW_COUNT[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

This register is for hardware's internal use for clock synchronization in F/S mode. It appears here in the MMIO space for debug purposes.

Bit	Name	Description
31:0	**CLOCK_LOW_COUNT[31:0]**	Indicates the low period of the last clock pulse on the I²C bus, measured using the I²C core clock.

10.19.14 I2C_BYTE_COUNT

I²C Byte Count Register

I2C_BYTE_COUNT	I²C Byte Count Register						0xE0000438
b31	b30	b29	b28	b27	b26	b25	b24
BYTE_COUNT[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

I2C_BYTE_COUNT	I²C Byte Count Register						
b23	b22	b21	b20	b19	b18	b17	b16
BYTE_COUNT[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

I2C_BYTE_COUNT	I²C Byte Count Register						
b15	b14	b13	b12	b11	b10	b9	b8
BYTE_COUNT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

I2C_BYTE_COUNT	I²C Byte Count Register						
b7	b6	b5	b4	b3	b2	b1	b0
BYTE_COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFFFFFF							

Indicates number of bytes left to read/write in data phase of I²C command (excluding the preamble). Software specifies this number which does not change during the transfer. Data phase is not entered if preamble repeat feature is enabled.

Bit	Name	Description
31:0	BYTE_COUNT[31:0]	Number of bytes in the data phase of the transfer. Please perform transfers in terms of fixed byte count and perform dummy transactions at the end if required to complete the byte count.

10.19.15 I2C_BYTES_TRANSFERRED

I²C Byte Transfer Register

I2C_BYTES_TRANSFERRED			Number of Bytes Transferred in the Data Phase				0xE000043C
b31	b30	b29	b28	b27	b26	b25	b24
BYTE_COUNT[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
I2C_BYTES_TRANSFERRED			Number of Bytes Transferred in the Data Phase				
b23	b22	b21	b20	b19	b18	b17	b16
BYTE_COUNT[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
I2C_BYTES_TRANSFERRED			Number of Bytes Transferred in the Data Phase				
b15	b14	b13	b12	b11	b10	b9	b8
BYTE_COUNT[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
I2C_BYTES_TRANSFERRED			Number of Bytes Transferred in the Data Phase				
b7	b6	b5	b4	b3	b2	b1	b0
BYTE_COUNT[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Indicates number of bytes left to read/write in data phase of I²C command (excluding the preamble). When this counter reaches 0 during read, block will stop reading bytes and NAK the last byte.

Bit	Name	Description
31:0	BYTE_COUNT[31:0]	Indicates number of bytes transferred in the data phase (with ACK or without) so far. Does not include preamble bytes. Useful for determining when NACK happened in data transmission. If data NACK error happens on at the end of fifth byte, the value in this register will be 5. If timeout happens while transmitting the 1 st bit of the 11th byte, this value will be 11 and so on.

10.19.16 I2C_SOCKET

I²C Socket Register

I2C_SOCKET			I²C Socket Register				0xE0000440
b31	b30	b29	b28	b27	b26	b25	b24

I2C_SOCKET			I²C Socket Register				
b23	b22	b21	b20	b19	b18	b17	b16

I2C_SOCKET			I²C Socket Register				
b15	b14	b13	b12	b11	b10	b9	b8
INGRESS_SOCKET[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

I2C_SOCKET			I²C Socket Register				
b7	b6	b5	b4	b3	b2	b1	b0
EGRESS_SOCKET[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Indicates sockets used for DMA based operation.

Bit	Name	Description
15:8	INGRESS_SOCKET[7:0]	Socket number for ingress data 0-7 Supported 8-.. Reserved
7:0	EGRESS_SOCKET[7:0]	Socket number for egress data 0-7 Supported 8-.. Reserved

10.19.17 I2C_ID

Block Identification and Version Number Register

I2C_ID				Block Identification and Version Number			0xE00007F0
b31	b30	b29	b28	b27	b26	b25	b24
BLOCK_VERSION[15:8]							
R	R	R	R	R	R	R	R

I2C_ID				Block Identification and Version Number			
b23	b22	b21	b20	b19	b18	b17	b16
BLOCK_VERSION[7:0]							
R	R	R	R	R	R	R	R
0x0001							

I2C_ID				Block Identification and Version Number			
b15	b14	b13	b12	b11	b10	b9	b8
BLOCK_ID[15:8]							
R	R	R	R	R	R	R	R

I2C_ID				Block Identification and Version Number			
b7	b6	b5	b4	b3	b2	b1	b0
BLOCK_ID[7:0]							
R	R	R	R	R	R	R	R
0x0001							

Every low performance peripheral IP block will implement a few MMIO registers at a fixed offset in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31:16	**BLOCK_VERSION[15:0]**	Version number for the IP
15:0	**BLOCK_ID[15:0]**	A unique number identifying the IP in the memory space

10.19.18 I2C_POWER

Power, Clock, and Reset Control Register

I2C_POWER			Power, Clock, and Reset Control				0xE00007F4
b31	b30	b29	b28	b27	b26	b25	b24
RESETN							
R/W							
R							
0							

I2C_POWER			Power, Clock, and Reset Control				
b23	b22	b21	b20	b19	b18	b17	b16

I2C_POWER			Power, Clock, and Reset Control				
b15	b14	b13	b12	b11	b10	b9	b8

I2C_POWER			Power, Clock, and Reset Control				
b7	b6	b5	b4	b3	b2	b1	b0
							ACTIVE
							R
							W
							0

Every low performance peripheral IP block will implement a few MMIO registers at a fixed offset in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31	**RESETN**	Active LOW reset signal for all logic in the block. Note that reset is active on all flops in the block when either system reset is asserted (RESET# pin or SYSTEM_POWER.RESETN is asserted) or this signal is active. After setting this bit to 1, firmware will poll and wait for the ‘active’ bit to assert. Reading ‘1’ from ‘resetn’ does not indicate the block is out of reset – this may take some time depending on initialization tasks and clock frequencies. This bit must be asserted ('0') for at least 10 µs for effective reset.
0	**ACTIVE**	For blocks that must perform initialization after reset before becoming operational, this signal will remain deasserted until initialization is complete. In other words reading active=1 indicates block is initialized and ready for operation.

10.20 UART Registers

10.20.1 UART_CONFIG

UART Configuration and Mode Register

UART_CONFIG			UART Configuration and Mode Register				0xE0000800
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE	**TX_CLEAR**	**RX_CLEAR**					
R/W	R/W	R/W					
R	R	R					
0	0	0					

UART_CONFIG			UART Configuration and Mode Register				
b23	b22	b21	b20	b19	b18	b17	b16
				RX_POLL[3:0]			
				R/W	R/W	R/W	R/W
				R	R	R	R
				0	0	0	0

UART_CONFIG			UART Configuration and Mode Register				
b15	b14	b13	b12	b11	b10	b9	b8
TX_BREAK	**RX_FLOW_CTRL_ENBL**	**TX_FLOW_CTRL_ENBL**	**RTS**		**DMA_MODE**	**STOP_BITS[1:0]**	
R/W	R/W	R/W	R/W		R/W	R/W	R/W
R	R	R	R		R	R	R
0	0	0	1		0	0	0

UART_CONFIG			UART Configuration and Mode Register				
b7	b6	b5	b4	b3	b2	b1	b0
RX_STICKY_BIT	**TX_STICKY_BIT**	**PARITY_STICKY**	**PARITY_ODD**	**PARITY**	**LOOP_BACK**	**TX_ENABLE**	**RX_ENABLE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	1	0

Various modes of the UART block.

Bit	Name	Description
31	**ENABLE**	Enable block here, but only after all the configuration is set. Do not set this bit to 1 while changing any other value in this register. This bit will be synchronized to the core clock. Setting this bit to 0 will complete transmission of current sample. When DMA_MODE=1 any remaining samples in the pipeline are discarded. When DMA_MODE=0 no samples are lost.
30	**TX_CLEAR**	Use only when ENABLE=0; behavior undefined when ENABLE=1 0 Do nothing 1 Clear transmit FIFO (After TX_CLEAR is set, software must wait for TX_DONE before clearing it)
29	**RX_CLEAR**	0 Do nothing 1 Clear receive FIFO (Software must wait for RX_DATA=0 before clearing this bit again)

continued on next page

10.20.1 UART_CONFIG *(continued)*

19:16	**RX_POLL[3:0]**	Set timing when to sample for RX input: 0 Sample on bits 0,1,2 1 Sample on bits 1,2,3 2 Sample on bits 2,3,4 3 Sample on bits 3,4,5 4 Sample on bits 4,5,6 5 Sample on bits 5,6,7
15	**TX_BREAK**	0 Default Behavior 1 Wait for the currently transmitting byte to complete (including the stop bit) and then transmit 0's indefinitely until this bit is cleared. Do not transmit other bytes or discard any data while BREAK is being transmitted. CDT52575.
14	**RX_FLOW_CTRL_ENBL**	1 Enable hardware flow control for RX data
13	**TX_FLOW_CTRL_ENBL**	1 Enable hardware flow control for TX data
12	**RTS**	When RX hardware flow control is disabled, the software may use this bit to perform software flow control. 1 would signal the transmitter that we are ready to receive data. (Request to send). Modify only when ENABLE=0.
10	**DMA_MODE**	0 Register-based transfers 1 DMA-based transfers
9:8	**STOP_BITS[1:0]**	00 Reserved 01 1 Stop bit. 10 2 Stop bits 11 Reserved for 1.5 stop bits (not supported) **Note** STOP_BITS=2 is supported only when PARITY=0. Behavior is undefined otherwise.
7	**RX_STICKY_BIT**	If sticky parity is effective, log interrupt request if the received bit does not match this value.
6	**TX_STICKY_BIT**	Append this value at parity bit if sticky parity is effective.
5	**PARITY_STICKY**	Only relevant when PARITY=1 0 Computed parity 1 Sticky parity
4	**PARITY_ODD**	Only relevant when PARITY=1 and PARITY_STICKY=0 0 Even Parity 1 Odd parity
3	**PARITY**	0 No parity 1 Include parity bit
2	**LOOP_BACK**	0 No effect 1 Connect the value being transmitted to the receive buffer. Disable external transmit and receive.
1	**TX_ENABLE**	0 Transmitter disable, do not transmit data 1 Transmitter enabled
0	**RX_ENABLE**	0 Receiver disabled, ignore incoming data 1 Receive enabled

10.20.2 UART_STATUS

UART Status Register

UART_STATUS			UART Status Register				0xE0000804
b31	b30	b29	b28	b27	b26	b25	b24
			BUSY	ERROR_CODE[3:0]			
			R	R	R	R	R
			W	W	W	W	W
			0	0xF			

UART_STATUS			UART Status Register				
b23	b22	b21	b20	b19	b18	b17	b16

UART_STATUS				UART Status Register			
b15	b14	b13	b12	b11	b10	b9	b8
						ERROR	BREAK
						R/W1C	R
						R/W1S	W
						0	0

UART_STATUS				UART Status Register			
b7	b6	b5	b4	b3	b2	b1	b0
CTS_TOGGLE	CTS_STAT	TX_HALF	TX_SPACE	TX_DONE	RX_HALF	RX_DATA	RX_DONE
R/W1C	R	R	R	R	R	R	R
R/W1S	W	W	W	W	W	W	W
0	0	1	1	0	0	0	0

Status and error register. Most status bits are used to generate an interrupt on positive edge into INTR register.

Bit	Name	Description
28	**BUSY**	Indicates the block is busy transmitting data. This field may remain asserted after the block is suspended and must be polled before changing any configuration values.
27:24	**ERROR_CODE**	Error code, only relevant when ERROR=1. ERROR logs only the FIRST error to occur and will never change value as long as ERROR=1. 0 Missing Stop bit 1 RX Parity error (received parity bit does not match RX_STICKY_BIT in sticky parity mode, or the computed parity in non-sticky mode.) 12 Write to TX FIFO when FIFO full 13 Read from RX FIFO when FIFO empty 14 RX FIFO overflow or DMA Socket Overflow 15 No error
9	**ERROR**	A protocol error has occurred with cause ERROR_CODE. Must be cleared by software. Sticky
8	**BREAK**	Break condition is detected. Non sticky.
7	**CTS_TOGGLE**	Set when CTS toggles.

continued on next page

10.20.2 UART_STATUS *(continued)*

6	**CTS_STAT**	CTS Status, polarity inverted from the pin. (CTS pin 0 = CTS_STAT 1 means FX3 can transmit) Non sticky
5	**TX_HALF**	Indicates that the TX FIFO is at least half empty. This bit can be used to create burst-based interrupts. This bit is updated immediately after writes to EGRESS_DATA register. Non sticky.
4	**TX_SPACE**	Indicates space is available in the TX FIFO. This bit is updated immediately after writes to EGRESS_DATA register. Non sticky.
3	**TX_DONE**	Indicates no more data is available for transmission. Non sticky. If DMA_MODE=0 this is defined as TX FIFO empty and shift register empty. If DMA_MODE=1 this is defined as BYTE_COUNT=0 and shift register empty. Note that this field will only assert after a transmission was started - its power up state is 0.
2	**RX_HALF**	Indicates that the RX FIFO is at least half full (only relevant when DMA_MODE=0). This bit can be used to create burst based interrupts. This bit is updated immediately after reads from INGRESS_DATA register. Non sticky
1	**RX_DATA**	Indicates data is available in the RX FIFO (only relevant when DMA_MODE=0). This bit is updated immediately after reads from INGRESS_DATA register. Non sticky
0	**RX_DONE**	Indicates receive operation completed. Only relevant when DMA_MODE=1). Receive operation is complete when transfer size bytes in socket have been received. Non sticky. Does not need software intervention to clear it.

10.20.3 UART_INTR

UART Interrupt Request Register

UART_INTR			UART Interrupt Request Register				0xE0000808
b31	b30	b29	b28	b27	b26	b25	b24

UART_INTR			UART Interrupt Request Register				
b23	b22	b21	b20	b19	b18	b17	b16

UART_INTR			UART Interrupt Request Register				
b15	b14	b13	b12	b11	b10	b9	b8
						ERROR	**BREAK**
						R/W1C	R/W1C
						R/W1S	R/W1S
						0	0

UART_INTR			UART Interrupt Request Register				
b7	b6	b5	b4	b3	b2	b1	b0
CTS_TOGGLE	**CTS_STAT**	**TX_HALF**	**TX_SPACE**	**TX_DONE**	**RX_HALF**	**RX_DATA**	**RX_DONE**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
0	0	0	0	0	0	0	0

Interrupt requests. Derived from STATUS register. Interrupt requests must be cleared by CPU and are generated regardless of values in INTR_MASK register.

Bit	Name	Description
9	**ERROR**	Set by hardware when corresponding STATUS asserts, cleared by software.
8	**BREAK**	Set by hardware when corresponding STATUS asserts, cleared by software.
7	**CTS_TOGGLE**	Set by hardware when corresponding STATUS asserts, cleared by software.
6	**CTS_STAT**	Set by hardware when corresponding STATUS asserts, cleared by software.
5	**TX_HALF**	Set by hardware when corresponding STATUS asserts, cleared by software.
4	**TX_SPACE**	Set by hardware when corresponding STATUS asserts, cleared by software.
3	**TX_DONE**	Set by hardware when corresponding STATUS asserts, cleared by software.
2	**RX_HALF**	Set by hardware when corresponding STATUS asserts, cleared by software.
1	**RX_DATA**	Set by hardware when corresponding STATUS asserts, cleared by software.
0	**RX_DONE**	Set by hardware when corresponding STATUS asserts, cleared by software.

10.20.4 UART_INTR_MASK

UART Interrupt Mask Register

UART_INTR_MASK			UART Interrupt Mask Register				0xE000080C
b31	b30	b29	b28	b27	b26	b25	b24

UART_INTR_MASK			UART Interrupt Mask Register				
b23	b22	b21	b20	b19	b18	b17	b16

UART_INTR_MASK			UART Interrupt Mask Register				
b15	b14	b13	b12	b11	b10	b9	b8
						ERROR	**BREAK**
						R/W	R/W
						R	R
						0	0

UART_INTR_MASK			UART Interrupt Mask Register				
b7	b6	b5	b4	b3	b2	b1	b0
CTS_TOGGLE	**TIMEOUT**	**TX_HALF**	**TX_SPACE**	**TX_DONE**	**RX_HALF**	**RX_DATA**	**RX_DONE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Interrupt mask bits. Determine if INTR bits are reported to CPU. Have no effect on creation of INTR bits.

Bit	Name	Description	
9	**ERROR**	1	Report interrupt to CPU
8	**BREAK**	1	Report interrupt to CPU
7	**CTS_TOGGLE**	1	Report interrupt to CPU
6	**CTS_STAT**	1	Report interrupt to CPU
5	**TX_HALF**	1	Report interrupt to CPU
4	**TX_SPACE**	1	Report interrupt to CPU
3	**TX_DONE**	1	Report interrupt to CPU
2	**RX_HALF**	1	Report interrupt to CPU
1	**RX_DATA**	1	Report interrupt to CPU
0	**RX_DONE**	1	Report interrupt to CPU

10.20.5 UART_EGRESS_DATA

UART Egress Data Register

UART_EGRESS_DATA			UART Egress Data Register				0xE0000810
b31	b30	b29	b28	b27	b26	b25	b24
UART_EGRESS_DATA			**UART Egress Data Register**				
b23	b22	b21	b20	b19	b18	b17	b16
UART_EGRESS_DATA			**UART Egress Data Register**				
b15	b14	b13	b12	b11	b10	b9	b8
UART_EGRESS_DATA			**UART Egress Data Register**				
b7	b6	b5	b4	b3	b2	b1	b0
DATA[7:0]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Accepts egress data one byte at a time. Writing to this register adds one byte to the FIFO if the FIFO has space available. It will result in ERROR if the FIFO is full. The size of the transmit FIFO is configured at design time.

Bit	Name	Description
7:0	**DATA[7:0]**	Data byte to be written to the peripheral in registered mode.

10.20.6 UART_INGRESS_DATA

UART Ingress Data Register

UART_INGRESS_DATA			UART Ingress Data Register				0xE0000814
b31	b30	b29	b28	b27	b26	b25	b24

UART_INGRESS_DATA			UART Ingress Data Register				
b23	b22	b21	b20	b19	b18	b17	b16

UART_INGRESS_DATA			UART Ingress Data Register				
b15	b14	b13	b12	b11	b10	b9	b8

UART_INGRESS_DATA			UART Ingress Data Register				
b7	b6	b5	b4	b3	b2	b1	b0
DATA[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Presents ingress data 1 byte at a time. Reading from this register removes one byte from the FIFO if the FIFO has data available. It will result in ERROR if the FIFO is empty. The size of the receive FIFO is configured at design time.

Bit	Name	Description
7:0	**DATA[7:0]**	Data byte read from the peripheral when DMA_MODE=0

10.20.7 UART_SOCKET

UART Socket Register

UART_SOCKET			UART Socket Register				0xE0000818
b31	b30	b29	b28	b27	b26	b25	b24
UART_SOCKET			**UART Socket Register**				
b23	b22	b21	b20	b19	b18	b17	b16
UART_SOCKET			**UART Socket Register**				
b15	b14	b13	b12	b11	b10	b9	b8
INGRESS_SOCKET[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
UART_SOCKET			**UART Socket Register**				
b7	b6	b5	b4	b3	b2	b1	b0
EGRESS_SOCKET[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Indicates sockets used for DMA based operation.

Bit	Name	Description
15:8	**INGRESS_SOCKET[7:0]**	Socket number for ingress data 0-7 Supported 8-.. Reserved
7:0	**EGRESS_SOCKET[7:0]**	Socket number for egress data 0-7 Supported 8-.. Reserved

10.20.8 UART_RX_BYTE_COUNT

UART Receive Byte Count Register

UART_RX_BYTE_COUNT			UART Receive Byte Count Register				0xE000081C
b31	b30	b29	b28	b27	b26	b25	b24
BYTE_COUNT[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

UART_RX_BYTE_COUNT			UART Receive Byte Count Register				
b23	b22	b21	b20	b19	b18	b17	b16
BYTE_COUNT[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

UART_RX_BYTE_COUNT			UART Receive Byte Count Register				
b15	b14	b13	b12	b11	b10	b9	b8
BYTE_COUNT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

UART_RX_BYTE_COUNT			UART Receive Byte Count Register				
b7	b6	b5	b4	b3	b2	b1	b0
BYTE_COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFFFFFF							

Indicates the number of bytes to receive. This register is relevant only if DMA_MODE=1.

No end-of-packet (EOP) will be sent to the DMA adapter. If receive flow control is enabled it will be set to off. If flow control is not enabled and data is received and the socket remains active, it will be forwarded to the adapter regardless of BYTE_COUNT. BYTE_COUNT will stay 0 in this case (not decrement).

Bit	Name	Description
31:0	**BYTE_COUNT[31:0]**	Number of bytes left to receive 0xFFFFFFFF indicates infinite (counter will not decrement)

10.20.9 UART_TX_BYTE_COUNT

UART Transmit Byte Count Register

UART_TX_BYTE_COUNT			UART Transmit Byte Count Register				0xE0000820
b31	b30	b29	b28	b27	b26	b25	b24
BYTE_COUNT[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

UART_TX_BYTE_COUNT			UART Transmit Byte Count Register				
b23	b22	b21	b20	b19	b18	b17	b16
BYTE_COUNT[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

UART_TX_BYTE_COUNT			UART Transmit Byte Count Register				
b15	b14	b13	b12	b11	b10	b9	b8
BYTE_COUNT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R

UART_TX_BYTE_COUNT			UART Transmit Byte Count Register				
b7	b6	b5	b4	b3	b2	b1	b0
BYTE_COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFFFFFF							

Indicates the number of bytes to transmit. This register is relevant only if DMA_MODE=1.

On transmit a command will complete (TX_DONE) when BYTE_COUNT=0. No more bytes will be sent until BYTE_COUNT is modified again. Any EOP signalling from the DMA adapter will be ignored.

Bit	Name	Description
31:0	BYTE_COUNT[31:0]	Number of bytes left to transmit 0xFFFFFFFF indicates infinite (counter will not decrement)

10.20.10 UART_ID

Block Identification and Version Number Register

I2C_ID			Block Identification and Version Number				0xE0000BF0
b31	b30	b29	b28	b27	b26	b25	b24
BLOCK_VERSION[15:8]							
R	R	R	R	R	R	R	R

I2C_ID			Block Identification and Version Number				
b23	b22	b21	b20	b19	b18	b17	b16
BLOCK_VERSION[7:0]							
R	R	R	R	R	R	R	R
0x0001							

I2C_ID			Block Identification and Version Number				
b15	b14	b13	b12	b11	b10	b9	b8
BLOCK_ID[15:8]							
R	R	R	R	R	R	R	R

I2C_ID			Block Identification and Version Number				
b7	b6	b5	b4	b3	b2	b1	b0
BLOCK_ID[7:0]							
R	R	R	R	R	R	R	R
0x0002							

Every low performance peripheral IP block will implement a few MMIO registers at a fixed offset in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31:16	**BLOCK_VERSION[15:0]**	Version number for the IP
15:0	**BLOCK_ID[15:0]**	A unique number identifying the IP in the memory space

10.20.11 UART_POWER

Power, Clock, and Reset Control Register

UART_POWER			Power, Clock, and Reset Control				0xE0000BF4
b31	b30	b29	b28	b27	b26	b25	b24
RESETN							
R/W							
R							
0							

UART_POWER			Power, Clock, and Reset Control				
b23	b22	b21	b20	b19	b18	b17	b16

UART_POWER			Power, Clock, and Reset Control				
b15	b14	b13	b12	b11	b10	b9	b8

UART_POWER			Power, Clock, and Reset Control				
b7	b6	b5	b4	b3	b2	b1	b0
							ACTIVE
							R
							W
							0

Every low performance peripheral IP block will implement a few MMIO registers at a fixed offset in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31	**RESETN**	Active LOW reset signal for all logic in the block. Note that reset is active on all flops in the block when either system reset is asserted (RESET# pin or SYSTEM_POWER.RESETN is asserted) or this signal is active. After setting this bit to 1, firmware will poll and wait for the ‘active’ bit to assert. Reading ‘1’ from ‘resetn’ does not indicate the block is out of reset – this may take some time depending on initialization tasks and clock frequencies. This bit must be asserted ('0') for at least 10 µs for effective reset.
0	**ACTIVE**	For blocks that must perform initialization after reset before becoming operational, this signal will remain deasserted until initialization is complete. In other words reading active=1 indicates block is initialized and ready for operation.

10.21 SPI Registers

10.21.1 SPI_CONFIG

SPI Configuration and Modes Register

SPI_CONFIG			SPI Configuration and Modes Register				0xE0000C00
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE	**TX_CLEAR**	**RX_CLEAR**					
R/W	R/W	R/W					
R	R	R					
0	0	0					

SPI_CONFIG			SPI Configuration and Modes Register				
b23	b22	b21	b20	b19	b18	b17	b16
DESELECT	**WL[5:0]**						**SSPOL**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	8						0

SPI_CONFIG			SPI Configuration and Modes Register				
b15	b14	b13	b12	b11	b10	b9	b8
LAG[1:0]		**LEAD[1:0]**		**CHPA**	**CPOL**	**SSNCTRL[1:0]**	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1		1		0	0	1	

SPI_CONFIG		SPI Configuration and Modes Register					
b7	b6	b5	b4	b3	b2	b1	b0
		LOOPBACK	**SSN_BIT**	**ENDIAN**	**DMA_MODE**	**TX_ENABLE**	**RX_ENABLE**
		R/W	R/W	R/W	R/W	R/W	R/W
		R	R	R	R	R	R
		0	0	0	0	1	0

Various modes of the SPI block.

Bit	Name	Description
31	**ENABLE**	Enable block here, but only after all the configuration is set. Do not set this bit to 1 while changing any other value in this register. This bit will be synchronized to the core clock. Setting this bit to 0 will complete transmission of current sample. When DMA_MODE=1 any remaining samples in the pipeline are discarded. When DMA_MODE=0 no samples are lost.
30	**TX_CLEAR**	Use only when ENABLE=0; behavior undefined when ENABLE=1 0 Do nothing 1 Clear transmit FIFO (After TX_CLEAR is set, software must wait for TX_DONE before clearing it)
29	**RX_CLEAR**	Use only when ENABLE=0; behavior undefined when ENABLE=1 0 Do nothing 1 Clear receive FIFO (Software must wait for RX_DATA=0 before clearing this bit again)

continued on next page

10.21.1 SPI_CONFIG *(continued)*

23	**DESELECT**	0	Normal SSN behavior.
		1	Never assert SSN. Used to direct SPI traffic to non-default slaves whose SSN are connected to GPIO.
22:17	**WL[5:0]**	SPI transaction-unit length from 4 to 32 bits. 0-3 reserved.	
16	**SSPOL**	0	SSN is active low, else active high.
15:14	**LAG[1:0]**	SCK-SSN lag time. Indicates the number of half-clock cycles for which SSN needs to remain asserted after the last SCK cycle including zero. Note that LAG must be >0 when CHPA=1.	
13:12	**LEAD[1:0]**	SSN-SCK lead time.Encodings are: 0, 0.5, 1 or 1.5 SCK cycles. Indicates the number of half-clock cycles SSN need to assert ahead of the first SCK cycle. However zero lead is not supported.	
11	**CPHA**	Transaction start mode. See section 3.2.2.2	
10	**CPOL**	0	SCK idles low
		1	SCK idles high
9:8	**SSNCTRL[1:0]**	00	SSN is toggled by firmware.
		01	SSN remains asserted between each 8-bit transfer.
		10	SSN asserts high at the end of the transfer.
		11	SSN is governed by CPHA
5	**LOOPBACK**	0	No effect
		1	Send the value being transmitted to the receive buffer. Disable external transmit and receive.
4	**SSN_BIT**	This bit controls SSN behavior at the end of each received and transmitted “byte” if SSNCTRL indicate firmware SSN control. This bit is transmitted over SSN line *as is* and without regards to how many cycles it is going out. SSPOL is applied. The firmware is expected to send one word at a time in this mode. DESELECT bit takes precedence over this bit. Typically the firmware will keep this bit asserted for the entire transaction that can span multiple words. If SSN needs to be de-asserted between words, only single word transfers are possible when SSN is under firmware control. Modify only when ENABLE=0.	
3	**ENDIAN**	0	MSB First
		1	LSB First
2	**DMA_MODE**	0	Register-based transfers
		1	DMA-based transfers
1	**TX_ENABLE**	0	Transmitter disable, do not transmit data
		1	Transmitter enabled
0	**RX_ENABLE**	0	Receiver disabled, ignore incoming data
		1	Receive enabled

10.21.2 SPI_STATUS

SPI Status Register

SPI_STATUS			SPI Status Register				0xE0000C04
b31	b30	b29	b28	b27	b26	b25	b24
			BUSY	**ERROR_CODE[3:0]**			
			R	R	R	R	R
			W	W	W	W	W
			0	0xF			

SPI_STATUS			SPI Status Register				
b23	b22	b21	b20	b19	b18	b17	b16

SPI_STATUS			SPI Status Register				
b15	b14	b13	b12	b11	b10	b9	b8

SPI_STATUS			SPI Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
	ERROR	**TX_HALF**	**TX_SPACE**	**TX_DONE**	**RX_HALF**	**RX_DATA**	**RX_DONE**
	R/W1C	R	R	R	R	R	R
	R/W1S	W	W	W	W	W	W
	0	1	1	0	0	0	0

Status and error register. Most status bits are used to generate an interrupt on positive edge into INTR register.

Bit	Name	Description
28	**BUSY**	Indicates the block is busy transmitting data. This field may remain asserted after the block is suspended and must be polled before changing any configuration values.
27:24	**ERROR_CODE**	Error code, only relevant when ERROR=1. ERROR logs only the FIRST error to occur and will never change value as long as ERROR=1. 12 Write to TX FIFO when FIFO full 13 Read from RX FIFO when FIFO empty 15 No error
6	**ERROR**	An internal error has occurred with cause ERROR_CODE. Must be cleared by software. Sticky
5	**TX_HALF**	Indicates that the TX FIFO is at least half empty. This bit can be used to create burst-based interrupts. This bit is updated immediately after writes to EGRESS_DATA register. Non sticky.
4	**TX_SPACE**	Indicates space is available in the TX FIFO. This bit is updated immediately after writes to EGRESS_DATA register. Non sticky.

continued on next page

10.21.2 SPI_STATUS *(continued)*

3	**TX_DONE**	Indicates no more data is available for transmission. Non sticky. If DMA_MODE=0 this is defined as TX FIFO empty and shift register empty. If DMA_MODE=1 this is defined as BYTE_COUNT=0 and shift register empty. Note that this field will only assert after a transmission was started - its power up state is 0.
2	**RX_HALF**	Indicates that the RX FIFO is at least half full (only relevant when DMA_MODE=0). This bit can be used to create burst based interrupts. This bit is updated immediately after reads from INGRESS_DATA register. Non sticky
1	**RX_DATA**	Indicates data is available in the RX FIFO (only relevant when DMA_MODE=0). This bit is updated immediately after reads from INGRESS_DATA register. Non sticky
0	**RX_DONE**	Indicates receive operation completed. Only relevant when DMA_MODE=1). Receive operation is complete when transfer size bytes in socket have been received. Non sticky. Does not need software intervention to clear it.

10.21.3 SPI_INTR

SPI Interrupt Request Register

SPI_INTR	SPI Interrupt Request Register						0xE0000C08
b31	b30	b29	b28	b27	b26	b25	b24

SPI_INTR	SPI Interrupt Request Register						
b23	b22	b21	b20	b19	b18	b17	b16

SPI_INTR	SPI Interrupt Request Register						
b15	b14	b13	b12	b11	b10	b9	b8

SPI_INTR	SPI Interrupt Request Register						
b7	b6	b5	b4	b3	b2	b1	b0
	ERROR	**TX_HALF**	**TX_SPACE**	**TX_DONE**	**RX_HALF**	**RX_DATA**	**RX_DONE**
	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
	0	0	0	0	0	0	0

Interrupt requests. Derived from STATUS register. Interrupt requests must be cleared by CPU and are generated regardless of values in INTR_MASK register.

Bit	Name	Description
6	**ERROR**	Set by hardware when corresponding STATUS asserts, cleared by software.
5	**TX_HALF**	Set by hardware when corresponding STATUS asserts, cleared by software.
4	**TX_SPACE**	Set by hardware when corresponding STATUS asserts, cleared by software.
3	**TX_DONE**	Set by hardware when corresponding STATUS asserts, cleared by software.
2	**RX_HALF**	Set by hardware when corresponding STATUS asserts, cleared by software.
1	**RX_DATA**	Set by hardware when corresponding STATUS asserts, cleared by software.
0	**RX_DONE**	Set by hardware when corresponding STATUS asserts, cleared by software.

10.21.4 SPI_INTR_MASK

SPI Interrupt Mask Register

SPI_INTR_MASK			SPI Interrupt Mask Register				0xE0000C0C
b31	b30	b29	b28	b27	b26	b25	b24

SPI_INTR_MASK			SPI Interrupt Mask Register				
b23	b22	b21	b20	b19	b18	b17	b16

SPI_INTR_MASK			SPI Interrupt Mask Register				
b15	b14	b13	b12	b11	b10	b9	b8

SPI_INTR_MASK			SPI Interrupt Mask Register				
b7	b6	b5	b4	b3	b2	b1	b0
	ERROR	**TX_HALF**	**TX_SPACE**	**TX_DONE**	**RX_HALF**	**RX_DATA**	**RX_DONE**
	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	R	R	R	R	R	R	R
	0	0	0	0	0	0	0

Interrupt mask bits. Determine if INTR bits are reported to CPU. Have no effect on creation of INTR bits.

Bit	Name	Description	
6	**ERROR**	1	Report interrupt to CPU
5	**TX_HALF**	1	Report interrupt to CPU
4	**TX_SPACE**	1	Report interrupt to CPU
3	**TX_DONE**	1	Report interrupt to CPU
2	**RX_HALF**	1	Report interrupt to CPU
1	**RX_DATA**	1	Report interrupt to CPU
0	**RX_DONE**	1	Report interrupt to CPU

10.21.5 SPI_EGRESS_DATA

SPI Egress Data Register

SPI_EGRESS_DATA			SPI Egress Data Register				0xE0000C10
b31	b30	b29	b28	b27	b26	b25	b24
DATA32[31:24]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
SPI_EGRESS_DATA			SPI Egress Data Register				
b23	b22	b21	b20	b19	b18	b17	b16
DATA32[23:16]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
SPI_EGRESS_DATA			SPI Egress Data Register				
b15	b14	b13	b12	b11	b10	b9	b8
DATA32[15:8]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
SPI_EGRESS_DATA			SPI Egress Data Register				
b7	b6	b5	b4	b3	b2	b1	b0
DATA32[7:0]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Accepts egress data one word at a time, LSB justified. Writing to this register adds one word to the FIFO if the FIFO has space available. It will result in ERROR if the FIFO is full. The size of the transmit FIFO is configured at design time.

Bit	Name	Description
31:0	**DATA32[31:0]**	Data word to be written to the peripheral in registered mode. Only the least significant SPI_CONF.WL bits are used. Other bits are ignored.

10.21.6 SPI_INGRESS_DATA

SPI Ingress Data Register

SPI_INGRESS_DATA			SPI Ingress Data Register				0xE0000C14
b31	b30	b29	b28	b27	b26	b25	b24
DATA32[31:24]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
SPI_INGRESS_DATA			**SPI Ingress Data Register**				
b23	b22	b21	b20	b19	b18	b17	b16
DATA32[23:16]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
SPI_INGRESS_DATA			**SPI Ingress Data Register**				
b15	b14	b13	b12	b11	b10	b9	b8
DATA32[15:8]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
SPI_INGRESS_DATA			**SPI Ingress Data Register**				
b7	b6	b5	b4	b3	b2	b1	b0
DATA32[7:0]							
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Presents ingress data one word at a time, LSB justified. Reading from this register removes one word from the FIFO if the FIFO has data available. It will result in ERROR if the FIFO is empty. The size of the receive FIFO is configured at design time.

Bit	Name	Description
31:0	**DATA32[31:0]**	Data word read from the peripheral when DMA_MODE=0. Only the least significant SPI_CONF.WL bits are provided. Other bits are set to 0.

10.21.7 SPI_SOCKET

SPI Socket Register

SPI_SOCKET			SPI Socket Register				0xE0000C18
b31	b30	b29	b28	b27	b26	b25	b24

SPI_SOCKET			SPI Socket Register				
b23	b22	b21	b20	b19	b18	b17	b16

SPI_SOCKET			SPI Socket Register				
b15	b14	b13	b12	b11	b10	b9	b8
INGRESS_SOCKET[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

SPI_SOCKET			SPI Socket Register				
b7	b6	b5	b4	b3	b2	b1	b0
EGRESS_SOCKET[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Indicates sockets used for DMA based operation.

Bit	Name	Description
15:8	**INGRESS_SOCKET[7:0]**	Socket number for ingress data 0-7 Supported 8-.. Reserved
7:0	**EGRESS_SOCKET[7:0]**	Socket number for egress data 0-7 Supported 8-.. Reserved

10.21.8 SPI_RX_BYTE_COUNT

SPI Receive Byte Count Register

SPI_RX_BYTE_COUNT			SPI Receive Byte Count Register				0xE0000C1C
b31	b30	b29	b28	b27	b26	b25	b24
WORD_COUNT[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SPI_RX_BYTE_COUNT			SPI Receive Byte Count Register				
b23	b22	b21	b20	b19	b18	b17	b16
WORD_COUNT[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SPI_RX_BYTE_COUNT			SPI Receive Byte Count Register				
b15	b14	b13	b12	b11	b10	b9	b8
WORD_COUNT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SPI_RX_BYTE_COUNT			SPI Receive Byte Count Register				
b7	b6	b5	b4	b3	b2	b1	b0
WORD_COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0xFFFFFFFF							

Indicates the number of bytes to receive. This register is relevant only if DMA_MODE=1.

On receive BYTE_COUNT bytes will be received and forwarded to the DMA adapter. No EOP will be sent to the DMA adapter. If receive flow control is enabled it will be set to off. If flow control is not enabled and data is received and the socket remains active, it will be forwarded to the adapter regardless of BYTE_COUNT. BYTE_COUNT will stay 0 in this case (not decrement). Words can consist of 4-32 bits.

Bit	Name	Description
31:0	**WORD_COUNT[31:0]**	Number of words left to read or write 0xFFFFFFFF indicates infinite (counter will not decrement)

10.21.9 SPI_TX_BYTE_COUNT

SPI Transmit Byte Count Register

SPI_TX_BYTE_COUNT			SPI Transmit Byte Count Register				0xE0000C20
b31	b30	b29	b28	b27	b26	b25	b24
WORD_COUNT[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SPI_TX_BYTE_COUNT			SPI Transmit Byte Count Register				
b23	b22	b21	b20	b19	b18	b17	b16
WORD_COUNT[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SPI_TX_BYTE_COUNT			SPI Transmit Byte Count Register				
b15	b14	b13	b12	b11	b10	b9	b8
WORD_COUNT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

SPI_TX_BYTE_COUNT			SPI Transmit Byte Count Register				
b7	b6	b5	b4	b3	b2	b1	b0
WORD_COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0xFFFFFFFF							

Indicates the number of bytes to transmit. This register is relevant only if DMA_MODE=1.

On transmit a command will complete (TX_DONE) when BYTE_COUNT=0. No more bytes will be sent until BYTE_COUNT is modified again. Any EOP signalling from the DMA adapter will be ignored. Words can consist of 4-32 bits.

Bit	Name	Description
31:0	**WORD_COUNT[31:0]**	Number of words left to read or write 0xFFFFFFFF indicates infinite (counter will not decrement)

10.21.10 SPI_ID

Block Identification and Version Number Register

SPI_ID				Block Identification and Version Number			0xE0000FF0
b31	b30	b29	b28	b27	b26	b25	b24
BLOCK_VERSION[15:8]							
R	R	R	R	R	R	R	R

SPI_ID				Block Identification and Version Number			
b23	b22	b21	b20	b19	b18	b17	b16
BLOCK_VERSION[7:0]							
R	R	R	R	R	R	R	R
0x0001							

SPI_ID				Block Identification and Version Number			
b15	b14	b13	b12	b11	b10	b9	b8
BLOCK_ID[15:8]							
R	R	R	R	R	R	R	R

SPI_ID				Block Identification and Version Number			
b7	b6	b5	b4	b3	b2	b1	b0
BLOCK_ID[7:0]							
R	R	R	R	R	R	R	R
0x0003							

Every low performance peripheral IP block will implement a few MMIO registers at a fixed offset in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31:16	**BLOCK_VERSION[15:0]**	Version number for the IP
15:0	**BLOCK_ID[15:0]**	A unique number identifying the IP in the memory space

10.21.11 SPI_POWER

Power, Clock, and Reset Control Register

SPI_POWER			Power, Clock, and Reset Control				0xE0000FF4
b31	b30	b29	b28	b27	b26	b25	b24
RESETN							
R/W							
R							
0							

SPI_POWER			Power, Clock, and Reset Control				
b23	b22	b21	b20	b19	b18	b17	b16

SPI_POWER			Power, Clock, and Reset Control				
b15	b14	b13	b12	b11	b10	b9	b8

SPI_POWER			Power, Clock, and Reset Control				
b7	b6	b5	b4	b3	b2	b1	b0
							ACTIVE
							R
							W
							0

Every low performance peripheral IP block will implement a few MMIO registers at a fixed offset in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31	**RESETN**	Active LOW reset signal for all logic in the block. Note that reset is active on all flops in the block when either system reset is asserted (RESET# pin or SYSTEM_POWER.RESETN is asserted) or this signal is active. After setting this bit to 1, firmware will poll and wait for the ‘active’ bit to assert. Reading ‘1’ from ‘resetn’ does not indicate the block is out of reset – this may take some time depending on initialization tasks and clock frequencies. This bit must be asserted ('0') for at least 10 µs for effective reset.
0	**ACTIVE**	For blocks that must perform initialization after reset before becoming operational, this signal will remain deasserted until initialization is complete. In other words reading active=1 indicates block is initialized and ready for operation.

10.22 General Purpose IO Block Registers

10.22.1 GPIO_SIMPLE

Simple General Purpose IO Register (one pin)

There are 61 GPIO_SIMPLE registers. The address of each is calculated as GPIO_SIMPLE(x) = 0xE0001100 + (x*0x4). Hence GPIO_SIMPLE(0) is at address 0xE0001100, GPIO_SIMPLE(1) is at address 0xE0001100 + 0x4 and so on. The definition of each of these is the same.

GPIO_SIMPLE	Simple General Purpose IO Register (one pin)						0xE0001100
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE				**INTR**	**INTRMODE[2:0]**		
R/W				R/W1C	R/W	R/W	R/W
R				R/W1S	R	R	R
0				0	0	0	0

GPIO_SIMPLE	Simple General Purpose IO Register (one pin)						
b23	b22	b21	b20	b19	b18	b17	b16

GPIO_SIMPLE	Simple General Purpose IO Register (one pin)						
b15	b14	b13	b12	b11	b10	b9	b8

GPIO_SIMPLE	Simple General Purpose IO Register (one pin)						
b7	b6	b5	b4	b3	b2	b1	b0
	INPUT_EN	**DRIVE_HI_EN**	**DRIVE_LO_EN**			**IN_VALUE**	**OUT_VALUE**
	R/W	R/W	R/W			R	R/W
	R	R	R			W	R
	0	0	0			0	1

This register controls mode of operation and configuration for a single IO Pin. It also exposes status and read value.

Bit	Name	Description
31	**ENABLE**	Enable GPIO logic for this pin.
27	**INTR**	Registers edge triggered interrupt condition. Only relevant when INTRMODE=1,2,3,6,7. When INTRMODE=4,5 pin status is fed directly to interrupt controller; condition can be observed through IN_VALUE in this case.

continued on next page

10.22.1 GPIO_SIMPLE *(continued)*

26:24	**INTRMODE[2:0]**	Interrupt mode 0 No interrupt 1 Interrupt on positive edge 2 Interrupt on negative edge 3 Interrupt on any edge 4 Interrupt when pin is low 5 Interrupt when pin is high 6-7 Reserved
6	**INPUT_EN**	0 Input stage is disabled 1 Input stage is enabled, value is readable in IN_VALUE
5	**DRIVE_HI_EN**	Output driver enable when OUT_VALUE=1 0 Output driver is tristated 1 Output driver is active (weak/strong is determined in IO Matrix)
4	**DRIVE_LO_EN**	Output driver enable when OUT_VALUE=0 0 Output driver is tristated 1 Output driver is active (weak/strong is determined in IO Matrix)
1	**IN_VALUE**	Present input measurement 0 Low 1 High
0	**OUT_VALUE**	Output value used for output drive (if DRIVE_EN=1) 0 Driven Low 1 Driven High

10.22.2 GPIO_INVALUE0

GPIO Input Value Vector Register

GPIO_INVALUE0				GPIO Input Value Vector			0xE00013D0
b31	b30	b29	b28	b27	b26	b25	b24
INVALUE0[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
GPIO_INVALUE0				**GPIO Input Value Vector**			
b23	b22	b21	b20	b19	b18	b17	b16
INVALUE0[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
GPIO_INVALUE0				**GPIO Input Value Vector**			
b15	b14	b13	b12	b11	b10	b9	b8
INVALUE0[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
GPIO_INVALUE0				**GPIO Input Value Vector**			
b7	b6	b5	b4	b3	b2	b1	b0
INVALUE0[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

One bit for each GPIO pin indicating interrupt status

Bit	Name	Description
31:0	**INVALUE0[31:0]**	If bit <x> is set, IN_VALUE is active for GPIO <x>.

10.22.3 GPIO_INVALUE1

GPIO Input Value Vector Register

GPIO_INVALUE1			GPIO Input Value Vector				0xE00013D4
b31	b30	b29	b28	b27	b26	b25	b24
			INVALUE1[28:24]				
			R	R	R	R	R
			W	W	W	W	W
			0	0	0	0	0

GPIO_INVALUE1			GPIO Input Value Vector				
b23	b22	b21	b20	b19	b18	b17	b16
INVALUE1[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

GPIO_INVALUE1			GPIO Input Value Vector				
b15	b14	b13	b12	b11	b10	b9	b8
INVALUE1[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

GPIO_INVALUE1			GPIO Input Value Vector				
b7	b6	b5	b4	b3	b2	b1	b0
INVALUE1[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

One bit for each GPIO pin indicating interrupt status

Bit	Name	Description
28:0	**INVALUE1[28:0]**	If bit <x> is set, IN_VALUE is active for GPIO <x+32>.

10.22.4 GPIO_INTR0

GPIO Interrupt Vector Register

GPIO_INTR0			GPIO Interrupt Vector				0xE00013E0
b31	b30	b29	b28	b27	b26	b25	b24
INTR0[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
GPIO_INTR0			**GPIO Interrupt Vector**				
b23	b22	b21	b20	b19	b18	b17	b16
INTR0[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
GPIO_INTR0			**GPIO Interrupt Vector**				
b15	b14	b13	b12	b11	b10	b9	b8
INTR0[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
GPIO_INTR0			**GPIO Interrupt Vector**				
b7	b6	b5	b4	b3	b2	b1	b0
INTR0[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

One bit for each GPIO pin indicating interrupt status

Bit	Name	Description
31:0	**INTR0[31:0]**	If bit <x> is set, INTR is active for GPIO <x>.

10.22.5 GPIO_INTR1

GPIO Interrupt Vector Register

GPIO_INTR1			GPIO Interrupt Vector				0xE00013E4
b31	b30	b29	b28	b27	b26	b25	b24
			INTR1[28:24]				
			R	R	R	R	R
			W	W	W	W	W
			0	0	0	0	0

GPIO_INTR1			GPIO Interrupt Vector				
b23	b22	b21	b20	b19	b18	b17	b16
INTR1[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

GPIO_INTR1			GPIO Interrupt Vector				
b15	b14	b13	b12	b11	b10	b9	b8
INTR1[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

GPIO_INTR1			GPIO Interrupt Vector				
b7	b6	b5	b4	b3	b2	b1	b0
INTR1[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

One bit for each GPIO pin indicating interrupt status

Bit	Name	Description
28:0	**INTR1[28:0]**	If bit <x> is set, INTR is active for GPIO <x+32>.

10.22.6 GPIO_INTR

GPIO Interrupt Vector for PINs Register

GPIO_INTR			GPIO Interrupt Vector for PINs				0xE00013E8
b31	b30	b29	b28	b27	b26	b25	b24

GPIO_INTR			GPIO Interrupt Vector for PINs				
b23	b22	b21	b20	b19	b18	b17	b16

GPIO_INTR			GPIO Interrupt Vector for PINs				
b15	b14	b13	b12	b11	b10	b9	b8

GPIO_INTR			GPIO Interrupt Vector for PINs				
b7	b6	b5	b4	b3	b2	b1	b0
INTR7	**INTR6**	**INTR5**	**INTR4**	**INTR3**	**INTR2**	**INTR1**	**INTR0**
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

One bit for each GPIO PIN structure indicating its interrupt status. The actual pin associated with a PIN structure (if any) is set in GCTL_GPIO_COMPLEX.

Bit	Name	Description
7	**INTR7**	Set by hardware when corresponding STATUS asserts, cleared by software.
6	**INTR6**	Set by hardware when corresponding STATUS asserts, cleared by software.
5	**INTR5**	Set by hardware when corresponding STATUS asserts, cleared by software.
4	**INTR4**	Set by hardware when corresponding STATUS asserts, cleared by software.
3	**INTR3**	Set by hardware when corresponding STATUS asserts, cleared by software.
2	**INTR2**	Set by hardware when corresponding STATUS asserts, cleared by software.
1	**INTR1**	Set by hardware when corresponding STATUS asserts, cleared by software.
0	**INTR0**	Set by hardware when corresponding STATUS asserts, cleared by software.

10.22.7 GPIO_ID

Block Identification and Version Number Register

GPIO_ID			Block Identification and Version Number				0xE00013F0
b31	b30	b29	b28	b27	b26	b25	b24
BLOCK_VERSION[15:8]							
R	R	R	R	R	R	R	R

GPIO_ID			Block Identification and Version Number				
b23	b22	b21	b20	b19	b18	b17	b16
BLOCK_VERSION[7:0]							
R	R	R	R	R	R	R	R
0x0001							

GPIO_ID			Block Identification and Version Number				
b15	b14	b13	b12	b11	b10	b9	b8
BLOCK_ID[15:8]							
R	R	R	R	R	R	R	R

GPIO_ID			Block Identification and Version Number				
b7	b6	b5	b4	b3	b2	b1	b0
BLOCK_ID[7:0]							
R	R	R	R	R	R	R	R
0x0004							

Every low performance peripheral IP block will implement a few MMIO registers at a fixed offset in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31:16	**BLOCK_VERSION[15:0]**	Version number for the IP
15:0	**BLOCK_ID[15:0]**	A unique number identifying the IP in the memory space

10.22.8 GPIO_POWER

Power, Clock, and Reset Control Register

GPIO_POWER			Power, Clock, and Reset Control				0xE00013F4
b31	b30	b29	b28	b27	b26	b25	b24
RESETN							
R/W							
R							
0							

GPIO_POWER			Power, Clock, and Reset Control				
b23	b22	b21	b20	b19	b18	b17	b16

GPIO_POWER			Power, Clock, and Reset Control				
b15	b14	b13	b12	b11	b10	b9	b8

GPIO_POWER			Power, Clock, and Reset Control				
b7	b6	b5	b4	b3	b2	b1	b0
							ACTIVE
							R
							W
							0

Every low performance peripheral IP block will implement a few MMIO registers at a fixed offset in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31	**RESETN**	Active LOW reset signal for all logic in the block. Note that reset is active on all flops in the block when either system reset is asserted (RESET# pin or SYSTEM_POWER.RESETN is asserted) or this signal is active. After setting this bit to 1, firmware will poll and wait for the ‘active’ bit to assert. Reading ‘1’ from ‘resetn’ does not indicate the block is out of reset – this may take some time depending on initialization tasks and clock frequencies. This bit must be asserted ('0') for at least 10 µs for effective reset.
0	**ACTIVE**	For blocks that must perform initialization after reset before becoming operational, this signal will remain deasserted until initialization is complete. In other words reading active=1 indicates block is initialized and ready for operation.

10.23 General Purpose IO Registers (one pin)

10.23.1 PIN_STATUS

IO Pin Status Register (one pin)

PIN_STATUS	Configuration, mode and status of IO Pin						0xE0001000
b31	b30	b29	b28	b27	b26	b25	b24
ENABLE	**TIMER_MODE[2:0]**			**INTR**	**INTRMODE[2:0]**		
R/W	R/W	R/W	R/W	R/W1C	R/W	R/W	R/W
R	R	R	R	R/W1S	R	R	R
0	0	0	0	0	0	0	0

PIN_STATUS	Configuration, mode and status of IO Pin						
b23	b22	b21	b20	b19	b18	b17	b16

PIN_STATUS	Configuration, mode and status of IO Pin						
b15	b14	b13	b12	b11	b10	b9	b8
				MODE[3:0]			
				R/W	R/W	R/W	R/W
				R/W	R/W	R/W	R/W
				0	0	0	0

PIN_STATUS	Configuration, mode and status of IO Pin						
b7	b6	b5	b4	b3	b2	b1	b0
	INPUT_EN	**DRIVE_HI_EN**	**DRIVE_LO_EN**			**IN_VALUE**	**OUT_VALUE**
	R/W	R/W	R/W			R	R/W
	R	R	R			W	R/W
	0	0	0			0	1

This register controls mode of operation and configuration for a single IO Pin. It also exposes status and read value.

Bit	Name	Description
31	**ENABLE**	Enable GPIO logic for this pin.
30:28	**TIMER_MODE[2:0]**	0 Shutdown GPIO_TIMER 1 Use high frequency (e.g. 50MHz) 2 Use low frequency (e.g. 1MHz) 3 Use standby frequency (32KHz) 4 Use positive edge (sampled using high frequency) 5 Use negative edge (sampled using high frequency) 6 Use any edge (sampled using high frequency) 7 Reserved Note: Changing TIMER_MODE when ENABLE=1 will result in undefined behavior.
27	**INTR**	Registers edge triggered interrupt condition. Only relevant when INTRMODE=1,2,3,6,7. When INTRMODE=4,5 pin status is fed directly to interrupt controller; condition can be observed through IN_VALUE in this case.

continued on next page

10.23.1 GPIO_SIMPLE *(continued)*

26:24	**INTRMODE[2:0]**	Interrupt mode 0 No interrupt 1 Interrupt on positive edge on IN_VALUE 2 Interrupt on negative edge on IN_VALUE 3 Interrupt on any edge on IN_VALUE 4 Interrupt when IN_VALUE is low 5 Interrupt when IN_VALUE is high 6 Interrupt on TIMER = THRESHOLD 7 Interrupt on TIMER = 0
11:8	**MODE[3:0]**	Mode/command. Behavior is undefined when MODE>2 and TIMER_MODE=4,5,6. 0 STATIC 1 TOGGLE 2 SAMPLENOW 3 PULSENOW 4 PULSE 5 PWM 6 MEASURE_LOW 7 MEASURE_HIGH 8 MEASURE_LOW_ONCE 9 MEASURE_HIGH_ONCE 10 MEASURE_NEG 11 MEASURE_POS 12 MEASURE_ANY 13 MEASURE_NEG_ONCE 14 MEASURE_POS_ONCE 15 MEASURE_ANY_ONCE Note that when setting TOGGLE/SAMPLENOW/PULSENOW while ENABLE=0 the behavior is undefined)
6	**INPUT_EN**	0 Input stage is disabled 1 Input stage is enabled, value is readable in IN_VALUE
5	**DRIVE_HI_EN**	Output driver enable when OUT_VALUE=1 0 Output driver is tristated 1 Output driver is active (weak/strong is determined in IO Matrix)
4	**DRIVE_LO_EN**	Output driver enable when OUT_VALUE=0 0 Output driver is tristated 1 Output driver is active (weak/strong is determined in IO Matrix)
1	**IN_VALUE**	Present input measurement 0 Low 1 High
0	**OUT_VALUE**	Output value used for output drive (if DRIVE_EN=1) 0 Driven Low 1 Driven High

10.23.2 PIN_TIMER

IO Pin Timer/Counter Register

PIN_TIMER			Timer/counter for pulse and measurement modes				0xE0001004
b31	b30	b29	b28	b27	b26	b25	b24
TIMER[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
PIN_TIMER			**Timer/counter for pulse and measurement modes**				
b23	b22	b21	b20	b19	b18	b17	b16
TIMER[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
PIN_TIMER			**Timer/counter for pulse and measurement modes**				
b15	b14	b13	b12	b11	b10	b9	b8
TIMER[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
PIN_TIMER			**Timer/counter for pulse and measurement modes**				
b7	b6	b5	b4	b3	b2	b1	b0
TIMER[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

32-bit revolving counter, using period (GPIO_PERIOD+1).

Note that each GPIO has its own independent timer/counter

Bit	Name	Description
31:0	**TIMER[31:0]**	32-bit timer-counter value. Use MODE=SAMPLE_NOW (in PIN_STATUS register) to sample the timer into PIN_THRESHOLD. When TIMER reaches GPIO_PERIOD it resets to 0.

10.23.3 PIN_PERIOD

Pin Period Length Register

PIN_PERIOD	Period length for revolving counter GPIO_TIMER							0xE0001008
b31	b30	b29	b28	b27	b26	b25	b24
PERIOD[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
PIN_PERIOD	**Period length for revolving counter GPIO_TIMER**							
b23	b22	b21	b20	b19	b18	b17	b16
PERIOD[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
PIN_PERIOD	**Period length for revolving counter GPIO_TIMER**							
b15	b14	b13	b12	b11	b10	b9	b8
PERIOD[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
PIN_PERIOD	**Period length for revolving counter GPIO_TIMER**							
b7	b6	b5	b4	b3	b2	b1	b0
PERIOD[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0xFFFFFFFF							

Period of 32-bit revolving counter (actual period is PERIOD+1)

Note that each GPIO has its own independent timer/counter.

Note: In FX3 PIN_PERIOD must be 1 or greater.

Bit	Name	Description
31:0	**PERIOD[31:0]**	32-bit period for GPIO_TIMER (counter resets to 0 when PERIOD=TIMER)

10.23.4 PIN_THRESHOLD

Threshold or Measurement Register

PIN_THRESHOLD			Threshold or Measurement Register				0xE000100C
b31	b30	b29	b28	b27	b26	b25	b24
THRESHOLD[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PIN_THRESHOLD			**Threshold or Measurement Register**				
b23	b22	b21	b20	b19	b18	b17	b16
THRESHOLD[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PIN_THRESHOLD			**Threshold or Measurement Register**				
b15	b14	b13	b12	b11	b10	b9	b8
THRESHOLD[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
PIN_THRESHOLD			**Threshold or Measurement Register**				
b7	b6	b5	b4	b3	b2	b1	b0
THRESHOLD[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

A 32-bit threshold or measurement. Usage depends on MODE field in PIN_STATUS register.

Note that each GPIO has its own independent timer/counter.

Bit	Name	Description
31:0	**THRESHOLD[31:0]**	32-bit threshold or measurement for counter.

10.24 Low Performance Peripherals Registers

10.24.1 LPP_ID

Block Identification and Version Number Register

LPP_ID			Block Identification and Version Number				0xE0007F00
b31	b30	b29	b28	b27	b26	b25	b24
BLOCK_VERSION[15:8]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
LPP_ID			**Block Identification and Version Number**				
b23	b22	b21	b20	b19	b18	b17	b16
BLOCK_VERSION[7:0]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0x0001							
LPP_ID			**Block Identification and Version Number**				
b15	b14	b13	b12	b11	b10	b9	b8
BLOCK_ID[15:8]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
LPP_ID			**Block Identification and Version Number**				
b7	b6	b5	b4	b3	b2	b1	b0
BLOCK_ID[7:0]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0xFFFE							

Every IP block will implement a few MMIO registers at offset 0 in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31:16	**BLOCK_VERSION[15:0]**	Version number for the IP
15:0	**BLOCK_ID[15:0]**	A unique number identifying the IP in the memory space

10.24.2 LPP_POWER

Power, Clock, and Reset Control Register

LPP_POWER			Power, Clock, and Reset Control				0xE0007F04
b31	b30	b29	b28	b27	b26	b25	b24
RESETN							
R/W							
R							
0							

LPP_POWER			Power, Clock, and Reset Control				
b23	b22	b21	b20	b19	b18	b17	b16

LPP_POWER			Power, Clock, and Reset Control				
b15	b14	b13	b12	b11	b10	b9	b8

LPP_POWER			Power, Clock, and Reset Control				
b7	b6	b5	b4	b3	b2	b1	b0
							ACTIVE
							R
							W
							0

Every IP block will implement a few MMIO registers at offset 0 in its MMIO to space that identify the block and control its power/clock/reset state. These registers are located in the CPU/Interconnect power and clock domains and are accessible even when power/clock of the block is switched off.

Bit	Name	Description
31	**RESETN**	Active LOW reset signal for all logic in the block. Note that reset is active on all flops in the block when either system reset is asserted (RESET# pin or SYSTEM_POWER.RESETN is asserted) or this signal is active. After setting this bit to 1, firmware will poll and wait for the ‘active’ bit to assert. Reading ‘1’ from ‘resetn’ does not indicate the block is out of reset – this may take some time depending on initialization tasks and clock frequencies. This bit must be asserted ('0') for at least 10us for effective reset.
0	**ACTIVE**	For blocks that must perform initialization after reset before becoming operational, this signal will remain deasserted until initialization is complete. In other words reading active=1 indicates block is initialized and ready for operation.

10.25 DMA Socket and Descriptor Registers

Each functional block (LPP, PIB, UIB, and UIBIN) has its own set of DMA socket registers that use the offset address of that block. Table 10-3 shows the offset address for each block.

Table 10-3. Offset Addresses

Functional Block	Offset Address
LPP	0xE0008000
PIB	0xE0018000
UIB	0xE0038000
UIBIN	0xE0048000

Each functional block has more than one DMA socket and each DMA socket has the same set of DMA Socket registers. The offset of each DMA Socket register set is 0x80 (that is, the offset of DMA #0 is 0, DMA #1 is 0x80, DMA #2 is 0x100, and so on). The address of a DMA Socket register can be calculated as:

functional block address + DMA # * 0x80 + register address

For example, the address of the DMA_STATUS register of DMA #2 in the UIB functional block can be calculated as:

0xE0038000 + 2 (DMA #2) * 80 + 0x0C = 0xE0038000 + 0x100 + 0x0C = 0xE003810C

10.25.1 SCK_DSCR

Descriptor Chain Pointer Register

SCK_DSCR	Descriptor Chain Pointer							0x00
b31	b30	b29	b28	b27	b26	b25	b24	
DSCR_LOW[7:0]								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
R	R	R	R	R	R	R	R	
X	X	X	X	X	X	X	X	

SCK_DSCR	Descriptor Chain Pointer							
b23	b22	b21	b20	b19	b18	b17	b16	
DSCR_COUNT[7:0]								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
X	X	X	X	X	X	X	X	

SCK_DSCR	Descriptor Chain Pointer							
b15	b14	b13	b12	b11	b10	b9	b8	
DSCR_NUMBER[15:8]								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
X	X	X	X	X	X	X	X	

SCK_DSCR	Descriptor Chain Pointer							
b7	b6	b5	b4	b3	b2	b1	b0	
DSCR_NUMBER[7:0]								
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
X	X	X	X	X	X	X	X	

This register describes the current descriptor chain for this socket. Only write to this register when socket is in suspend state or disabled state.

Bit	Name	Description
31:24	**DSCR_LOW[31:24]**	The low watermark for dscr_count. When dscr_count is equal or less than dscr_low the status bit dscr_is_low is set and an interrupt can be generated (depending on int mask).
23:16	**DSCR_COUNT[7:0]**	Number of descriptors still left to process. This value is unrelated to actual number of descriptors in the list. It is used only to generate an interrupt to the CPU when the value goes low or zero (or both). When this value reaches 0 it will wrap around to 255. The socket will not suspend or be otherwise affected unless the descriptor chains ends with 0xFFFF descriptor number.
15:0	**DSCR_NUMBER[15:0]**	Descriptor number of currently active descriptor. A value of 0xFFFF designates no (more) active descriptors available. When activating a socket CPU will write number of first descriptor in here. Only modify this field when go_suspend=1 or go_enable=0

10.25.2 SCK_SIZE

Transfer Size Register

SCK_SIZE			Transfer Size Register				0x04
b31	b30	b29	b28	b27	b26	b25	b24
TRANS_SIZE[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
X	X	X	X	X	X	X	X
SCK_SIZE			**Transfer Size Register**				
b23	b22	b21	b20	b19	b18	b17	b16
TRANS_SIZE[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
X	X	X	X	X	X	X	X
SCK_SIZE			**Transfer Size Register**				
b15	b14	b13	b12	b11	b10	b9	b8
TRANS_SIZE[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
X	X	X	X	X	X	X	X
SCK_SIZE			**Transfer Size Register**				
b7	b6	b5	b4	b3	b2	b1	b0
TRANS_SIZE[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
X	X	X	X	X	X	X	X

Only write to this register when socket is in suspend state or disabled state.

Bit	Name	Description
31:0	**TRANS_SIZE[31:0]**	The number of bytes or buffers (depends on unit bit in SCK_STATUS) that are part of this transfer. A value of 0 signals an infinite/undetermined transaction size. Valid data bytes remaining in the last buffer beyond the transfer size will be read by socket and passed on to the core. Firmware must ensure that no additional bytes beyond the transfer size are present in the last buffer.

10.25.3 SCK_COUNT

Transfer Count Register

SCK_COUNT			Transfer Count Register				0x08
b31	b30	b29	b28	b27	b26	b25	b24
TRANS_COUNT[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X
SCK_COUNT			**Transfer Count Register**				
b23	b22	b21	b20	b19	b18	b17	b16
TRANS_COUNT[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X
SCK_COUNT			**Transfer Count Register**				
b15	b14	b13	b12	b11	b10	b9	b8
TRANS_COUNT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X
SCK_COUNT			**Transfer Count Register**				
b7	b6	b5	b4	b3	b2	b1	b0
TRANS_COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

Only write to this register when socket is in suspend state or disabled state.

Bit	Name	Description
31:0	TRANS_COUNT[31:0]	The number of bytes or buffers (depends on unit bit in SCK_STATUS) that have been transferred through this socket so far. If trans_size is >0 and trans_count >= trans_size the 'trans_done' bits in SCK_STATUS is both set. If SCK_STATUS.susp_trans=1 the socket is also suspended and the 'suspend' bit set. This count is updated only when a descriptor is completed and the socket proceeds to the next one. Exception: When socket suspends with PARTIAL_BUF=1, this value is (incorrectly) incremented by 1 (UNIT=1) or DSCR_SIZE.BYTE_COUNT (UNIT=0). Firmware must correct this before resuming the socket.

10.25.4 SCK_STATUS

Socket Status Register

SCK_STATUS			Socket Status Register				0x0C
b31	b30	b29	b28	b27	b26	b25	b24
GO_ENABLE	**GO_SUSPEND**	**UNIT**	**WRAPUP**	**SUSP_EOP**	**SUSP_TRANS**	**SUSP_LAST**	**SUSP_PARTIAL**
R/W	R/W	R/W	R/W1S	R/W	R/W	R/W	R/W
R	R	R	R/W0C	R	R	R	R
0	0	0	0	0	1	0	0

SCK_STATUS			Socket Status Register				
b23	b22	b21	b20	b19	b18	b17	b16
EN_CONS_ EVENTS	**EN_PROD_ EVENTS**	**TRUNCATE**	**ENABLED**	**SUSPENDED**	**ZLP_RCVD**	**STATE[2:1]**	
R/W	R/W	R/W	R	R	R	R	R
R	R	R	R/W	R/W	R/W	R/W	R/W
1	1	1	0	0	0	0	0

SCK_STATUS			Socket Status Register				
b15	b14	b13	b12	b11	b10	b9	b8
STATE[0]					**AVL_ENABLE**	**AVL_MIN[4:3]**	
R					R/W	R/W	R/W
R/W					R	R	R
0					0	0	0

SCK_STATUS			Socket Status Register				
b7	b6	b5	b4	b3	b2	b1	b0
AVL_MIN[2:0]			**AVL_COUNT[4:0]**				
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

This register contains the status and interrupt control bits. The interrupt request bits are independent of the interrupt mask bits. The mask bits affect which interrupt request bits are reported to CPU only.

Bit	Name	Description
31	**GO_ENABLE**	Indicates whether socket is enabled. When go_enable is cleared while socket is active, ongoing transfers are aborted after an unspecified amount of time. No update occurs from the descriptor registers back into memory. When go_enable is changed from 0 to 1, the socket will reload the active descriptor from memory regardless of the contents of DSCR_ registers. The socket will not wait for an EVENT to become active if the descriptor is available and ready for transfer (has space or data). The 'enabled' bit indicates whether the socket is actually enabled or not. This field lags go_enable by a short but unspecified time.
30	**GO_SUSPEND**	Directs a socket to go into suspend mode when the current descriptor completes. The main use of this bit is to safely append descriptors to an active socket without actually suspending it (in most cases). The process looks as follows: 1. GO_SUSPEND=1 2. Modify the chain in memory 3. Check if active descriptor is 0xFFFF or last in chain 4. If so, make corrections as necessary (complicated) 5. Clear any pending suspend interrupts (SCK_INTR[9:5]) 6. GO_SUSPEND=0 Note that the socket resumes only when SCK_INTR[9:5]=0 and GO_SUSPEND=0.

continued on next page

10.25.4 SCK_STATUS *(continued)*

29	**UNIT**	Indicates whether descriptors (1) or bytes (0) are counted by trans_count and trans_size. Descriptors are counting regardless of whether they contain any data or have eop set.
28	**WRAPUP**	Setting this bit will forcibly wrap-up a socket, whether it is out of data or not. This option is intended mainly for ingress sockets, but works also for egress sockets. Any remaining data in fetch buffers is ignored, in write buffers is flushed. Transaction and buffer counts are updated normally, and suspend behavior also happens normally (depending on various other settings in this register).G45
27	**SUSP_EOP**	When set, the socket will suspend after wrapping up the first buffer with dscr.eop=1. Note that this function will work the same for both ingress and egress sockets.
26	**SUSP_TRANS**	When set, the socket will suspend when trans_count >= trans_size. This equation is checked (and hence the socket will suspend) only at the boundary of buffers and packets (ie. buffer wrapup or EOP assertion).
25	**SUSP_LAST**	When set, the socket will suspend before activating a descriptor with TRANS_COUNT+BUFFER_SIZE>=TRANS_SIZE. This is relevant for both ingress and egress sockets.
24	**SUSP_PARTIAL**	When set, the socket will suspend before activating a descriptor with BYTE_COUNT<BUFFER_SIZE. This is relevant for egress sockets only.
23	**EN_CONS_EVENTS**	Enable (1) or disable (0) sending of consume events from any descriptor in this socket. If 0, events will be suppressed, and the descriptor will not be copied back into memory when completed.
22	**EN_PROD_EVENTS**	Enable (1) or disable (0) sending of produce events from any descriptor in this socket. If 0, events will be suppressed, and the descriptor will not be copied back into memory when completed.
21	**TRUNCATE**	Enable (1) or disable (0) truncating of BYTE_COUNT when TRANS_COUNT+BYTE_COUNT>=TRANS_SIZE. When enabled, ensures that an ingress transfer never contains more bytes then allowed. This function is needed to implement burst-based protocols that can only transmit full bursts of more than 1 byte.
20	**ENABLED**	Indicates the socket is currently enabled when asserted. After go_enable is changed, it may take some time for enabled to make the same change. This value can be polled to determine this fact.
19	**SUSPENDED**	Indicates the socket is currently in suspend state. In suspend mode there is no active descriptor; any previously active descriptor is wrapped up, copied back to memory and SCK_DSCR.dscr_number is updated using DSCR_CHAIN as needed. If the next descriptor is known (SCK_DSCR.dscr_number!=0xFFFF), this descriptor will be loaded after the socket resumes from suspend state. A socket can only be resumed by changing go_suspend from 1 to 0. If the socket is suspended while go_suspend=0, it must first be set and then again cleared.
18	**ZLP_RCVD**	Indicates the socket received a ZLP

continued on next page

10.25.4 SCK_STATUS *(continued)*

17:15 STATE[2:0] Internal operating state of the socket. This field is used for debugging and to safely modify active sockets (see go_suspend).

State	Default	Description
DESCR	9	Descriptor state. This is the default initial state indicating the descriptor registers are NOT valid in the Adapter. The Adapter will start loading the descriptor from memory if the socket becomes enabled and not suspended. Suspend has no effect on any other state.
STALL	1	Stall state. Socket is stalled waiting for data to be loaded into the Fetch Queue or waiting for an event.
ACTIVE	2	Active state. Socket is available for core data transfers.
EVENT	3	Event state. Core transfer is done. Descriptor is being written back to memory and an event is being generated if enabled.
CHECK1	4	Check states. An active socket gets here based on the core's EOP request to check the Transfer size and determine whether the buffer should be wrapped up. Depending on result, socket will either go back to Active state or move to the Event state.
SUSPENDED	5	Socket is suspended
CHECK2	6	Check states. An active socket gets here based on the core's EOP request to check the Transfer size and determine whether the buffer should be wrapped up. Depending on result, socket will either go back to Active state or move to the Event state.
WAITING	7	Waiting for confirmation that event was sent.

10 AVL_ENABLE Enables the functioning of AVL_COUNT and AVL_MIN. When 0, it will disable both stalling on AVL_MIN and generation of the sck_more_buf_avl signal described above.

9:5 AVL_MIN Minimum number of available buffers required by the adapter before activating a new one. This can be used to guarantee a minimum number of buffers available with old data to implement rollback. If AVL_ENABLE, the socket will remain in STALL state until AVL_COUNT>=AVL_MIN.

4:0 AVL_COUNT Number of available (free for ingress, occupied for egress) descriptors beyond the current one. This number is incremented by the adapter whenever an event is received on this socket and decremented whenever it activates a new descriptor. This value is used to create a signal to the IP Cores that indicates at least one buffer is available beyond the current one (sck_more_buf_avl).

10.25.5 SCK_INTR

Socket Interrupt Request Register

SCK_INTR	Socket Interrupt Request Register						0x10
b31	b30	b29	b28	b27	b26	b25	b24

SCK_INTR	Socket Interrupt Request Register						
b23	b22	b21	b20	b19	b18	b17	b16

SCK_INTR	Socket Interrupt Request Register						
b15	b14	b13	b12	b11	b10	b9	b8
						LAST_BUF	**PARTIAL_BUF**
						R/W1C	R/W1C
						W1S	W1S
						0	0

SCK_INTR	Socket Interrupt Request Register						
b7	b6	b5	b4	b3	b2	b1	b0
TRANS_DONE	**ERROR**	**SUSPEND**	**STALL**	**DSCR_NOT_AVL**	**DSCR_IS_LOW**	**CONSUME_ EVENT**	**PRODUCE_ EVENT**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
W1S	W1S	W1S	W1S	W1S	W1S	W1S	W1S
0	0	0	0	0	0	0	0

This register contains the interrupt request bits. The interrupt request bits are independent of the interrupt mask bits. The mask bits affect which interrupt request bits are reported to CPU only. If a socket is disabled, all pending interrupts are cleared automatically.

Bit	Name	Description
9	**LAST_BUF**	Indicates that the socket was suspended because of SUSP_LAST condition. Note that the socket resumes only when SCK_INTR[9:5]=0 and GO_SUSPEND=0.
8	**PARTIAL_BUF**	Indicates that the (egress) socket was suspended because of SUSP_PARTIAL condition. Note that the socket resumes only when SCK_INTR[9:5]=0 and GO_SUSPEND=0.
7	**TRANS_DONE**	Indicates that TRANS_COUNT has reached the limit TRANS_SIZE. This flag is only set when SUSP_TRANS=1. Note that because TRANS_COUNT is updated only at the granularity of entire buffers, it is possible that TRANS_COUNT exceeds TRANS_SIZE before the socket suspends. Software must detect and deal with these situations. When asserting EOP to the adapter on ingress, the trans_count is not updated unless the socket actually suspends (see SUSP_TRANS). Note that the socket resumes only when SCK_INTR[9:5]=0 and GO_SUSPEND=0.

continued on next page

10.25.5 SCK_INTR *(continued)*

6	**ERROR**	Indicates the socket is suspended because of an error condition (internal to the adapter) – if error=1 then suspend=1 as well. Possible error causes are: - dscr_size.buffer_error bit already set in the descriptor. - dscr_size.byte_count > dscr_size.buffer_size - core writes into an inactive socket. - core did not consume all the data in the buffer. Note that the socket resumes only when SCK_INTR[9:5]=0 and GO_SUSPEND=0.
5	**SUSPEND**	Indicates the socket has gone into suspend mode. This may be caused by any hardware initiated condition (e.g. DSCR_NOT_AVL, any of the SUSP_*) or by setting GO_SUSPEND=1. Note that this bit will remain asserted until cleared by software, regardless of whether the suspend condition is resolved. Note that the socket resumes only when SCK_INTR[9:5]=0 and GO_SUSPEND=0.
4	**STALL**	Indicates the socket has stalled, waiting for an event signaling its descriptor has become available. Note that this bit will remain asserted until cleared by software, regardless of whether the socket resumes.
3	**DSCR_NOT_AVL**	Indicates the no descriptor is available. Not available means that the current descriptor number is 0xFFFF. Note that this bit will remain asserted until cleared by software, regardless of whether a new descriptor number is loaded.
2	**DSCR_IS_LOW**	Indicates that dscr_count has fallen below its watermark dscr_low. If dscr_count wraps around to 255 dscr_is_low will remain asserted until cleared by software
1	**CONSUME_EVENT**	Indicates that a consume event is received or transmitted since last cleared.
0	**PRODUCE_EVENT**	Indicates that a produce event is received or transmitted since last cleared.

10.25.6 SCK_INTR_MASK

Socket Interrupt Mask Register

SCK_INTR_MASK			Socket Interrupt Mask Register				0x14
b31	b30	b29	b28	b27	b26	b25	b24

SCK_INTR_MASK			Socket Interrupt Mask Register				
b23	b22	b21	b20	b19	b18	b17	b16

SCK_INTR_MASK			Socket Interrupt Mask Register				
b15	b14	b13	b12	b11	b10	b9	b8
						LAST_BUF	**PARTIAL_BUF**
						R/W1C	R/W1C
						W1S	W1S
						0	0

SCK_INTR_MASK			Socket Interrupt Mask Register				
b7	b6	b5	b4	b3	b2	b1	b0
TRANS_DONE	**ERROR**	**SUSPEND**	**STALL**	**DSCR_NOT_AVL**	**DSCR_IS_LOW**	**CONSUME_ EVENT**	**PRODUCE_ EVENT**
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
W1S	W1S	W1S	W1S	W1S	W1S	W1S	W1S
0	0	0	0	0	0	0	0

The interrupt request bits in SCK_INTR function independent of the interrupt mask bits. The mask bits affect which interrupt request bits are reported to CPU only.

Bit	Name	Description
9	**LAST_BUF**	1: Report interrupt to CPU
8	**PARTIAL_BUF**	1: Report interrupt to CPU
7	**TRANS_DONE**	1: Report interrupt to CPU
6	**ERROR**	1: Report interrupt to CPU
5	**SUSPEND**	1: Report interrupt to CPU
4	**STALL**	1: Report interrupt to CPU
3	**DSCR_NOT_AVL**	1: Report interrupt to CPU
2	**DSCR_IS_LOW**	1: Report interrupt to CPU

continued on next page

10.25.6 SCK_INTR_MASK *(continued)*

1	**CONSUME_EVENT**	1: Report interrupt to CPU
0	**PRODUCE_EVENT**	1: Report interrupt to CPU

10.25.7 DSCR_BUFFER

Descriptor Buffer Base Address Register

DSCR_BUFFER			Descriptor Buffer Base Address Register				0x20
b31	b30	b29	b28	b27	b26	b25	b24
BUFFER_ADDR[31:24]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X
SCK_COUNT			**Transfer Count Register**				
b23	b22	b21	b20	b19	b18	b17	b16
BUFFER_ADDR[23:16]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X
SCK_COUNT			**Transfer Count Register**				
b15	b14	b13	b12	b11	b10	b9	b8
BUFFER_ADDR[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X
SCK_COUNT			**Transfer Count Register**				
b7	b6	b5	b4	b3	b2	b1	b0
BUFFER_ADDR[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

This register is only modified by the adapter when being read from memory. This register is not intended to be modified by software directly.

Bit	Name	Description
31:0	BUFFER_ADDR[31:0]	The base address of the buffer where data is written. This address is not necessarily word-aligned to allow for header/trailer/length modification.

10.25.8 DSCR_SYNC

Descriptor Synchronization Pointers Register

DSCR_SYNC			Descriptor Synchronization Pointers Register				0x24
b31	b30	b29	b28	b27	b26	b25	b24
EN_PROD_INT	EN_PROD_EVENT	PROD_IP[5:0]					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

DSCR_SYNC			Descriptor Synchronization Pointers Register				
b23	b22	b21	b20	b19	b18	b17	b16
PROD_SCK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

DSCR_SYNC			Descriptor Synchronization Pointers Register				
b15	b14	b13	b12	b11	b10	b9	b8
EN_CONS_INT	EN_CONS_EVENT	CONS_IP[5:0]					
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

DSCR_SYNC			Descriptor Synchronization Pointers Register				
b7	b6	b5	b4	b3	b2	b1	b0
CONS_SCK[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

This register is only modified by the adapter when being read from memory. This register is not intended to be modified by software directly.

Bit	Name	Description
31	**EN_PROD_INT**	Enable generation of a produce event interrupt for this descriptor only. This interrupt will only be seen by the CPU if SCK_STATUS. int_mask has this interrupt enabled as well. Note that this flag influences the logging of the interrupt in SCK_STATUS; it has no effect on the reporting of the interrupt to the CPU similar to the SCK_STATUS.int_mask.
30	**EN_PROD_EVENT**	Enable sending of a produce events from this descriptor only. Events are sent only if SCK_STATUS.en_produce_ev=1. If 0, events will be suppressed, and the descriptor will not be copied back into memory when completed.
29:24	**PROD_IP[5:0]**	The IP number of the producing socket to which the consume event will be sent. Use 0x3F to designate ARM CPU (so software) as producer; the event will be lost in this case and an interrupt should also be generated to observe this condition.
23:16	**PROD_SCK[7:0]**	The socket number of the producing socket to which the consume event will be sent. If prod_ip and prod_sck matches the socket's IP and socket number then the matching socket becomes consuming socket.

continued on next page

10.25.8 DSCR_SYNC *(continued)*

15	**EN_CONS_INT**	Enable generation of a consume event interrupt for this descriptor only. This interrupt will only be seen by the CPU if SCK_STATUS.int_mask has this interrupt enabled as well. Note that this flag influences the logging of the interrupt in SCK_STATUS; it has no effect on the reporting of the interrupt to the CPU similar to the SCK_STATUS.int_mask.
14	**EN_CONS_EVENT**	Enable sending of consume events from this descriptor only. Events are sent only if SCK_STATUS.en_consume_ev=1. When events are disabled, the adapter also does not update the descriptor in memory to clear its occupied bit.
13:8	**CONS_IP[5:0]**	The IP number of the consuming socket to which the produce event will be sent. Use 0x3F to designate ARM CPU (so software) as consumer; the event will be lost in this case and an interrupt should also be generated to observe this condition.
7:0	**CONS_SCK[7:0]**	The socket number of the consuming socket to which the produce event will be sent. If cons_ip and cons_sck matches the socket's IP and socket number then the matching socket becomes consuming socket.

10.25.9 DSCR_CHAIN

Descriptor Chain Pointers Register

DSCR_CHAIN			Descriptor Chain Pointers Register				0x28
b31	b30	b29	b28	b27	b26	b25	b24
WR_NEXT_DSCR[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X
DSCR_CHAIN			**Descriptor Chain Pointers Register**				
b23	b22	b21	b20	b19	b18	b17	b16
WR_NEXT_DSCR[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X
DSCR_CHAIN			**Descriptor Chain Pointers Register**				
b15	b14	b13	b12	b11	b10	b9	b8
RD_NEXT_DSCR[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X
DSCR_CHAIN			**Descriptor Chain Pointers Register**				
b7	b6	b5	b4	b3	b2	b1	b0
RD_NEXT_DSCR[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

This register is only modified by the adapter when being read from memory. Only write to this register when socket is in suspend state or disabled state.

Bit	Name	Description
31:16	**WR_NEXT_DSCR[15:0]**	Descriptor number of the next task for producer. A value of 0xFFFF signals end of this list.
15:0	**RD_NEXT_DSCR[15:0]**	Descriptor number of the next task for consumer. A value of 0xFFFF signals end of this list.

10.25.10 DSCR_SIZE

Descriptor Size Register

DSCR_SIZE			Descriptor Size Register				0x02C
b31	b30	b29	b28	b27	b26	b25	b24
BYTE_COUNT[15:8]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

DSCR_SIZE			Descriptor Size Register				
b23	b22	b21	b20	b19	b18	b17	b16
BYTE_COUNT[7:0]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

DSCR_SIZE			Descriptor Size Register				
b15	b14	b13	b12	b11	b10	b9	b8
BUFFER_SIZE[11:4]							
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

DSCR_SIZE			Descriptor Size Register				
b7	b6	b5	b4	b3	b2	b1	b0
BUFFER_SIZE[3:0]				BUFFER_ OCCUPIED	BUFFER_ERROR	EOP	MARKER
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

The error and start/end of packet markers are provided by the IP through hardware signals when streaming data into the socket. These signal are optional and are used only for IP that can produce packets larger than on buffer and/or IP that can detect errors and produce meaningful packets in these cases.

Only eop,byte_count,error,buffer_occupied are modified by the adapter after the initial read from memory. This is the only register that is written back into memory by the adapter. This register is not intended to be modified by software directly.

Bit	Name	Description
31:16	**BYTE_COUNT[15:0]**	The number of data bytes present in the buffer. An occupied buffer is not always full, in particular when variable length packets are transferred.
15:4	**BUFFER_SIZE[11:0]**	The size of the buffer in multiples of 16 bytes
3	**BUFFER_OCCUPIED**	Indicates the buffer is in use (1) or empty (0). A consumer will interpret this as: 0 Buffer is empty, wait until filled. 1 Buffer has data that can be consumed A producer will interpret this as: 0 Buffer is ready to be filled 1 Buffer is occupied, wait until empty
2	**BUFFER_ERROR**	Indicates the buffer data is valid (0) or in error (1).

continued on next page

10.25.10 DSCR_SIZE *(continued)*

1	**EOP**	A marker indicating this descriptor refers to the last buffer of a packet or transfer. Packets/transfers may span more than one buffer. The producing IP provides this marker by providing the EOP signal to its DMA adapter. The consuming IP observes this marker by inspecting its EOP return signal from its own DMA adapter. For more information, refer to the section on packets, buffers and transfers in the FX3 DMA Subsystem chapter on page 61.
0	**MARKER**	A marker that is provided by software and can be observed by the IP. Its meaning is defined by the IP that uses it. This bit has no effect on the other DMA mechanisms.

10.25.11 EVENT

Event Communication Register

EVENT	Event Communication Register						0x7C
b31	b30	b29	b28	b27	b26	b25	b24

EVENT	Event Communication Register						
b23	b22	b21	b20	b19	b18	b17	b16
							EVENT_TYPE
							W
							R/W
							0

EVENT	Event Communication Register						
b15	b14	b13	b12	b11	b10	b9	b8
ACTIVE_DSCR[15:8]							
W	W	W	W	W	W	W	W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

EVENT	Event Communication Register						
b7	b6	b5	b4	b3	b2	b1	b0
ACTIVE_DSCR[7:0]							
W	W	W	W	W	W	W	W
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

This register is located in the top of the MMIO map for each socket in each IP block. This register is write-only and cannot be read from (it represents an action that impacts the descriptor registers, not a state). Writing to this register signals a produce/consume event (depending on whether the socket is consuming or producing).

A read to this location will return undefined data.

Bit	Name	Description
16	EVENT_TYPE	Type of event 0 Consume event descriptor is marked empty - BUFFER_OCCUPIED=0) 1 Produce event descriptor is marked full = BUFFER_OCCUPIED=1)
15:0	ACTIVE_DSCR[15:0]	The active descriptor number for which the event is sent.

10.26 DMA Adapter Global Registers

Each functional block (LPP, PIB, UIB, and UIBIN) has its own set of DMA registers that use the offset address of that block. Table 10-4 shows the offset address for each block.

Table 10-4. Offset Addresses

Functional Block	Offset Address
LPP	0xE000FF00
PIB	0xE001FF00
UIB	0xE003FF00
UIBIN	0xE00FF000

Each functional block has more than one DMA and each DMA has the same set of DMA Socket registers. The offset of each DMA Socket register set is 0x80 (that is, the offset of DMA #0 is 0, DMA #1 is 0x80, DMA #2 is 0x100, and so on). The address of a DMA Socket register can be calculated as:

functional block address + DMA # * 0x80 + register address

For example, the address of the ADAPTER_DEBUG register of DMA #2 in the UIB functional block can be calculated as:

0xE003FF00 + 2 (DMA #2) * 80 + 0xF4 = 0xE003FF00 + 0x100 + 0xF4 = 0xE00400F4.

10.26.1 SCK_INTR

Socket Interrupt Request Register

SCK_INTR	Socket Interrupt Request Register						0x00
b255	b254	b253	b252	b251	b250	b249	b248
SCKINTR[255:248]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b247	b246	b245	b244	b243	b243	b241	b240
SCKINTR[247:240]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b239	b238	b237	b236	b235	b234	b233	b232
SCKINTR[239:232]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b231	b230	b229	b228	b227	b226	b225	b224
SCKINTR[231:224]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

continued on next page

10.26.1 SCK_INTR *(continued)*

SCK_INTR	Socket Interrupt Request Register						
b223	b222	b221	b220	b219	b218	b217	b216
SCKINTR[223:216]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b215	b214	b213	b212	b211	b210	b209	b208
SCKINTR[215:208]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b207	b206	b205	b204	b203	b202	b201	b200
SCKINTR[207:200]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b199	b198	b197	b196	b195	b194	b193	b192
SCKINTR[199:192]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b191	b190	b189	b188	b187	b186	b185	b184
SCKINTR[191:184]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b183	b182	b181	b180	b179	b178	b177	b176
SCKINTR[183:176]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b175	b174	b173	b172	b171	b170	b169	b168
SCKINTR[175:168]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

continued on next page

10.26.1 SCK_INTR *(continued)*

SCK_INTR				Socket Interrupt Request Register			
b167	b166	b165	b164	b163	b162	b161	b160
SCKINTR[167:160]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR				Socket Interrupt Request Register			
b159	b158	b157	b156	b155	b154	b153	b152
SCKINTR[159:152]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR				Socket Interrupt Request Register			
b151	b150	b149	b148	b147	b146	b145	b144
SCKINTR[151:144]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR				Socket Interrupt Request Register			
b143	b142	b141	b140	b139	b138	b137	b136
SCKINTR[143:136]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR				Socket Interrupt Request Register			
b135	b134	b133	b132	b131	b130	b129	b128
SCKINTR[135:128]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR				Socket Interrupt Request Register			
b127	b126	b125	b124	b123	b122	b121	b120
SCKINTR[127:120]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR				Socket Interrupt Request Register			
b119	b118	b117	b116	b115	b114	b113	b112
SCKINTR[119:112]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

continued on next page

10.26.1 SCK_INTR *(continued)*

SCK_INTR	Socket Interrupt Request Register						
b111	b110	b109	b108	b107	b106	b105	b104
SCKINTR[111:104]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b103	b102	b101	b100	b99	b98	b97	b96
SCKINTR[103:96]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b95	b94	b93	b92	b91	b90	b89	b88
SCKINTR[95:88]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b87	b86	b85	b84	b83	b82	b81	b80
SCKINTR[87:80]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b79	b78	b77	b76	b75	b74	b73	b72
SCKINTR[79:72]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b71	b70	b69	b68	b67	b66	b65	b64
SCKINTR[71:64]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR	Socket Interrupt Request Register						
b63	b62	b61	b60	b59	b58	b57	b56
SCKINTR[63:56]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

continued on next page

10.26.1 SCK_INTR *(continued)*

SCK_INTR				Socket Interrupt Request Register			
b55	b54	b53	b52	b51	b50	b49	b48
SCKINTR[55:48]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR				Socket Interrupt Request Register			
b47	b46	b45	b44	b43	b42	b41	b40
SCKINTR[47:40]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR				Socket Interrupt Request Register			
b39	b38	b37	b36	b35	b34	b33	b32
SCKINTR[39:32]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR				Socket Interrupt Request Register			
b31	b30	b29	b28	b27	b26	b25	b24
SCKINTR[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR				Socket Interrupt Request Register			
b23	b22	b21	b20	b19	b18	b17	b16
SCKINTR[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR				Socket Interrupt Request Register			
b15	b14	b13	b12	b11	b10	b9	b8
SCKINTR[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

SCK_INTR				Socket Interrupt Request Register			
b7	b6	b5	b4	b3	b2	b1	b0
SCKINTR[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

continued on next page

10.26.1 SCK_INTR *(continued)*

The SCK_INTR registers contain the interrupt request bits for each socket in the respective IP, which is the logical OR of all interrupt request bits in each socket. This register is read-only – interrupt bits are cleared by clearing the interrupt cause or bit in the socket itself.

Bit	Name	Description
255:0	**SCKINTR[255:0]**	Socket <x> asserts interrupt when bit <x> is set in this vector. Multiple bits may be set to 1 simultaneously. This register is only as wide as the number of sockets in the adapter; 256 is just the maximum width. All other bits always return 0.

10.26.2 ADAPTER_STATUS

Adapter Global Status Register

ADAPTER_STATUS			Adapter Global Status Fields				0xFC
b31	b30	b29	b28	b27	b26	b25	b24

ADAPTER_STATUS			Adapter Global Status Fields				
b23	b22	b21	b20	b19	b18	b17	b16
WORD_SIZE[1:0]		FQ_SIZE[5:0]					
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
H	H	H	H	H	H	H	H

ADAPTER_STATUS			Adapter Global Status Fields				
b15	b14	b13	b12	b11	b10	b9	b8
IG_ONLY[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
H	H	H	H	H	H	H	H

ADAPTER_STATUS			Adapter Global Status Fields				
b7	b6	b5	b4	b3	b2	b1	b0
TTL_SOCKETS[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
H	H	H	H	H	H	H	H

Bit	Name	Description
23:22	**WORD_SIZE[1:0]**	Internal word size of the prefetch queue (FQ); not the same as bus width of AHB bus or thread interface to the IP. 0 32b 1 64b 2 128b 3 256b
21:16	**FQ_SIZE[5:0]**	Number of words in a socket fetch queue (FQ). The total buffer space in the adapter is EG_SOCKETS*FQ_SIZE words of size WORD_SIZE.
15:8	**IG_ONLY[7:0]**	First socket number that is ingress only. 0..IG_ONLY-1: Sockets capable of both in and egress IG_ONLY..TTL_SOCKETS-1: Ingress sockets only
7:0	**TTL_SOCKETS[7:0]**	Total number of sockets in this adapter. This number is different for each instance of the adapter and varies with the core IP needs.

10.26.3 SIB_ID

Storage Interface Block ID Register

SIB_ID							
Storage Interface Block ID register							
0xE0027F00							
b31	b30	b29	b28	b27	b26	b25	b24
BLOCK_VERSION[15:8]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
BLOCK_VERSION[7:0]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	1
b15	b14	b13	b12	b11	b10	b9	b8
BLOCK_ID[15:8]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
BLOCK_ID[7:0]							
R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R
0	0	0	0	0	0	1	0

Bit	Name	Description
16:31	**BLOCK_VERSION**	Version number for the SIB block IP. Set to 0x0001.
0:15	**BLOCK_ID**	Block ID for the SIB IP. Set to 0x0002.

10.26.4 SIB_POWER

Power Control Register for Storage Interface Block

A single block controls both storage ports (S0 and S1) on the FX3S device. The clock for at least one of the storage port needs to be enabled before the block is powered on.

SIB_POWER							
SIB Power Control							
0xE0027F04							
b31	b30	b29	b28	b27	b26	b25	b24
RESETN							
R							
R/W							
0							
b23	b22	b21	b20	b19	b18	b17	b16
b15	b14	b13	b12	b11	b10	b9	b8
b7	b6	b5	b4	b3	b2	b1	b0
							ACTIVE
							W
							R
							0

Bit	Name	Description
31	**RESETN**	Active LOW reset signal for all logic in the block. After setting this bit to 1, firmware will poll and wait for the 'active' bit to assert.
0	**ACTIVE**	Indicates whether the block is powered up and active.

10.26.5 SDMMC_CMD_IDX

SDMMC Command Index Register

The cmd field specifies the command-index that is to be included in the command sent. When the command is sent, HW send out start and transmission-bit followed by (SDMMC_CMD_FMT.CMDFRMT + 1 bits), 7-bit CRC, and end bit. The (SDMMC_CMD_FMT.CMDFRMT + 1) bits include as many bits as necessary in the following order: command index (6 bits), argument (starting from bit 31 of SDMMC_CMD_ARG0), SDMMC_CMD_ARG1.

There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020000 + (port * 0x0400).

SDMMC_CMD_IDX							
SDMMC Command Index							
0xE0020000							
b31	b30	b29	b28	b27	b26	b25	b24
b23	b22	b21	b20	b19	b18	b17	b16
b15	b14	b13	b12	b11	b10	b9	b8
b7	b6	b5	b4	b3	b2	b1	b0
		CMD[5:0]					
		R					
		R/W					
		0	0	0	0	0	0

Bit	Name	Description
5:0	**CMD**	6-bit command index.

10.26.6 SDMMC_CMD_ARG0

SDMMCC Command Argument Register

The SDMMC_CMD_ARG0 register holds the lower 32 bits of the SD/MMC/SDIO command argument. Any additional argument bits for expanded command may be placed in the SDMMC_CMD_ARG1 register. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020004 + (port * 0x0400).

SDMMC_CMD_ARG0							
SDMMC Command Argument 0							
0xE0020004							
b31	b30	b29	b28	b27	b26	b25	b24
ARG[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
ARG[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b15	b14	b13	b12	b11	b10	b9	b8
ARG[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
ARG[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Bit	Name	Description
31:0	**ARG**	Lower 32 bits of the command argument.

10.26.7 SDMMC_CMD_ARG1

SDMMCC Command Argument Register

The SDMMC_CMD_ARG1 register holds the upper 32 bits of the SD/MMC/SDIO command argument, if the argument is longer than 32 bits. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020008 + (port * 0x0400).

SDMMC_CMD_ARG1							
SDMMC Command Argument 1							
0xE0020008							
b31	b30	b29	b28	b27	b26	b25	b24
ARG[63:56]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
ARG[55:48]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b15	b14	b13	b12	b11	b10	b9	b8
ARG[47:40]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
ARG[39:32]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Bit	Name	Description
31:0	**ARG**	Upper 32 bits of the command argument.

10.26.8 SDMMC_RESP_IDX

SDMMC Response Index Register

The first eight bits of the response is captured in the RESP_IDX register. These eight bits include start, transmission, and command index fields. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe002000C + (port * 0x0400).

SDMMC_RESP_IDX							
SDMMC Response Index							
0xE002000C							
b31	b30	b29	b28	b27	b26	b25	b24
b23	b22	b21	b20	b19	b18	b17	b16
b15	b14	b13	b12	b11	b10	b9	b8
b7	b6	b5	b4	b3	b2	b1	b0
ST_BIT	TR_BIT	CMD[5:0]					
W	W	W	W	W	W	W	W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Bit	Name	Description
7	**ST_BIT**	Start-bit: As received in response.
6	**TR_BIT**	Transmission bit: As received in response.
5:0	**CMD**	Command index. As received in response

10.26.9 SDMMC_RESP_REG0

SDMMC Response Register0

Response data received from the card on the response pin. The response from bit 8 is placed in this register starting at the MSB of the register. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020010 + (port * 0x0400).

SDMMC_RESP_REG0							
SDMMC Command Response 0							
0xE0020010							
b31	b30	b29	b28	b27	b26	b25	b24
RESP[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
RESP[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b15	b14	b13	b12	b11	b10	b9	b8
RESP[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
RESP[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Bit	Name	Description
31:0	RESP	Bits 8 to 39 of the command response from MSB to LSB.

10.26.10 SDMMC_RESP_REG1

SDMMC Response Register1

Response data received from the card on the response pin. The response from bit 40 is placed in this register starting at the MSB of the register. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020014 + (port * 0x0400).

SDMMC_RESP_REG1							
SDMMC Command Response 1							
0xE0020014							
b31	b30	b29	b28	b27	b26	b25	b24
RESP[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
RESP[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b15	b14	b13	b12	b11	b10	b9	b8
RESP[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
RESP[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Bit	Name	Description
31:0	RESP	Bits 40 to 71 of the command response from MSB to LSB.

10.26.11 SDMMC_RESP_REG2

SDMMC Response Register2

Response data received from the card on the response pin. The response from bit 72 is placed in this register starting at the MSB of the register. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020018 + (port * 0x0400).

SDMMC_RESP_REG2							
SDMMC Command Response 2							
0xE0020018							
b31	b30	b29	b28	b27	b26	b25	b24
RESP[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
RESP[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b15	b14	b13	b12	b11	b10	b9	b8
RESP[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
RESP[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Bit	Name	Description
31:0	RESP	Bits 72 to 103 of the command response from MSB to LSB.

10.26.12 SDMMC_RESP_REG3

SDMMC Response Register3

Response data received from the card on the response pin. The response from bit 104 is placed in this register starting at the MSB of the register. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe002001C + (port * 0x0400).

SDMMC_RESP_REG3							
SDMMC Command Response 3							
0xE002001C							
b31	b30	b29	b28	b27	b26	b25	b24
RESP[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
RESP[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b15	b14	b13	b12	b11	b10	b9	b8
RESP[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
RESP[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Bit	Name	Description
31:0	RESP	Bits 104 to 135 of the command response from MSB to LSB.

10.26.13 SDMMC_RESP_REG4

SDMMC Response Register4

Response data received from the card on the response pin. The response from bit 136 is placed in this register starting at the MSB of the register. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020020 + (port * 0x0400).

SDMMC_RESP_REG4							
SDMMC Command Response 4							
0xE0020020							
b31	b30	b29	b28	b27	b26	b25	b24
RESP[31:24]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
RESP[23:16]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b15	b14	b13	b12	b11	b10	b9	b8
RESP[15:8]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
RESP[7:0]							
R	R	R	R	R	R	R	R
W	W	W	W	W	W	W	W
0	0	0	0	0	0	0	0

Bit	Name	Description
31:0	RESP	Bits 136 to 167 of the command response from MSB to LSB.

10.26.14 SDMMC_CMD_RESP_FMT

SD/MMC Command Response Format

Configuration register that specifies the length and format of the command and response. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020024 + (port * 0x0400).

SDMMC_CMD_RESP_FMT							
SDMMC Command Response Format							
0xE0020024							
b31	b30	b29	b28	b27	b26	b25	b24
RESPCRC_START	**CORCRC_ODD**		**R_CRC_EN**	**CORCRC**	**RESPCONF[10:8]**		
R	R		R	R	R	R	R
R/W	R/W		R/W	R/W	R/W	R/W	R/W
0	0		0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
RESPCONF[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	1	1	1	1	1	0
b15	b14	b13	b12	b11	b10	b9	b8
b7	b6	b5	b4	b3	b2	b1	b0
		CMDFRMT[5:0]					
		R	R	R	R	R	R
		R/W	R/W	R/W	R/W	R/W	R/W
		1	0	0	1	0	1

Bit	Name	Description
31	**RESPCRC_START**	0: Compute response CRC from bit 0. 1: Compute response CRC from bit 8. Option 1 is used for R2 response.
30	**CORCRC_ODD**	0: corrupt CRC16 for even bytes. 1: corrupt CRC16 for odd bytes. This bit is effective only during DDR mode of operation.
28	**R_CRC_EN**	1: Enable CRC check in response. 0: Don't check CRC in response.
27	**CORCRC**	Enable CRC error injection on outgoing data CRC16 field.
26:16	**RESPCONF**	Response length in bits that does not include start, transmit, CRC7, and end bits. 0 indicates that no response is expected. Non-zero values indicate the size in bits
5:0	**CMDFRMT**	Command length minus 1, in bits, that includes command index bits and argument bits. This length does not include start, transmit, CRC7 and end bits.

10.26.15 SDMMC_BLOCK_COUNT

SDMMC Block Count Register

This register holds the number of data blocks that should be completed during the ongoing SD/MMC command transfer. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020028 + (port * 0x0400).

SDMMC_BLOCK_COUNT							
SDMMC Block Count							
0xE0020028							
b31	b30	b29	b28	b27	b26	b25	b24
NOBD[31:24]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
NOBD[23:16]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
b15	b14	b13	b12	b11	b10	b9	b8
NOBD[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
NOBD[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	1

Bit	Name	Description
31:0	**NOBD**	Number of data blocks to be transferred.

10.26.16 SDMMC_BLOCK_LEN

SDMMC Block Length Register

This register holds the block length for transfer. The CRC is computed on this block length. It is expected that the block length used in SET_BLOCKLEN command matches this block length. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe002002C + (port * 0x0400).

SDMMC_BLOCK_LEN							
SDMMC Block Length							
0xE002002C							
b31	b30	b29	b28	b27	b26	b25	b24
DATABLKS[31:24]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
DATABLKS[23:16]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
b15	b14	b13	b12	b11	b10	b9	b8
DATABLKS[15:8]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	1	0
b7	b6	b5	b4	b3	b2	b1	b0
DATABLKS[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Name	Description
31:0	DATABLKS	Data block size in bytes. LSB is ignored for DDR mode of operation, because block length needs to be even.

10.26.17 SDMMC_MODE_CFG

SD/MMC Mode Configuration Register

Configures clock, signaling, and so on for each of the storage ports. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020030 + (port * 0x0400).

SDMMC_MODE_CFG							
SDMMC Mode Configuration							
0xE0020030							
b31	b30	b29	b28	b27	b26	b25	b24
IDLE_STOP_CLK_EN	IDLE_STOP_CLK_DELAY[6:0]						
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	1	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
	EXP_BOOT_ACK	BLK_END_STOP_CLK		RD_END_STOP_CLK_EN	CARD_DETECT_POLARITY		WR_STOP_CLK_EN
	R	R		R	R		R
	R/W	R/W		R/W	R/W		R/W
	0	0		0	0		0
b15	b14	b13	b12	b11	b10	b9	b8
RD_STOP_CLK_EN	EN_CMD_COMP	DATABUSWIDTH[1:0]		MODE[1:0]		SIGNALING[3:2]	
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	1	0	0
b7	b6	b5	b4	b3	b2	b1	b0
SIGNALING[1:0]		DLL_BYPASS_EN					
R	R	R					
R/W	R/W	R/W					
0	0	1					

Bit	Name	Description
31	IDLE_STOP_CLK_EN	0: Do not stop clock automatically after each command. 1: Stop clock automatically IDLE_STOP_CLK_DELAY cycles after each command. Clock is not stopped if the card indicates busy and stopped when the card gets out of busy.
30:24	IDLE_STOP_CLK_DELAY	Number of clock cycles delay after the completion of the last command after which the clock will be stopped. This value may be changed only when IDLE_STOP_CLK_EN is 0.
22	EXP_BOOT_ACK	1: Boot acknowledgement is expected when booting from eMMC. 0: Boot acknowledgement is not expected.
21	BLK_END_STOP_CLK	0: SDMMC interface clock can be stopped at any time when data overflow/underflow is detected. 1: SDMMC interface clock can be stopped only at the end of data block transfer. Clock is not stopped in the middle of a block data transfer. It is recommended that this bit should be set to 1.

continued on next page

10.26.17 SDMMC_MODE_CFG *(Continued)*

19	**RD_END_STOP_CLK_EN**	0: Do not Stop SD/MMC interface clock at the end of a read data transfer. 1: Stop interface clock at the end of read data transfer.
18	**CARD_DETECT_POLARITY**	0: CardDetect GPIO is active LOW. 1: CardDetect GPIO is active HIGH. This value is used by hardware to reflect the S0S1_INS status in the SDMMC_STATUS.card_detect register bit.
16	**WR_STOP_CLK_EN**	This enables SIB to perform detection of underflow and stop the clock to the SD/SDIO/MMC card during data transfer. 1: Enable stop clock. 0: Disable stop clock.
15	**RD_STOP_CLK_EN**	This enables SIB to perform detection of overflow and stop the clock to the SD/SDIO/MMC card during data transfer: 1: Enable stop clock. 0: Disable stop clock. This bit should always be set during read operations to prevent loss of data.
14	**EN_CMD_COMP**	Enable command completion feature for CE-ATA interface. 0: Don't detect command completion event. 1: Detect command completion event. When EN_CMD_COMP is enabled, RD_END_STOP_CLK should not be enabled because card requires a clock edge to provide command completion signal.
13:12	**DATABUSWIDTH**	SIB data bus width: 00: 1-bit 01: 4-bit 10: 8-bit 11: Reserved
11:10	**MODE**	SIB interface mode: 00: Reserved 01: MMC (or CE-ATA) 10: SD 11: Reserved
9:6	**SIGNALING**	SD/MMC data signaling parameters: 0000: DS - Default-Speed (0 - 20 MHz for MMC; 0 - 25 MHz for SD) 0001: HS - High-Speed (0 - 52 MHz for MMC; 0 - 50 MHz for SD) 0010: SDR12: SD SDR 25 MHz @1.8 V 0011: SDR25: SD SDR 50 MHz @1.8 V 0100: SDR50_FD: SD SDR 100 MHz @ 1.8 V 0101 - 0111: Reserved 1000: DDR52_V33_MMC 1001: DDR52_V18_MMC 1010: Reserved 1011: DDR50_V18_SD: As defined in SD3.0 1100 - 1111: Reserved
5	**DLL_BYPASS_EN**	Setting this bit causes the SIB internal clock to be driven out on the pad bypassing the DLL. It is recommended that this bit should always be set to 1.

10.26.18 SDMMC_DATA_CFG

SDMMC Data Configuration Register

Holds configuration parameters that affect SD/MMC data transfers. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020034 + (port * 0x0400).

SDMMC_DATA_CFG							
SDMMC Data Configuration							
0xE0020034							
b31	b30	b29	b28	b27	b26	b25	b24
b23	b22	b21	b20	b19	b18	b17	b16
b15	b14	b13	b12	b11	b10	b9	b8
b7	b6	b5	b4	b3	b2	b1	b0
				EXPBUSY	**EXPCRCR**	**CARD_BUSY_DET**	**RD_CRC_EN**
				R	R	R	R
				R/W	R/W	R/W	R/W
				1	1	0	1

Bit	Name	Description
3	**EXPBUSY**	1: Expect BUSY state after each block of write data. 0: Do not expect BUSY after each block of write data.
2	**EXPCRCR**	1: Expect and check CRC response status after block of write data. 0: Do not expect CRC response after a block of write data.
1	**CARD_BUSY_DET**	1: Enable the SIB state machine to wait for busy before sending first block of data. 0: SIB state machine pushes 1st block of data without checking busy status.
0	**RD_CRC_EN**	1: Check CRC16 on incoming blocks of data. 0: Ignore CRC16 checking on incoming data.

10.26.19 SDMMC_CS

Card Command and Status Register

This register is used to initiate SD/MMC/SDIO commands and data transfers. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020038 + (port * 0x0400).

SDMMC_CS							
SDMMC Command and Status							
0xE0020038							
b31	b30	b29	b28	b27	b26	b25	b24
EOP_EOT			**SOCKET[4:0]**				
R			R	R	R	R	R
R/W			R/W	R/W	R/W	R/W	R/W
0			0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
b15	b14	b13	b12	b11	b10	b9	b8
SDIO_READ_WAIT_EN	**SDMMC_CLK_DIS**			**CLR_BOOTCMD**	**CLR_RDDCARD**	**CLR_WRDCARD**	**CLR_SNDCMD**
R	R			R/W0C	R/W0C	R/W0C	R/W0C
R/W	R/W			R/W1S	R/W1S	R/W1S	R/W1S
0	0			0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
CMDCOMP_DIS	**BOOTCMD**	**RSTCONT**			**RDDCARD**	**WRDCARD**	**SNDCMD**
R/W0C	R/W0C	R/W0C			R/W0C	R/W0C	R/W0C
R/W1S	R/W1S	R/W1S			R/W1S	R/W1S	R/W1S
0	0	0			0	0	0

Bit	Name	Description
31	**EOP_EOT**	Configure end-of-packet signaling to DMA adapter. 0: Set EOP in the descriptor with last byte of last block during read. 1: Set EOP in the descriptor with last byte of every block.
28:24	**SOCKET**	SIB socket number to be used for data read/write operation. Should be in the range 0–7.
15	**SDIO_READ_WAIT_EN**	Enable Read Wait to SDIO device on DAT[2].
14	**SDMMC_CLK_DIS**	Manual enable/disable for the SIB interface clock to enable power saving. 1: Clock is disabled 0: Clock is enabled
11	**CLR_BOOTCMD**	FW writes ‘1’ to clear the BOOTCMD bit and abort boot operation. HW writes ‘0’ after the BOOTCMD bit is cleared.
10	**CLR_RDDCARD**	FW writes ‘1’ to clear the RDDCARD bit so that the next command can be issued. This bit is cleared by HW when RDDCARD is cleared.

continued on next page

10.26.19 SDMMC_CS

9	**CLR_WRDCARD**	FW writes ‘1’ to clear the WRDCARD bit so that the next command can be issued. This bit is cleared by HW when WRDCARD is cleared.
8	**CLR_SNDCMD**	FW writes ‘1’ to clear the SNDCMD bit so that the next command can be issued. This bit is cleared by HW when SNDCMD is cleared.
7	**CMDCOMP_DIS**	FW writes ‘1’ to send a Command Completion Disable signal to the CE-ATA device. HW clears this bit after signal is sent.
6	**BOOTCMD**	FW writes ‘1’ to initiate a CMD_LINE_LOW boot or alternate boot command. For alternate boot, the command should be sent by FW. HW writes ‘0’ when NOBD blocks are transferred or when FW aborts the boot by writing ‘1’ to the CLR_BOOTCMD bit.
5	**RSTCONT**	RSTCONT is used for error recovery. FW writes ‘1’ to bring SIB state machines to a known state. HW writes ‘0’ when the reset is completed. Values of CFG registers are not affected by RSTCONT and only internal queues are flushed. If a complete reset of SIB controller is required, then SIB_POWER.RESETN should be used.
2	**RDDCARD**	FW writes ‘1’ to initiate data read from SD/MMC/SDIO peripheral. HW writes ‘0’ when the transfer is completed. FW may clear this bit by writing 1 to the CLR_RDDCARD bit.
1	**WRDCARD**	FW writes ‘1’ to initiate data write to SD/MMC/SDIO peripheral. HW writes ‘0’ when the transfer is completed. FW may clear this bit by writing 1 to the CLR_WRDCARD bit.
0	**SNDCMD**	FW writes ‘1’ to initiate sending of command. HW writes ‘0’ when command is sent and the response, if any, is received. FW may clear this bit on response timeout by writing 1 to CLR_SNDCMD bit.

10.26.20 SDMMC_STATUS

SDMMC Status Register

Reflects the current status of the SIB state machines. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe002003C + (port * 0x0400).

SDMMC_STATUS							
SDMMC Status							
0xE002003C							
b31	b30	b29	b28	b27	b26	b25	b24
CRC16_EVEN_ERROR	**CRC16_ODD_ERROR**		**DATA_SM_BUSY**	**COMMAND_SM_BUSY**	**CRCFC[2:0]**		
R/W	R/W		R/W	R/W	R/W	R/W	R/W
R	R		R	R	R	R	R
0	0		0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
					RD_END_DATA_ERROR	**DAT0_STAT**	**DLL_LOCKED**
					R/W	R/W	R/W
					R	R	R
					0	0	0
b15	b14	b13	b12	b11	b10	b9	b8
DLL_LOST_LOCK	**BOOT_ACK**	**CMD_COMP**	**FIFO_U_DET**	**FIFO_O_DET**	**DAT3_STAT**	**CARD_DETECT**	**SDIO_INTR**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
1	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
CRC16_ERROR	**RD_DATA_TIMEOUT**	**BLOCKS_RECEIVED**	**BLOCK_COMP**	**CRC7_ERROR**	**RESPTIMEOUT**	**RCVDRES**	**CMDSENT**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Bit	Name	Description
31	**CRC16_EVEN_ERROR**	Indicates that CRC error was encountered on even bytes of data transfer. This bit may be interpreted in DDR mode data transfer only. This bit will be cleared when CRC16_ERROR is cleared.
30	**CRC16_ODD_ERROR**	Indicates that CRC error was encountered on odd bytes of data transfer. This bit may be interpreted in DDR mode data transfer only. This bit will be cleared when CRC16_ERROR is cleared.
28	**DATA_SM_BUSY**	1: Data state machine is busy. 0: Data state machine is idle.
27	**COMMAND_SM_BUSY**	1: Command state machine is busy. 0: Command state machine is idle.
26:24	**CRCFC**	3-bit CRC response received from the card following a data write.

continued in next page

10.26.20 SDMMC_STATUS

18	**RD_END_DATA_ERROR**	This error is flagged when command, response or NRC counting is in progress when the last byte of the last block of read-data is read while RD_END_CLK_STOP is enabled. This condition causes data from card to be read out of card and dropped. FW should re-issue read command to card to restart read command. This bit is cleared when SDMMC_CS.RDDCARD is set by FW.
17	**DAT0_STAT**	Reflects the current status of the SD_DAT[0] pin.
16	**DLL_LOCKED**	Reflects the current lock status of the SIB DLL. 1: DLL is locked. 0: DLL Is not locked.
15	**DLL_LOST_LOCK**	Reflects the current lock status of the SIB DLL. 1: DLL is not locked. 0: DLL Is locked.
14	**BOOT_ACK**	HW sets this bit to '1' when a MMC boot acknowledgement is detected. This bit is cleared when FW clears the SDMMC_MODE_CFG.EXP_BOOT_ACK bit.
13	**CMD_COMP**	HW sets this bit to '1' when Command Completion signaling from a CE-ATA peripheral is detected. This bit is cleared when the FW writes '1' to SDMMC_CS.SNDCMD again.
12	**FIFO_U_DET**	HW sets this bit to '1' if a FIFO underflow is detected during a write operation. The bit is cleared when FW writes '1' to SDMMC_CS.WRDCARD again.
11	**FIFO_O_DET**	HW sets this bit to '1' if a FIFO overflow is detected during a read operation. The bit is cleared when FW writes '1' to SDMMC_CS.RDDCARD again.
10	**DAT3_STAT**	Reflects the state of the SD_DAT[3] pin.
9	**CARD_DETECT**	Reflects the state of the S0S1_INS pin. 0: No card inserted (GPIO is inactive). 1: Card is inserted (GPIO is active).
8	**SDIO_INTR**	HW sets this bit to '1' when it detects an SDIO interrupt condition. This bit will represent the latest result of the interrupt sampling.
7	**CRC16_ERROR**	HW sets this bit to '1' if a CRC error is detected during a read or write data operation. This bit is cleared when FW writes '1' to SDMMC_CS.RDDCARD or WRDCARD again.
6	**RD_DATA_TIMEOUT**	HW sets this bit to '1' if a read data operation times out. This bit is cleared when the FW writes '1' to SDMMC_CS.RDDCARD again.
5	**BLOCKS_RECEIVED**	HW sets this bit to '1' when the read of NOBD blocks of data is completed. This bit is cleared when the FW writes '1' to SDMMC_CS.RDDCARD again.
4	**BLOCK_COMP**	HW sets this bit to '1' when the write of NOBD blocks of data is completed. This bit is cleared when the FW writes '1' to SDMMC_CS.WRDCARD again.
3	**CRC7_ERROR**	HW sets this bit to '1' if a CRC error is detected in the response received. This bit is cleared when the FW writes '1' to SDMMC_CS.SNDCMD again.
2	**RESPTIMEOUT**	HW sets this bit to '1' if no response was received within the timeout period. This bit is cleared when the FW writes '1' to SDMMC_CS.SNDCMD again.
1	**RCVDRES**	HW sets this bit to '1' when the command response is received. This bit is cleared when the FW writes '1' to SDMMC_CS.SNDCMD again.
0	**CMDSENT**	HW sets this bit to '1' when a command is sent. This bit is cleared when the FW writes '1' to SDMMC_CS.SNDCMD again.

10.26.21 SDMMC_INTR

SDMMC Interrupt Status Register

These bits display interrupt requests. These bits are set to '1' whenever a bit in SDMMC_STATUS changes from 0 to 1. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020040 + (port * 0x0400).

SDMMC_INTR							
SDMMC Interrupt Status							
0xE0020040							
b31	b30	b29	b28	b27	b26	b25	b24
b23	b22	b21	b20	b19	b18	b17	b16
					RD_END_DATA_ERROR	**DAT0_OUTOF_BUSY**	**DLL_LOCKED**
					R/W1S	R/W1S	R/W1S
					R/W1C	R/W1C	R/W1C
					0	0	0
b15	b14	b13	b12	b11	b10	b9	b8
DLL_LOST_LOCK	**BOOT_ACK**	**CMD_COMP**	**FIFO_U_DET**	**FIFO_O_DET**	**DAT3_CHANGE**	**CARD_DETECT**	**SDIO_INTR**
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
0	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
CRC16_ERROR	**RD_DATA_TIMEOUT**	**BLOCKS_RECEIVED**	**BLOCK_COMP**	**CRC7_ERROR**	**RESPTIMEOUT**	**RCVDRES**	**CMDSENT**
R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S	R/W1S
R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C	R/W1C
0	0	0	0	0	0	0	0

Bit	Name	Description
18	**RD_END_DATA_ERROR**	HW writes 1 to indicate that the RD_END_DATA error interrupt is active. FW writes 1 to this bit to clear the interrupt.
17	**DAT0_OUTOF_BUSY**	HW writes 1 to indicate that the DAT0 pin state has changed from 0 to 1. FW writes 1 to clear the interrupt.
16	**DLL_LOCKED**	HW writes 1 to indicate that the DLL lock is achieved. FW writes 1 to clear the interrupt.
15	**DLL_LOST_LOCK**	HW writes 1 to indicate that the DLL lock is lost. FW writes 1 to clear the interrupt.
14	**BOOT_ACK**	HW writes 1 to indicate that a BOOT acknowledgement is received. FW writes 1 to clear the interrupt.

continued on next page

10.26.21 SDMMC_INTR

13	**CMD_COMP**	HW writes 1 to indicate that Command Completion signaling from a CE-ATA peripheral is detected. FW writes 1 to clear the interrupt.
12	**FIFO_U_DET**	HW writes 1 to indicate a FIFO underflow is detected. FW writes 1 to clear the interrupt.
11	**FIFO_O_DET**	HW writes 1 to indicate a FIFO overflow is detected. FW writes 1 to clear the interrupt.
10	**DAT3_CHANGE**	HW writes 1 to indicate that the SD_DAT[3] pin state has changed. FW writes 1 to clear the interrupt.
9	**CARD_DETECT**	HW writes 1 to indicate that the S0S1_INS pin state has changed. FW writes 1 to clear the interrupt.
8	**SDIO_INTR**	HW writes 1 to indicate that an SDIO interrupt is detected. FW writes 1 to clear the interrupt.
7	**CRC16_ERROR**	HW writes 1 to indicate that a data CRC error is detected. FW writes 1 to clear the interrupt.
6	**RD_DATA_TIMEOUT**	HW writes 1 to indicate that a read operation has timed out. FW writes 1 to clear the interrupt.
5	**BLOCKS_RECEIVED**	HW writes 1 to indicate that a read operation is completed. FW writes 1 to clear the interrupt.
4	**BLOCK_COMP**	HW writes 1 to indicate that a write operation is completed. FW writes 1 to clear the interrupt.
3	**CRC7_ERROR**	HW writes 1 to indicate that a response CRC error is detected. FW writes 1 to clear the interrupt.
2	**RESPTIMEOUT**	HW writes 1 to indicate that a response timeout occurred. FW writes 1 to clear the interrupt.
1	**RCVDRES**	HW writes 1 to indicate that a command response is received. FW writes 1 to clear the interrupt.
0	**CMDSENT**	HW writes 1 to indicate that a command is sent to the target device. FW writes 1 to clear the interrupt.

10.26.22 SDMMC_INTR_MASK

SDMMC Interrupt Mask Register

These bits correspond to the interrupt status bits in the SDMMC_INTR register and control whether the interrupt status should trigger an interrupt to the CPU. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020044 + (port * 0x0400).

SDMMC_INTR_MASK							
SDMMC Interrupt Mask							
0xE0020044							
b31	b30	b29	b28	b27	b26	b25	b24
b23	b22	b21	b20	b19	b18	b17	b16
					RD_END_DATA_ERROR	**DAT0_OUTOF_BUSY**	**DLL_LOCKED**
					R	R	R
					R/W	R/W	R/W
					0	0	0
b15	b14	b13	b12	b11	b10	b9	b8
DLL_LOST_LOCK	**BOOT_ACK**	**CMD_COMP**	**FIFO_U_DET**	**FIFO_O_DET**	**DAT3_CHANGE**	**CARD_DETECT**	**SDIO_INTR**
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
CRC16_ERROR	**RD_DATA_TIMEOUT**	**BLOCKS_RECEIVED**	**BLOCK_COMP**	**CRC7_ERROR**	**RESPTIMEOUT**	**RCVDRES**	**CMDSENT**
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Name	Description
18	**RD_END_DATA_ERROR**	1: Enable interrupt due to RD_END data error. 0: Disable interrupt due to RD_END data error.
17	**DAT0_OUTOF_BUSY**	1: Enable interrupt due to DAT0 state change. 0: Disable interrupt due to DAT0 state change.
16	**DLL_LOCKED**	1: Enable interrupt due to DLL lock completion. 0: Disable interrupt due to DLL lock completion.
15	**DLL_LOST_LOCK**	1: Enable interrupt due to DLL lock loss. 0: Disable interrupt due to DLL lock loss.
14	**BOOT_ACK**	1: Enable interrupt due to BOOT acknowledgement. 0: Disable interrupt due to BOOT acknowledgement.

continued on next page

10.26.22 SDMMC_INTR_MASK *(continued)*

13	**CMD_COMP**	1: Enable interrupt due to CE-ATA command completion. 0: Disable interrupt due to CE-ATA command completion.
12	**FIFO_U_DET**	1: Enable interrupt due to FIFO underflow. 0: Disable interrupt due to FIFO underflow.
11	**FIFO_O_DET**	1: Enable interrupt due to FIFO overflow. 0: Disable interrupt due to FIFO overflow.
10	**DAT3_CHANGE**	1: Enable interrupt due to DAT3 state change. 0: Disable interrupt due to DAT3 state change.
9	**CARD_DETECT**	1: Enable interrupt due to S0S1_INS change. 0: Disable interrupt due to S0S1_INS change.
8	**SDIO_INTR**	1: Enable interrupt due to SDIO interrupt. 0: Disable interrupt due to SDIO interrupt.
7	**CRC16_ERROR**	1: Enable interrupt due to data CRC error. 0: Disable interrupt due to data CRC error.
6	**RD_DATA_TIMEOUT**	1: Enable interrupt due to read timeout. 0: Disable interrupt due to read timeout.
5	**BLOCKS_RECEIVED**	1: Enable interrupt due to read completion. 0: Disable interrupt due to read completion.
4	**BLOCK_COMP**	1: Enable interrupt due to write completion. 0: Disable interrupt due to write completion.
3	**CRC7_ERROR**	1: Enable interrupt due to response CRC error. 0: Disable interrupt due to response CRC error.
2	**RESPTIMEOUT**	1: Enable interrupt due to response timeout. 0: Disable interrupt due to response timeout.
1	**RCVDRES**	1: Enable interrupt due to response reception. 0: Disable interrupt due to response reception.
0	**CMDSENT**	1: Enable interrupt due to command transmission. 0: Disable interrupt due to command transmission.

10.26.23 SDMMC_NCR

SDMMC Command Response Timing Register #1

This register controls timing between command and response. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020048 + (port * 0x0400).

SDMMC_NCR							
SDMMC Command Response Timing Register #1							
0xE0020048							
b31	b30	b29	b28	b27	b26	b25	b24
b23	b22	b21	b20	b19	b18	b17	b16
b15	b14	b13	b12	b11	b10	b9	b8
NRC_MIN[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	1	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
	NCR_MAX[6:0]						
	R	R	R	R	R	R	R
	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	1	0	0	0	1	1	1

Bit	Name	Description
15:8	**NRC_MIN**	Specifies the minimum separation in clock cycles between the end bit of a response and the start bit of the next command.
6:0	**NCR_MAX**	Specifies the timeout period for which the SIB will wait to receive a response, in terms of number of cycles from end bit of command.

10.26.24 SDMMC_NCC_NWR

SDMMC Command Response Timing Register #2

This register controls the timing between multiple commands. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe002004C + (port * 0x0400).

SDMMC_NCC_NWR							
SDMMC Command Response Timing Register #2							
0xE002004C							
b31	b30	b29	b28	b27	b26	b25	b24
b23	b22	b21	b20	b19	b18	b17	b16
b15	b14	b13	b12	b11	b10	b9	b8
NWR_MIN[7:0]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	1	0
b7	b6	b5	b4	b3	b2	b1	b0
				NCC_MIN[3:0]			
				R	R	R	R
				R/W	R/W	R/W	R/W
				1	0	0	0

Bit	Name	Description
15:8	**NWR_MIN**	Specifies the minimum number of clock cycles between end bit of response and start bit of write data.
3:0	**NCC_MIN**	Specifies the minimum number of clock cycles between consecutive commands.

10.26.25 SDMMC_NAC

SDMMC Read Timeout Register

This register controls the timeout period for data read operations. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020050 + (port * 0x0400).

SDMMC_NAC							
SDMMC Read Timeout Register							
0xE0020050							
b31	b30	b29	b28	b27	b26	b25	b24
RDTMOUT[29:22]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
b23	b22	b21	b20	b19	b18	b17	b16
RDTMOUT[21:14]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	1	1	1	1
b15	b14	b13	b12	b11	b10	b9	b8
RDTMOUT[13:6]							
R	R	R	R	R	R	R	R
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1
b7	b6	b5	b4	b3	b2	b1	b0
RDTMOUT[5:0]							
R	R	R	R	R	R		
R/W	R/W	R/W	R/W	R/W	R/W		
1	1	1	1	1	1		

Bit	Name	Description
31:2	RDTMOUT	Specifies the timeout duration in clock cycles to be applied to data read operations.

10.26.26 SDMMC_HW_CTRL

SDMMC Hardware Control Register

This register provides a mechanism to reset eMMC devices. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020054 + (port * 0x0400).

SDMMC_HW_CTRL							
SDMMC Hardware Control Register							
0xE0020054							
b31	b30	b29	b28	b27	b26	b25	b24
b23	b22	b21	b20	b19	b18	b17	b16
b15	b14	b13	b12	b11	b10	b9	b8
b7	b6	b5	b4	b3	b2	b1	b0
							HW_RESET_EMMC
							R
							R/W
							0

Bit	Name	Description
0	**HW_RESET_EMMC**	FW writes ‘1’ to assert the eMMC4.4 card HW reset using the SxMMCRST pin. FW writes ‘0’ to de-assert reset after tRSTW (1 usec) and waits for tRSCA (200 usec) before issuing commands.

10.26.27 SDMMC_DLL_CTRL

SDMMC DLL Control Register

This register is used to configure the SIB DLL. Note that the use of SIB DLL is not recommended; also, the DLL should be kept disabled. There are two copies of this register corresponding to the two storage ports. The address of each register is calculated as 0xe0020058 + (port * 0x0400).

SDMMC_DLL_CTRL							
SDMMC DLL Control							
0xE0020058							
b31	b30	b29	b28	b27	b26	b25	b24
b23	b22	b21	b20	b19	b18	b17	b16
					DLL_RESET_N		
					R		
					R/W		
					0		
b15	b14	b13	b12	b11	b10	b9	b8
	PIN_CLOCK_PHASE_SEL[3:0]				SAMPLE_CMD_PHASE_SEL[3:1]		
	R	R	R	R	R	R	R
	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	0	0	0	0	0	0	0
b7	b6	b5	b4	b3	b2	b1	b0
SAMPLE_CMD_PHASE_SEL[0]	SAMPLE_DATA_PHASE_SEL[3:0]				DLL_STAT	HIGH_FREQ	ENABLE
R	R	R	R	R	R/W	R	R
R/W	R/W	R/W	R/W	R/W	R	R/W	R/W
0	0	0	0	0	0	0	0

Bit	Name	Description
18	**DLL_RESET_N**	1: Bring DLL out of reset 0: Keep DLL in reset.
14:11	**PIN_CLOCK_PHASE_SEL**	Select clock phase (0–15) to be driven out on the SD_CLK pin.
10:7	**SAMPLE_CMD_PHASE_SEL**	Select clock phase (0–15) used to sample the SD_CMD pin to look for response.
6:3	**SAMPLE_DATA_PHASE_SEL**	Select clock phase (0–15) used to sample the SD_DAT[] pins to get read data.
2	**DLL_STAT**	Reflects DLL lock status. 1: DLL is locked. 0: DLL is not locked.
1	**HIGH_FREQ**	Select high frequency mode of DLL operation. 0: Use when clock frequency is less than 70 MHz. 1: Use when clock frequency is greater than 70 MHz.
0	**ENABLE**	1: Enable the SIB DLL 0: Disable the SIB DLL

Revision History

Revision History

Document Title: EZ-USB® FX3™ Technical Reference Manual Document Number: 001-76074				
Revision	**ECN#**	**Issue Date**	**Origin of Change**	**Description of Change**
**	3819447	11/22/2012	DBIR	Register descriptions - Initial release.
*A	4007727	05/22/2013	OSG	Content restructure and rewrite.
*B	4515890	09/26/2014	RSKV	Content restructure and rewrite.
*C	4595112	12/12/2014	GAYA	Updated Figure 5-10
*D	5293633	06/02/2016	RSKV	Updated the template.
*E	5757258	05/31/2017	SHEA	Updated logo and copyright