Xilleon™ 220
Databook

Rev 3.9
Technical Reference Manual
P/N: CHS-215H2-00-39

CONFIDENTIAL MATERIAL

All information contained in this manual is confidential material of ATI Technologies Inc. Unauthorized use or disclosure of the information contained herein is prohibited.

You may be held responsible for any loss or damage suffered by ATI for your unauthorized disclosure hereof, in whole or in part. Please exercise the following precautions:

- Store all hard copies in a secure place when not in use.
- Save all electronic copies on password protected systems.
- Do not reproduce or distribute any portions of this manual in paper or electronic form (except as permitted by ATI).
- Do not post this manual on any LAN or WAN (except as permitted by ATI).

Your protection of the information contained herein may be subject to periodic audit by ATI. This manual is subject to possible recall by ATI.

Table of Contents

Chapter 1: Introduction

Chapter 2: Device Architecture

Chapter 3: Multimedia Subsystem

Chapter 6: PCI Peripherals

Chapter 7: Other Peripherals

Chapter 8: Memory Controller

Chapter 9: Pinout and Strap Descriptions

Chapter 10: Electrical and Physical Characteristics

Chapter 11: Timing Specifications

Chapter 12: EXOR Tree

Appendix A: Pinout Listing

Appendix B: Revision History

Chapter 1
Introduction

1.1 About this Manual

This manual is part of a set of reference documents that provides information necessary to design the Xilleon 220 chip into a set-top box or other consumer electronics device.

1.2 ATI Component Part Number Legend

This manual covers the following parts:

215H21AGA11 (Xilleon 220H A11 stepping)
215H21AGA12 (Xilleon 220H A12 stepping)
215H21AGA21 (Xilleon 220H A21 stepping)

215S21AGA11 (Xilleon 220S A11 stepping).
215S21AGA12 (Xilleon 220S A12 stepping)
215S21AGA21 (Xilleon 220S A21 stepping)

It will be updated periodically to include the latest component revisions and respective additional/changed specifications.

Figure 1-1 below shows how to read the coded information contained in the branded ATI component part number.

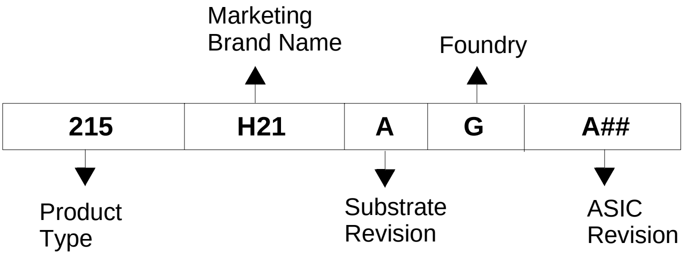


Figure 1-1 Xilleon 220 Branded Part Number Coding Scheme

The following describes each field shown in *Figure 1-1* above:

Product Type

All desktop parts are identified by the 215 code.

Marketing Brand Name

These three digits identify the marketing brand name of the component.

Substrate Revision

This digit tracks the substrate revision for the component. The substrate revision is designated in alphabetical order such that: A = revision 1, B = revision 2, and so forth.

Foundry

This digit identifies the foundry. *Table 1-1* below lists the letters assigned to different foundries.

Table 1-1 Foundry Codes

Foundry Code	Description
C	TSMC Fab3
S	TSMC Fab4
F	TSMC Fab5
G	TSMC Fab6
W	TSMC-WSMC
Q	TSMC-WSMC(8B)
T	TSMC WaferTech
U	UMC 8B (USC)
D	UMC 8D (USC2)
J	UMC 8E (Utek)

ASIC Revision

The three digits identify the ASIC revision.

1.3 Conventions and Notations

The following conventions are used throughout this manual:

Frame Rate

Throughout this document, the term “30i” is used to denote 30 frames/second interlaced. Similarly, “30p” denotes 30 frames/second progressive.

Register and Field Names

Mnemonics in upper-case are used throughout this document to represent hardware register names and field names. The naming conventions for registers and bit fields are indicated below:

- REGISTER_MNEMONIC

  For example, CONFIG_CHIP_ID is the mnemonic for the Configuration Chip ID register.

- REGISTER_MNEMONIC[Bit_Numbers] or FIELD_NAME@REGISTER_MNEMONIC

  For example, CONFIG_CHIP_ID[15:0] refers to the bit field that occupies bit positions 0 through 15 within this register, whereas CFG_CHIP_TYPE@CONFIG_CHIP_ID gives the field name CFG_CHIP_TYPE (Product Type Code) instead of the bits position.

1.3.1 Pin/Signal Names

Mnemonics are used to represent pin and external strap resistors. For example the Device Select pin and the Interrupt Enable external strap are represented by DEVSEL# and ENINT# respectively.

Note: All active-low signal names are identified by the suffix #, e.g. BLANK#, or by the suffix B, e.g. GNTB.

Pins may be identified by their *signal names* or ball references. The terms *pin name* and *signal name* are used indistinguishedly in the industry; in this document pin name is used to allow for situations in which pins have different functions and hence signal names (for example pins in the multimedia group). For such multiplexed pins, the alternate names will be adequately noted.

1.3.2 Pin Types

The assigned codes for the various pin types based on operational characteristics are listed in *Table 1-2* below:

Table 1-2 Pin Type Code

Code	Pin Type / Operational Characteristics
I	Input
O	Output

Table 1-2 Pin Type Code (Continued)

Code	Pin Type / Operational Characteristics
I/O	Bi-Directional
M	Multifunction
Pwr	Power
Gnd	Ground
A	Analog

1.3.3 Numeric Representation

Hexadecimal numbers are appended with “h” (Intel assembly-style notation) whenever there is a risk of ambiguity. Other numbers are assumed to be in decimal.

When the same pin name (except the following running integer) is used for pins that have identical functions, e.g. AD0, AD1, etc, a short-hand notation is used to refer to all of them, i.e., AD[31:0] refers to AD0, AD1, ..., and AD31.

Note: The above shorthand notation is not to be confused with that used to indicate bit occupation in a register. For example, SUBSYS_VEN_ID[15:0] refers to the Product Type Code field that occupies bit positions 0 through 15 within the 16-bit vendor ID register in PCI configuration space.

1.3.4 Acronyms

Standard acronyms used in the literature are presumed known and will not be explained. When in doubt, the reader can refer to *Table 1-3* for a quick check. Less frequently used or ATI-specific acronyms will have the full definition alongside in parenthesis when they appear for the first time in the document.

Table 1-3 Acronyms

Acronym	Full Expression
ACPI	Advanced Configuration and Power Interface
AF	Adaptation Field
AGP	Accelerated Graphics Port
AMC	ATI Multimedia Channel
ATSC	Advanced Television Systems Committee
BGA	Ball Grid Array
BLT	Blit or Bit-Blit
BPP	Bits Per Pixel
CC	Continuity Counter or Closed Captioning
CRC	Cyclic Redundancy Check
DAC	Digital-to-Analog Converter
DDC	Display Data Channel (VESA standard)
DES	Data Encryption Standard

Table 1-3 Acronyms (Continued)

Acronym	Full Expression
DPMS	Display Power Management Signaling (VESA standard)
DSTN	Dual Super-Twisted Nematic - passive matrix
DTS	Decode Time Stamp
DVS	Digital Video System
EDID	Extended Display Identification Data (VESA standard)
EPG	Enhanced Programming Guide
EPROM	Erasable Programmable Read Only Memory
ES	Elementary Stream
ESCR	Extra System Clock Reference
FIFO	First In, First Out
GOP	Group of Pictures
I^2C	Bus Protocol (Philips specification)
I2S	Point-to-point bus protocol for digital sound transmission
LCD	Liquid Crystal Display
LOD	Level of Details (refers to texture pixel selection)
LVDS	Low Voltage Differential Signaling
MDP	MPEG Data Port
MPP	Multimedia Peripheral Port
PEROM	Flash Programmable and Erasable Read Only Memory
PES	Packetized Elementary Stream
PCI	Peripheral Component Interconnect
PCR	Programmable Clock Reference
PID	Packet ID
PIP	Picture in picture
PSI	Program Specific Information
PTS	Presentation Time Stamp
PUSI	Payload Unit Start Indicator
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
ROP	Raster Operation
SDRAM	Synchronous Dynamic Random Access Memory
SGRAM	Synchronous Graphic Random Access Memory
SI	System/Service Information
SPDIF	Sony/Philips Digital Interface (up to 6 channels for digital sound transmission)
STC	System Time Clock
STN	Super-Twisted Nematic
TEI	Transport Error Indicator
TFT	Thin Film Transister - active matrix
TP	Transport Packet
TPS	Transport Processor System
TS	Transport Stream
TMDS	Transmission Minimized Differential Signaling
UV	Chrominance (also CrCb) (corresponds to the color of an image pixel)

Table 1-3 Acronyms (Continued)

Acronym	Full Expression
VBI	Vertical Blank Interval
VFC	VGA Feature Connector
VQ	Vector Quantization (refers to texture compression algorithm)
YCbCr, YUV	The method of video signal color encoding. Includes luma (Y, black and white component) and chroma (UV/CbCr, color component) The terms YCbCr and YUV are often used interchangeably. YCbCr is a digital format of video; YUV is an analog format of video.
ZV	Zoom Video

Chapter 2
Device Architecture

2.1 Overview

Xilleon 220 is the most advanced and highly integrated component for digital set-top boxes, information appliances, and televisions. Xilleon 220 provides dual-stream high-definition decode and display, an assortment of peripheral device controllers, and an embedded microprocessor. It can be used as either an SOC (system-on-a-chip) solution or as a discrete PCI multimedia device when coupled with an external CPU.

Xilleon 220 provides system design flexibility for digital video and audio products. It supports transport stream selection and de-multiplexing in a variety of serial and parallel input formats. It descrambles, filters, and decodes world-wide video and audio formats. Xilleon 220 displays graphics and high-quality scaled video. It also supports dual independent displays, watch-one-record-one, and picture-in-picture display.

2.2 Block Diagram

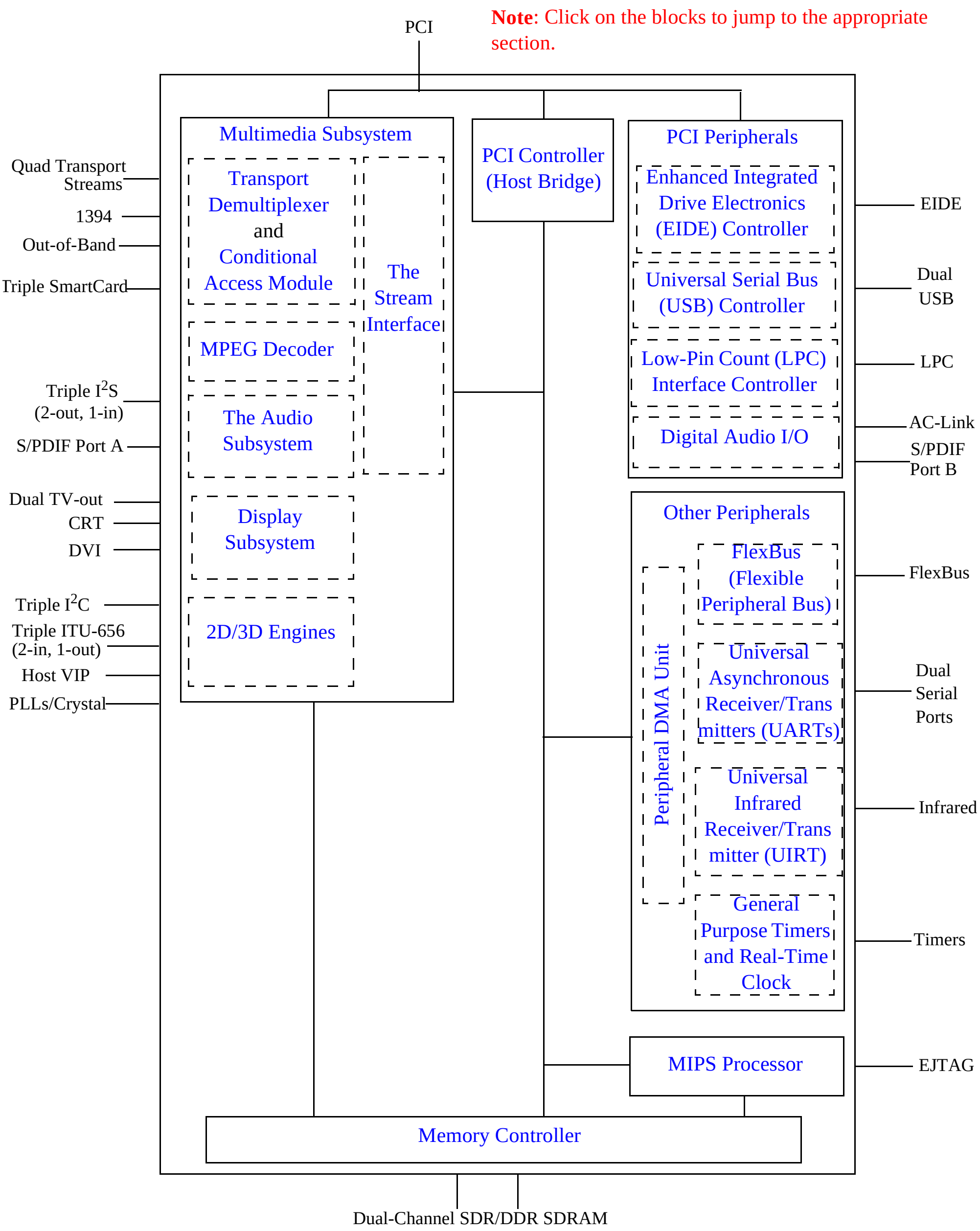

Note: Click on the blocks to jump to the appropriate section.
PCI
Multimedia Subsystem
Transport Demultiplexer and Conditional Access Module
The Stream Interface
MPEG Decoder
The Audio Subsystem
Display Subsystem
2D/3D Engines
PCI Controller (Host Bridge)
PCI Peripherals
Enhanced Integrated Drive Electronics (EIDE) Controller
Universal Serial Bus (USB) Controller
Low-Pin Count (LPC) Interface Controller
Digital Audio I/O
Other Peripherals
Peripheral DMA Unit
FlexBus (Flexible Peripheral Bus)
Universal Asynchronous Receiver/Transmitters (UARTs)
Universal Infrared Receiver/Transmitter (UIRT)
General Purpose Timers and Real-Time Clock
MIPS Processor
Memory Controller
Quad Transport Streams
1394
Out-of-Band
Triple SmartCard
Triple I²S (2-out, 1-in)
S/PDIF Port A
Dual TV-out
CRT
DVI
Triple I²C
Triple ITU-656 (2-in, 1-out)
Host VIP
PLLs/Crystal
EIDE
Dual USB
LPC
AC-Link
S/PDIF Port B
FlexBus
Dual Serial Ports
Infrared
Timers
EJTAG
Dual-Channel SDR/DDR SDRAM

Figure 2-1 Block Diagram of Xilleon 220

2.2.1 Multimedia Subsystem

- MPEG-2 video decoder that can be used for any:
    - Xilleon 220H - 2HD decode
    - Xilleon 220S - SD decode only
- Hardware AC-3 & MPEG-2 audio decode for one audio channel.
  Support for external audio DSPs for other channels and standards.
- Four micro-controlled transport stream demultiplexers, with independent PID filtering and a shared section filter.
- Conditional Access (CA) support for each independent transport stream.
- DVD capability with integrated hardware CSS.
- Picture-in-Picture (PIP) and time-shifting capabilities.
- Dual independent TV-Out channels that support PAL, SECAM, NTSC video standards.
- Dual ITU-656 interfaces to support analog video or external MPEG decoders.
- AC-LINK and dual I2S and SPDIF ports for audio support.
- Serial or parallel 1394A - 5C external component support.
- DVI (TMDS) external component support with integrated DHCP copy protection.
- Three Smart Card interfaces; two for independent networks that have different CA schemes, and one for shopping/banking.
- Out-of-band (OOB) Interface.
- 2D/3D graphics engines.

2.2.2 MIPS Processor

- MIPS, Incorporated 4Kc processor core supporting MIPS32 instruction set.
- Up to 300 MHz operation.
- Supports big-endian and little-endian addressing modes.
- 16 KB 4-way set-associative instruction cache.
- 16 KB 4-way set-associative data cache.
- Virtual Memory Management Unit (MMU) with Translation Lookaside Buffer (TLB).
- Enhanced JTAG debug interface.
- Programmable low-power capability, with wake-up on interrupt.

2.2.3 PCI Interface

- Complies with *PCI Local Bus Specification, Revision 2.2*.
- Built-in Scatter/Gather DMA engines, driven by descriptor tables in memory.

- Host Bridge has intelligent pre-fetch algorithm to minimize read latency to SDRAM (System Memory) when in SOLO Mode.

2.2.4 PCI Peripherals

- Single-channel EIDE disk controller supporting ATA100 operation and up to two disks. Complies with *ANSI-NCITS ATA/ATAPI-5 Rev a1 (May 26, 1999)*. First-party scatter/gather DMA is built into the EIDE Controller.
- Two-port USB 1.1 Host Controller. Complies with *USB specification 1.1* and *USB Open HCI 1.0a*.
- LPC Controller for SuperIO expansion.
- Digital Audio I/O Controller, supporting up to two AC'97 audio CODECs, following the *Intel Audio Codec ’97 Specification, Rev. 1.0*. The AC-Link protocol is used to communicate between Xilleon 220 and the AC’97 CODEC(s).

2.2.5 Other Peripherals

The peripherals in this group do not have PCI configuration-space registers, and do not actually reside directly on a PCI bus like the "PCI Peripherals" do, they are connected to some "internal" bus of the Xilleon 220.

- Two "PC-compatible" UARTs.
- Infrared receiver and transmitter supporting "consumer IR".
- Flexible expansion bus (FlexBus^{TM}) supporting ROM, Flash ROM, "68k-like" devices, and other generic external peripheral devices.
- Six programmable timers, and a Real Time Clock (RTC).
- DMA unit.

2.2.6 Memory Controller

- Dual 32-bit high speed SDRAM memory interfaces, supporting SDR and DDR memory types.

2.3 Device Operating Modes

The Xilleon 220 has been designed for use in a variety of systems. For example, in a low-cost system, the internal MIPS processor can function as the CPU for the system. Alternatively, if a more powerful CPU is required, the Xilleon 220 can be attached to such a CPU via the standard PCI bus. These two basic system configurations are dubbed *Solo* and *Peer* respectively. These terms indicate to the Xilleon 220 what mode it is operating in with respect to the system as a whole.

For the Xilleon 220, the operating mode only has an effect on the MIPS' ability to access the PCI bus, and on which microprocessor (the internal MIPS vs. an external host CPU) has control over the PCI peripherals of the Xilleon 220. The Multimedia subsystem of the Xilleon 220 always operates in the same fashion, regardless of the operating mode.

For the Peer mode, there is a variant known as the *Peer Plus* mode. In this mode, the registers of the internal PCI peripherals of the Xilleon 220 are not visible to the external host CPU.

The table below summarizes, for the various operating modes of the Xilleon 220, how each of the two processors in a system view the PCI Peripherals.

Table 2-1 Differences between Solo and Peer Mode for Internal and External Processors

Operating Mode	Internal MIPS	External Host CPU
SOLO	During PCI enumeration, the multimedia subsystem appears as a "multi-function" device, with the following functions: 0 = Multimedia Subsystem 1 = EIDE Controller 2 = USB Controller 3 = LPC Controller 5,6 = Digital Audio I/O Controller (DAIO)	No external CPU present
PEER	Cannot and does not perform PCI enumeration. The registers of the internal PCI Peripherals are not visible to the internal MIPS.	During PCI enumeration, the Xilleon 220 appears as a "multi-function" device, with the following functions: 0 = Multimedia Subsystem 1 = EIDE Controller 2 = USB Controller 3 = LPC Controller 5,6 = Digital Audio I/O Controller (DAIO)

Table 2-1 Differences between Solo and Peer Mode for Internal and External Processors (Continued)

Operating Mode	Internal MIPS	External Host CPU
PEER Plus	During PCI enumeration, the PCI Peripherals appear as a "multi-function" device, with the following functions: 0 = EIDE Controller 2 = USB Controller 3 = LPC Controller 5,6 = Digital Audio I/O Controller (DAIO) The internal PCI Peripherals reside on an internal 66MHz PCI bus, which is isolated from the external PCI bus. The software running on the internal MIPS is in control of the IDE, USB, LPC, and DAIO peripherals, or optionally these peripherals are not used at all in the system.	During PCI enumeration, the Xilleon 220 appears as a "single-function" device, namely a 'Multimedia Subsystem'. The registers of the internal PCI Peripherals are not visible to the external host CPU.

The operating mode is set by strap inputs to the Xilleon 220, which are sampled at the time when Reset to the chip is de-asserted. Once the mode is set at reset, it is not possible to change it, unless another reset is applied. Refer to *“Configuration/Straps” on page 9-24* for details about power-up straps.

2.4 Address Map

2.4.1 Programmable Address Apertures

Within the Xilleon 220, requests are routed from an initiator to a target by address-space apertures. Each aperture maps a region of addresses to one of the address spaces. From the point of view of each initiator, the Xilleon 220 memory map is configurable. The apertures are controlled by a set of programmable registers.

Figure 2-2,"Aperture Decoding Diagram," on page 2-8 shows which apertures any given request gets routed through. For example, in PEER Mode, a request from an initiator on the PCI bus will enter from the left side of the diagram, and will be compared to the four Host Bus Interface (HBIU) aperture control registers (HBIU_MEM_BASE, HBIU_IO_BASE, HBIU_REG_BASE, and HBIU_PCU_BASE) using the respective size(s) shown. (Note that the size of the HBIU_MEM and HBIU_PCU apertures is controlled by the PCIMEMAPERSIZE strap. See the register reference or *"Configuration/Straps" on page 9-24* for details). The apertures are shown from the highest to the lowest priority from top to bottom, so in the case where apertures are programmed to overlap, the aperture with the higher priority wins. Continuing the example, suppose that the address is a hit in the HBIU_PCU aperture, the request then gets forwarded to the seven PCU apertures, where a similar comparison is performed in order to determine the final target for this request.

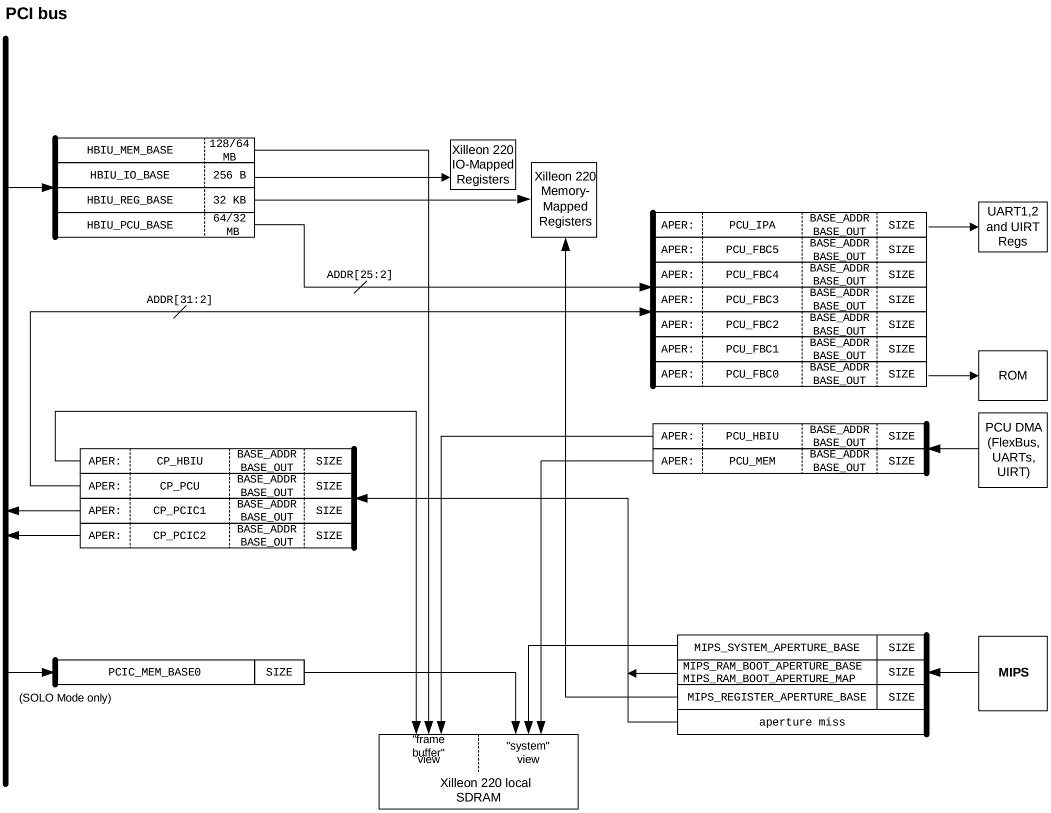

PCI bus
HBIU_MEM_BASE
128/64 MB
HBIU_IO_BASE
256 B
HBIU_REG_BASE
32 KB
HBIU_PCU_BASE
64/32 MB
Xilleon 220 IO-Mapped Registers
Xilleon 220 Memory-Mapped Registers
ADDR[25:2]
ADDR[31:2]
APER: PCU_IPA BASE_ADDR BASE_OUT SIZE
APER: PCU_FBC5 BASE_ADDR BASE_OUT SIZE
APER: PCU_FBC4 BASE_ADDR BASE_OUT SIZE
APER: PCU_FBC3 BASE_ADDR BASE_OUT SIZE
APER: PCU_FBC2 BASE_ADDR BASE_OUT SIZE
APER: PCU_FBC1 BASE_ADDR BASE_OUT SIZE
APER: PCU_FBC0 BASE_ADDR BASE_OUT SIZE
UART1,2 and UIRT Regs
ROM
APER: PCU_HBIU BASE_ADDR BASE_OUT SIZE
APER: PCU_MEM BASE_ADDR BASE_OUT SIZE
PCU DMA (FlexBus, UARTs, UIRT)
APER: CP_HBIU BASE_ADDR BASE_OUT SIZE
APER: CP_PCU BASE_ADDR BASE_OUT SIZE
APER: CP_PCIC1 BASE_ADDR BASE_OUT SIZE
APER: CP_PCIC2 BASE_ADDR BASE_OUT SIZE
PCIC_MEM_BASE0
SIZE
(SOLO Mode only)
MIPS_SYSTEM_APERTURE_BASE SIZE
MIPS_RAM_BOOT_APERTURE_BASE MIPS_RAM_BOOT_APERTURE_MAP SIZE
MIPS_REGISTER_APERTURE_BASE SIZE
aperture miss
MIPS
"frame buffer" view
"system" view
Xilleon 220 local SDRAM

Figure 2-2 Aperture Decoding Diagram

2.4.2 Example Address Map for a System

This section works through an example address map for an Xilleon 220-based system, in SOLO or PEER mode.

Note that these are not “hardware register defaults”; these register values are loaded by initialization software.

Common Address Map

This address map can be used for SOLO mode or PEER mode.

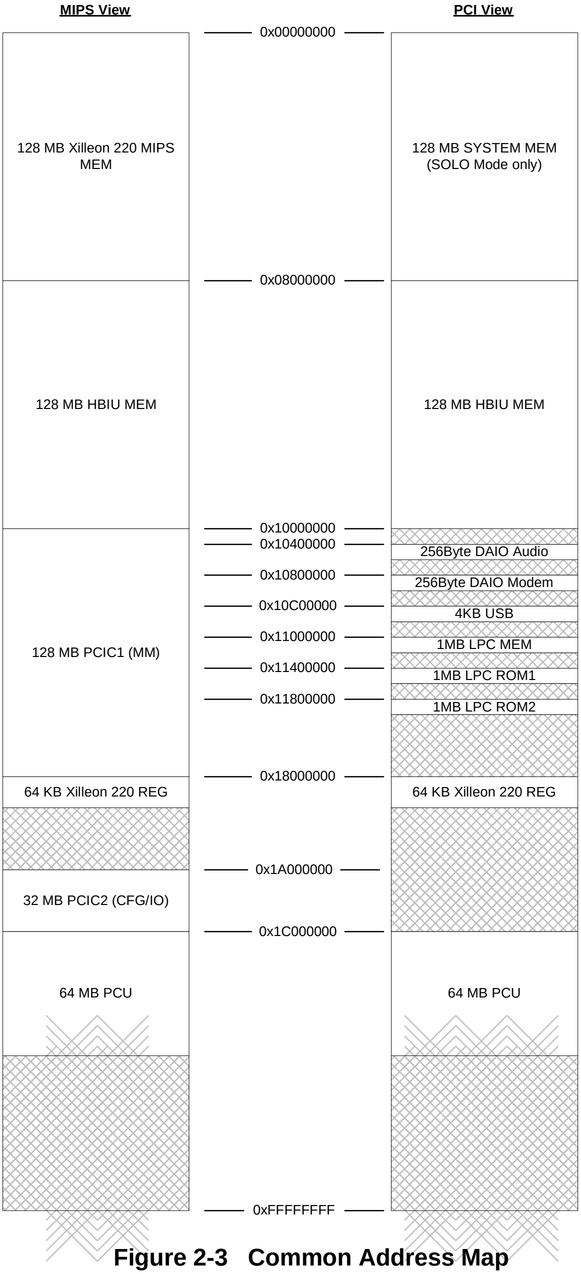


Figure 2-3 Common Address Map

Physical addresses as seen by MIPS:

(Actual addresses used by programs need to account for KSEG offset(s) and virtual page translation)

0x00000000	0x07FFFFFF	128MB	Xilleon 220 MIPS MEM
0x08000000	0x0FFFFFFF	128MB	HBIU MEM
0x10000000	0x17FFFFFF	128MB	PCI1 (MEM)
0x18000000	0x1800FFFF	64KB	Xilleon 220 REG
0x18010000	0x19FFFFFF		
0x1A000000	0x1BFFFFFF	32MB	PCI2 (shared between PCI CFG and PCI IO)
0x1C000000	0x1FFFFFFF	64MB	PCU
0x20000000	0xFFFFFFFF		

PCI addresses as seen by PCI bus masters:

Memory

0x00000000	0x07FFFFFF	128MB	System Memory
0x08000000	0x0FFFFFFF	128MB	HBIU MEM
0x10000000	0x103FFFFF		
0x10400000	0x104000FF	256B	DAIO audio
0x10400100	0x107FFFFF		
0x10800000	0x108000FF	256B	DAIO modem
0x10800100	0x10BFFFFF		
0x10C00000	0x10C00FFF	4KB	USB
0x10C01000	0x10FFFFFF		
0x11000000	0x110FFFFF	1MB	LPC MEM
0x11100000	0x113FFFFF		
0x11400000	0x114FFFFF	1MB	LPC ROM1
0x11500000	0x117FFFFF		
0x11800000	0x118FFFFF	1MB	LPC ROM2
0x11900000	0x17FFFFFF		
0x18000000	0x1800FFFF	64KB	Xilleon 220 REG
0x18010000	0x1BFFFFFF		
0x1C000000	0x1FFFFFFF	64MB	HBIU PCU
0x20000000	0xFFFFFFFF		

IO

0x0000	0x0FFF	4KB	sparse IO (LPC, IDE)
0x1000	0x1FFF		
0x2000	0x20FF	256B	IDE cmd
0x2100	0x3FFF		
0x4000	0x40FF	256B	IDE cntl
0x4100	0x5FFF		
0x6000	0x60FF	256B	IDE bm
0x6100	0x7FFF		
0x8000	0x81FF	512B	LPC IO
0x8200	0x9FFF		
0xA000	0xA0FF	256B	HBIU IO (block IO)
0xA100	0xFFFF		

CFG				**not PEER+**	**PEER+**
0x00000000	0x000000FF	256B	Xilleon 220 Function 0	HBIU	IDE
0x00000100	0x000001FF	256B	Xilleon 220 Function 1	IDE	
0x00000200	0x000002FF	256B	Xilleon 220 Function 2	USB	USB
0x00000300	0x000003FF	256B	Xilleon 220 Function 3	LPC	LPC
0x00000400	0x000004FF	256B	Xilleon 220 Function 4		
0x00000500	0x000005FF	256B	Xilleon 220 Function 5	DAIO audio	DAIO audio
0x00000600	0x000006FF	256B	Xilleon 220 Function 6	DAIO modem	DAIO modem
0x00000700	0x000007FF	256B	Xilleon 220 Function 7		

(Note: Actual CFG addresses need to account for Bus Number and Device Number offsets)

Breakdown of the PCU Peripheral Address Space:

0x00000000	0x0000FFFF	64KB	Internal Peripheral Aperture (IPA) (Internal UART, UIRT)
0x00010000	0x003FFFFF		
0x00400000	0x007FFFFF	4MB	Flexbus Device 1
0x00800000	0x00BFFFFF	4MB	Flexbus Device 2
0x00C00000	0x00FFFFFF	4MB	Flexbus Device 3
0x01000000	0x013FFFFF	4MB	Flexbus Device 4
0x01400000	0x017FFFFF	4MB	Flexbus Device 5
0x01800000	0x01FFFFFF		
0x02000000	0x03FFFFFF	32MB	Flexbus Device 0(ROM)
0x04000000	0xFFFFFFFF		

Register Settings to Achieve Common Address Map

(See Xilleon 220 Register Reference for detailed descriptions of these register fields).

- **Host Bus Interface (HBIU) Apertures**
  HBIU_MEM_BASE.MEM_BASE = 0x08000000 (0x08000000)
  HBIU_IO_BASE.IO_BASE = 0x0000A000 (0x0000A000)
  HBIU_REG_BASE.REG_BASE = 0x18000000 (0x18000000)
  HBIU_PCU_BASE.PCU_BASE = 0x1C000000 (0x1C000000)

- **Control Processor (MIPS) Apertures**
  MIPS_SYSTEM_APERTURE_BASE.APERTURE = 0x0 (128MB)
  MIPS_SYSTEM_APERTURE_BASE.BASE_ADDRESS = 0x000000 (0x00000000)

  MIPS_REGISTER_APERTURE_BASE.APERTURE = 0x0 (64KB)
  MIPS_REGISTER_APERTURE_BASE.BASE_ADDRESS = 0x180000 (0x18000000)
  (* this is also the power-up hardware default)

- **Access Router (XXR) Apertures**

APER_CP_PCIC1_ADDR.BASE_ADDR = 0x1400 (0x10000000)
APER_CP_PCIC1_ADDR.BASE_OUT = 0x1400 (0x10000000)
APER_CP_PCIC1_CNTL.APERSIZE = 0xB (64MB)
APER_CP_PCIC1_CNTL.SWAP = 0x0 (no swap)
APER_CP_PCIC1_CNTL.PCI_ATYPE = 0x2 (MEMR/MEMW)

APER_CP_PCIC2_ADDR.BASE_ADDR = 0x1A00 (0x1A000000)
APER_CP_PCIC2_ADDR.BASE_OUT = 0x0000 (0x00000000)
APER_CP_PCIC2_CNTL.APERSIZE = 0xA (32MB)
APER_CP_PCIC2_CNTL.SWAP = 0x0 (no swap)
APER_CP_PCIC2_CNTL.PCI_ATYPE = 0x3 (CFGR/CFGW) during initialization
= 0x1 (IOR/IOW) for normal operation

APER_CP_PCU_ADDR.BASE_ADDR = 0x1C00 (0x1C000000)
APER_CP_PCU_ADDR.BASE_OUT = 0x0000 (0x00000000)
APER_CP_PCU_CNTL.APERSIZE = 0xB (64MB)

APER_CP_HBIU_ADDR.BASE_ADDR = 0x0800 (0x08000000)
APER_CP_HBIU_ADDR.BASE_OUT = 0x0000 (0x00000000)
APER_CP_HBIU_CNTL.APERSIZE = 0xC (128MB)

APER_PCU_HBIU_ADDR.BASE_ADDR = 0x0800 (0x08000000)
APER_PCU_HBIU_ADDR.BASE_OUT = 0x0000 (0x00000000)
APER_PCU_HBIU_CNTL.APERSIZE = 0xC (128MB)

APER_PCU_MEM_ADDR.BASE_ADDR = 0x0000 (0x00000000)
APER_PCU_MEM_ADDR.BASE_OUT = 0x0000 (0x00000000)
APER_PCU_MEM_CNTL.APERSIZE = 0xC (128MB)

- **Host Bridge (PCIC) Apertures**

PCIC_MEM_BASE0.MEM_APER_SIZE0 = 0x0 (128MB)
PCIC_MEM_BASE0.MEM_BASE0 = 0x000 (0x00000000)

- **Peripheral Controller Unit (PCU) Apertures**

  APER_PCU_IPA_ADDR.BASE_ADDR = 0x0000 (0x00000000)
  APER_PCU_IPA_ADDR.BASE_OUT = 0x0000 (0x00000000)
  APER_PCU_IPA_CNTL.APERSIZE = 0x1 (64KB)

  APER_PCU_FBC0_ADDR.BASE_ADDR = 0x0200 (0x02000000)
  APER_PCU_FBC0_ADDR.BASE_OUT = 0x0000 (0x00000000)
  APER_PCU_FBC0_CNTL.APERSIZE = 0xA (32MB)

  APER_PCU_FBC1_ADDR.BASE_ADDR = 0x0040 (0x00400000)
  APER_PCU_FBC1_ADDR.BASE_OUT = 0x0000 (0x00000000)
  APER_PCU_FBC1_CNTL.APERSIZE = 0x7 (4MB)

  APER_PCU_FBC2_ADDR.BASE_ADDR = 0x0080 (0x00800000)
  APER_PCU_FBC2_ADDR.BASE_OUT = 0x0000 (0x00000000)
  APER_PCU_FBC2_CNTL.APERSIZE = 0x7 (4MB)

  APER_PCU_FBC3_ADDR.BASE_ADDR = 0x00C0 (0x00C00000)
  APER_PCU_FBC3_ADDR.BASE_OUT = 0x0000 (0x00000000)
  APER_PCU_FBC3_CNTL.APERSIZE = 0x7 (4MB)

  APER_PCU_FBC4_ADDR.BASE_ADDR = 0x0100 (0x01000000)
  APER_PCU_FBC4_ADDR.BASE_OUT = 0x0000 (0x00000000)
  APER_PCU_FBC4_CNTL.APERSIZE = 0x7 (4MB)

  APER_PCU_FBC5_ADDR.BASE_ADDR = 0x0140 (0x01400000)
  APER_PCU_FBC5_ADDR.BASE_OUT = 0x0000 (0x00000000)
  APER_PCU_FBC5_CNTL.APERSIZE = 0x7 (4MB)

2.5 Reset

A low voltage level on the RESETb input pin puts the Xilleon 220 into the reset state. In system configurations that utilize the Xilleon 220 in PEER or PEERPlus mode, this signal normally comes from the PCI RST# signal for this PCI "slot". In system configurations that utilize the Xilleon 220 in SOLO mode, this signal normally comes from board-level reset-generation logic.

During reset, all of the internal clocks of the Xilleon 220 are driven with a single reference clock. All internal registers that have power-up default values are loaded with these values.

The Xilleon 220 drives the RESETOUTb output pin low as soon as it sees RESETb asserted; and when RESETb deasserts, the Xilleon 220 will wait sixteen (16) clocks before driving the RESETOUTb pin high. The RESETOUTb signal can be used to reset external FlexBus devices, external I^2C devices, and in SOLO configurations, any external PCI devices.

If the BOOTMIPSONHARDRESET strap is a '1', then the internal MIPS processor will come out of reset 32 clocks after the RESETb input pin deasserts.

2.6 Boot Sequence

2.6.1 Boot Sequence for SOLO Configurations

1 RESETb asserted by external reset logic

2 RESETb deasserted by external reset logic

3 Xilleon 220 captures strap configuration (BOOTMIPSONHARDRESET='1')

4 Internal MIPS core (CPU) begins fetching instructions

- **a** CPU issues instruction read at initial boot vector (IBV) instruction address
- **b** The IBV "misses" all of the MIPS_* apertures, and gets forwarded to the CP_* aperture group (see *Figure 2-2,“Aperture Decoding Diagram,” on page 2-8*).
- **c** The CP_PCU aperture identifies IBV address as a PCU address and passes request to PCU.
- **d** ROM of FlexBus is accessed through FlexBus ROM aperture

5 CPU receives instruction, executes it and continues fetch/execute sequence.

6 CPU performs Power-On-Self-Test (POST)

7 CPU enumerates and configures peripheral devices

8 CPU loads operating system

9 Operating system loads drivers

2.6.2 Boot Sequence for PEER and PEERPlus Configurations

An external processor, if used in a system, is connected to the Xilleon 220 via the PCI bus, and the Xilleon 220 is put into PEER or PEERPlus mode.

If the system designer so desires, a ROM, which is connected to the Xilleon 220’s FlexBus, can be used as the boot ROM for the system processor.

In PEER mode, the Xilleon 220 presents itself to the PCI bus as a multi-function device, with the functions being:

0: HBIU (Multimedia Subsystem)
1: IDE
2: USB
3: LPC
5: DAIO Audio
6: DAIO Modem

In PEERPlus mode, the Xilleon 220 presents itself to the PCI bus as a single-function device, with the function being:
0: HBIU (Multimedia Subsystem)

Each of these functions has its own PCI config space, and each has its own set of one or more PCI base address registers that define apertures in PCI space. The HBIU in particular has three memory-space apertures, defined by the registers HBIU_MEM_BASE, HBIU_REG_BASE, and HBIU_PCU_BASE.

The HBIU has an additional, optional aperture, which is **configured by strap values** at reset-time. This aperture can be enabled at reset-time (before PCI configuration), and allows the Xilleon 220 to respond to addresses in the boot-vector range of the external processor. Another strap (BOOT_VECTOR) tells the Xilleon 220 what type of processor it is connected to.

The table below shows the range of addresses that the aperture responds to, based on the values of the two strap fields.

Table 2-2 Configuration/Straps for HBIU Optional Aperture

BOOTROM_ ENABLE (strap) (active low)	BOOT_ VECTOR (strap)	Address Range of Boot Aperture	Description
1	xx	Not Applicable	Boot Aperture Disabled
0	00	0x1FC00000 – 0x1FFFFFFF (4 MB)	MIPS CPU
0	01	0x000E0000 – 0x000FFFFF (128 KB) or 0xFFFE0000 – 0xFFFFFFFF (128 KB)	x86-Family CPU (BootROM is aliased to both of these apertures)
0	10	0xA0000000 – 0xA03FFFFF (4 MB)	SH-Family CPU
0	11	0x00000000 – 0x003FFFFF (4 MB)	StrongARM/ARM-Family CPU

There is a one-bit programmable register field HBIU_BOOTAPER.HBIU_BOOTAPER_EN, which gets initialized with the value of the BOOTROM_ENABLE strap, and is a read/write register field. This field controls whether this boot aperture is enabled or not. The fact that it is writable allows the host to disable the aperture during its initialization, if it so desires.

All flash and disk-on-chip boot devices are treated equally by the FlexBus Controller (FBC). FBC aperture 0 defaults to a common slow access cycle.

SOLO modes cannot be ROM-less. PEER modes boot the external processor from ROM, then the external processor initializes the internal processor. The internal processor can be booted from ROM or from the video memory, depending on the setting of the MIPS_RAM_BOOT_APERTURE.MIPS_RAM_BOOT_APERTURE_EN register bit that is configured by the external processor.

2.7 Interrupt

2.7.1 Introduction

This section describes the overall interrupt structure within the Xilleon 220. It describes how the interrupts within each functional block behave, and how they are routed at the top-level to the external host CPU of the internal MIPS processor. It also describes how an interrupt handler determines the source of an interrupt and how the handler can clear the interrupt after servicing.

Any interrupt from an internal functional block of Xilleon 220 can be routed to either the external CPU or the internal MIPS processor. There are lots of potential interrupt sources to be serviced. Each interrupt source is serviced by either the internal MIPS processor, or the external CPU, but never both. Since the number of interrupt sources is greater than the number of interrupt request inputs to the processors, the interrupt sources are grouped and arranged hierarchically. To preserve the software prioritization of the MIPS architecture, the number of interrupt groups is matched to the number of interrupt request inputs to the internal MIPS core, namely six.

There are six interrupt request inputs to the internal MIPS core. Each interrupt group has an interrupt status register and an interrupt mask register covering all the interrupt sources in the group. The Xilleon 220 interrupts are grouped as follows:

- The MIPS CP0 Timer/Counter interrupts.
- Internal Peripheral Controller interrupts.
- External PCI interrupts.
- Graphics and MPEG housekeeping interrupts.
- General purpose external interrupts.
- Spares.

There are four PCI interrupt request pins. The external PCI requests are isolated from the internal PCI interrupt requests as a separate group. This makes the IDE, USB, LPC and DAIO controllers appear as an alternate PCI interrupt group to the internal MIPS core.

The graphics, MPEG and HDTV1 legacy interrupts are grouped together. The GEN_INT_CNTL (General Interrupt Control) and GEN_INT_STATUS (General Interrupt Status) registers provide compatibility with HDTV1 where possible.

Interrupt signals from devices residing on the external FlexBus should be wired to GPIO input pins at the board-level, and would therefore appear in the "General Purpose" group.

Xilleon 220 provides both processors with a low-latency, convenient place to find the cause of an interrupt. Each processor has access to a common interrupt status register, and, depending on which bits are asserted, the processor may need to read additional register(s) before it can resolve the exact cause of an interrupt.

2.7.2 Block Level Interrupt Structure

Each functional block in the Xilleon 220 follows a simple convention for its register model for interrupt support. There are two basic types of registers: the block level interrupt status register, and the block level interrupt control (enable/mask) register. These two registers control an interrupt signal out of the functional block.

There can be multiple interrupt sources within a functional block. Each interrupt source has a status bit in the block level interrupt status register(s). When a source generates an interrupt, the corresponding interrupt status register bit is set to '1'. When the interrupt gets serviced, the CPU will write '1' (acknowledge) to the specific interrupt status bit in the status register to clear the interrupt. After such a write, the interrupt status bit becomes '0'. For each interrupt status register bit, there is a corresponding enable (mask) bit in the block level interrupt control (enable/mask) register. The interrupt enable bit controls whether a specific interrupt source should be allowed to go to the chip level. Setting the enable bit to '1' permits the specific interrupt to go to the chip level, while setting the enable bit to '0' masks the interrupt. The block interrupt output is an 'OR' of all of the status bits in the interrupt status register, after being masked by each individual enable bit. *Figure 2-4* shows an example of the interrupt structure in the Xilleon 220 internal audio block.

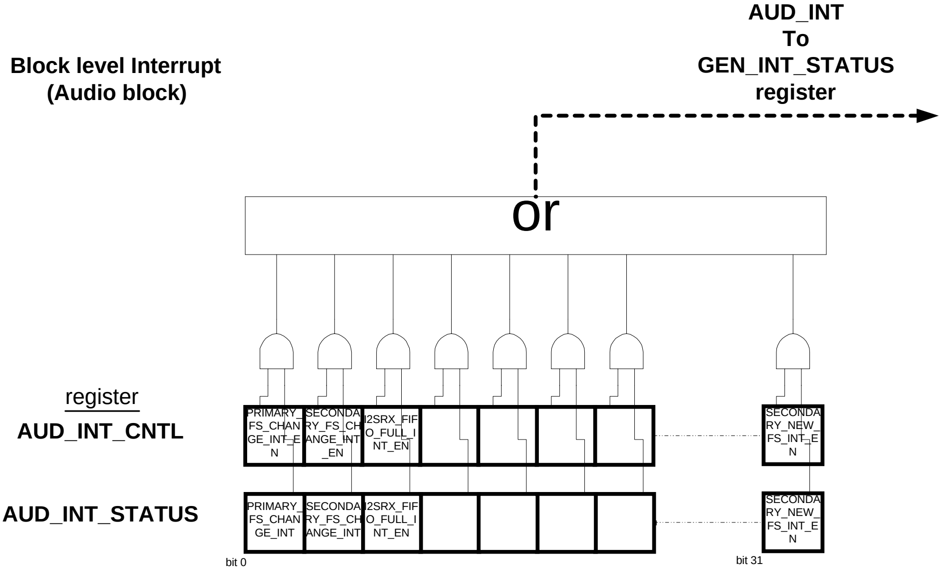


Figure 2-4 Interrupt Structure within a Functional Block

Even if the specific interrupt source is masked, software can still poll the specific interrupt status bit to see whether the event has occurred (this mode of operation is called polling). Software is still able to clear the interrupt status bit by writing '1' to the specific interrupt status bit after servicing it.

A second interrupt control (enable/mask) register may exist if a functional block wants to route some of its interrupts to MIPS, and some of its interrupt to the external CPU. One interrupt control register controls the enabling of interrupt to MIPS, and the other controls the enabling of interrupt to the external CPU. In this case, two interrupt output lines go out of the block: one goes to the external CPU, while the other goes to the external MIPS. Each interrupt line is the 'OR' of all the status bits in the interrupt status register, after being masked by each individual enable bit. *Figure 2-5,“Block Level Interrupt Structure,” on page 2-19* shows a Xilleon 220 internal functional block, Transport Demux or TPS, with two separate control registers.

In order to prevent a single interrupt source from generating an interrupt to both the external CPU and the internal MIPS, apply the following rule: never set both control registers for the same status bit to '1'. If both enable bits for the same interrupt are set, both the external CPU and the internal MIPS will see the same interrupt at the same time. If both CPUs attempt to service the same interrupt at the same time, a race condition will occur, and the system behavior will become unpredictable.

The block with a single interrupt output line will have all its interrupts routed to either the external CPU or the MIPS at the chip level. It is not possible to route part of the interrupts source to the CPU while attempting to route the other portion of the interrupts source to the MIPS for these blocks. For most of the Xilleon 220 internal function blocks, it is adequate for a single CPU to service all of its interrupts. Only a small portion of the functional blocks needs to route some of its interrupts to one processor, and some to the other.

Block level Interrupt

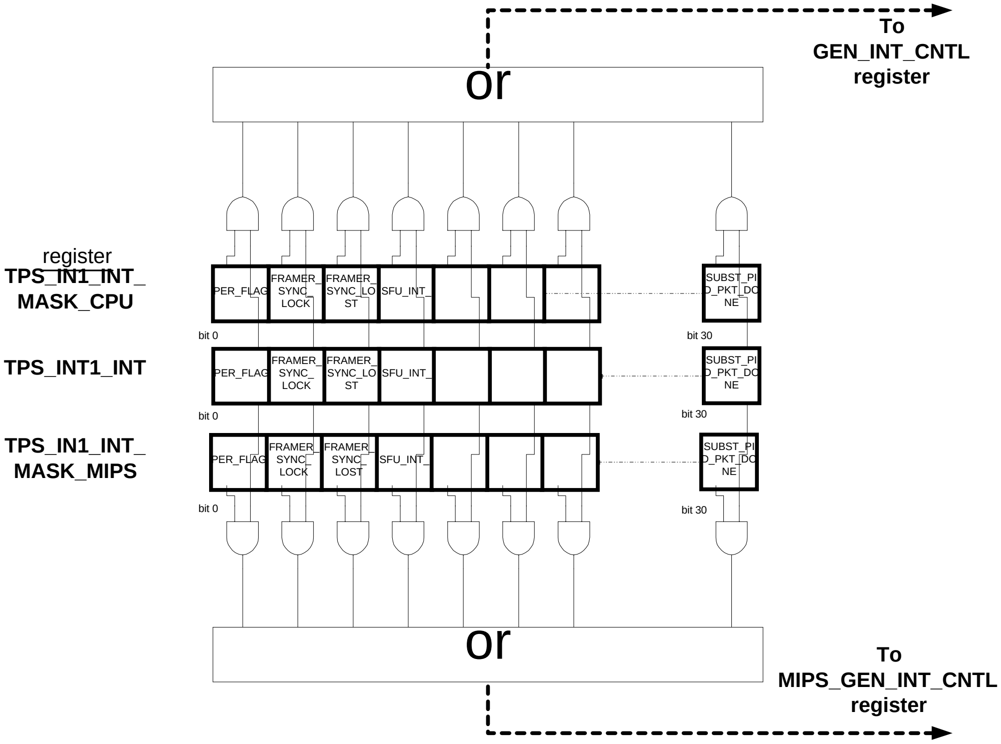


Figure 2-5 Block Level Interrupt Structure

2.7.3 Chip Level Interrupt Structure

Each Xilleon 220 internal functional block outputs its interrupt signal to the chip-level. The chip-level interrupt control logic is responsible for routing the block level interrupts to either the external CPU or the internal MIPS processor. In Xilleon 220, block interrupts are divided into four groups according to their functionality. They are:

- Firmware (FW) group
- Peripheral Controller Unit (PCU) group
- Southbridge (SB) group
- External PCI group.

Two registers control each group of interrupts: one is a status register that contains individual interrupt bits, and the other is a control register that controls the enabling (masking) of individual interrupts. A second set of control registers may exist for a group if the specific group of interrupts can be routed to either the external CPU or the internal MIPS.

The Firmware, PCU, and Southbridge groups can be routed to either the external CPU or the internal MIPS. The fourth group, the external PCI group, can only be routed to the internal MIPS; therefore, there is only a single control register for this group.

Each group of the Xilleon 220 top level interrupts has a unique interrupt line hook-up to the Xilleon 220 internal MIPS. The mapping from the Xilleon 220 internal interrupt line to the MIPS interrupts input port is shown in *Table 2-3*.

Table 2-3 Xilleon 220 Internal Interrupt Line Mapping to the MIPS Interrupts Input Port

MIPS Input Port	Xilleon 220 Internal Interrupt Line
INT 1	PCU_interrupt
INT 2	Firmware_interrupt
INT 3	External_PCI_interrupt
INT 4	Southbridge_interrpt

Unlike the internal MIPs, which has multiple interrupt inputs, there is only a single interrupt signal going out to the external CPU. Therefore, all of the three groups of interrupts that are capable of going to the external CPU are merged into a single interrupt output line to the output pin of the chip. This chip interrupt output pin is mapped to the INTA# of the external PCI bus INT lines. To the external CPU, Xilleon 220 is a PCI device that can generate interrupts on the PCI INTA# line. This means that all the Xilleon 220 internal function blocks will have their interrupt configuration register hard-wired to indicate that they use PCI INTA#. The internal functional blocks that contain PCI configuration registers include HBIU, IDE, and DAIO.

To the internal MIPS, the interrupt configuration follows *Table 2-3,"Xilleon 220 Internal Interrupt Line Mapping to the MIPS Interrupts Input Port," on page 2-20*. The PCI interrupt configuration register within each functional block has no meaning here. These PCI interrupt configuration registers are only used to indicate PCI interrupt configuration to the external PCI bus.

The following are detailed descriptions of the Xilleon 220 chip level interrupt groups.

2.7.3.1 Firmware Interrupt

The firmware interrupt group is controlled by the GEN_INT_STATUS register and GEN_INT_CNTL and MIPS_GEN_INT_CNTL register. *Figure 2-6,"Chip Level Interrupt Control Logic for Firmware Interrupt Group," on page 2-22* shows the firmware interrupts group logic. The two control registers, GEN_INT_CNT and MIPS_GEN_INTCNTL, are the enable/mask register for the interrupt to go to the external CPU or the internal MIPS.

Each functional block that belongs to this interrupt group has a bit in the GEN_INT_STATUS register that represents the interrupt. The corresponding enable/mask bit in the GEN_INT_CNTL and MIPS_GEN_INT_CNTL can be programmed to enable the interrupt to the external CPU or the internal MIPS. In order to prevent a single interrupt source from generating an interrupt to both the external CPU and the internal MIPS at the same time, only one of the interrupt enable bit for a single interrupt status bit can be set to '1' at a time.

In case a functional block has two interrupt line outputs, one for the external CPU and one for the internal MIPS, both bits will show up in the GEN_INT_STATUS register: one that represents the MIPS interrupt, and one that represents the external CPU interrupt. For the bit that represents the MIPS interrupt, the corresponding enable bit in the GEN_INT_CNTL will be permanently disabled. This is the case because the interrupt status bit itself already indicates that it wants to go to the MIPS and not to the external CPU. Similarly, for the status bit that represents the external CPU interrupt, the corresponding enable bit in the MIPS_GEN_INT_CNTL will be permanently disabled. This is the case because the interrupt status bit itself already indicates that it wants to go to the external CPU not to the internal MIPS.

The Firmware interrupt control and status register also contains the logic for merging three groups of the Xilleon 220 interrupts into a single interrupt line to connect to the PCI INTA# line. The two bits that represent the two other chip level interrupts group, PCU and Southbridge, are contained in the GEN_INT_STATUS register. The GEN_INT_CNTL register contains the corresponding enable/mask bits for these two groups.

Two sets of merged interrupt lines are output from the Firmware interrupt control logic. On the one side is the single interrupt line that hooks up to the external CPU. On the other side is the firmware interrupt line that hooks up to the MIPS INT2 interrupt port.

The reason for placing, in the firmware interrupt, the merging logic of the three groups of interrupts to the external CPU, is for better performance. This arrangement allows the external CPU to decide which group of interrupts are causing the interrupt. This is done by reading a single Xilleon 220 internal interrupt register. Otherwise, three individual group status registers have to be read in order to decide which interrupt belongs to which group.

The highest bit, bit 31, in the GEN_INT_CNTL register contains the master enable bit for the chip interrupt. When this bit is programmed to '1', the Xilleon 220 chip is allowed to generate interrupt on the external PCI bus's INTA# line to the external CPU. When this bit is programmed to '0', the external CPU will not see the Xilleon 220 generate interrupt on the PCI bus. However, the external CPU can still poll the Xilleon 220 internal GEN_INT_STATUS register to figure out which internal functional block wants to generate the interrupt.

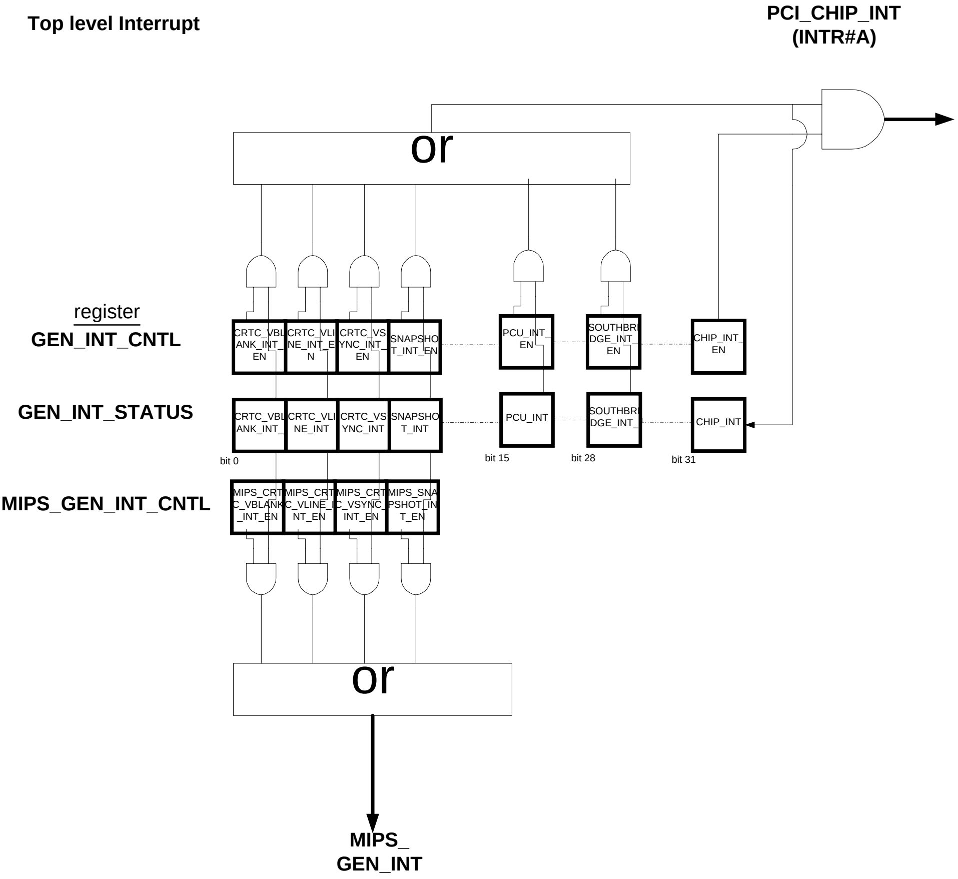


Figure 2-6 Chip Level Interrupt Control Logic for Firmware Interrupt Group

2.7.3.2 PCU Interrupt Group

PCU stands for "peripheral controller unit". The PCU group of interrupts contains interrupts from the Xilleon 220 internal peripheral controller blocks. These interrupts control such peripherals as I^{2}C, Smart card, Flex bus and Timer/RTC. Two sets of control (enable/mask) registers exist for this group of interrupt, one for the MIPS and one for the external CPU. The outputs from the PCU interrupt control logic are two interrupt lines. One output is the PCU interrupt to MIPS, and the other output is the PCU interrupt to the external CPU. The PCU interrupt to MIPS is hooked up to the MIPS INT1 port. The PCU interrupt to the external CPU is merged with two other interrupt groups to form a single interrupt output to the external CPU. *Figure 2-7,"Control Logic for PCU Interrupt," on page 2-23* is block diagram of the PCU interrupt group control logic.

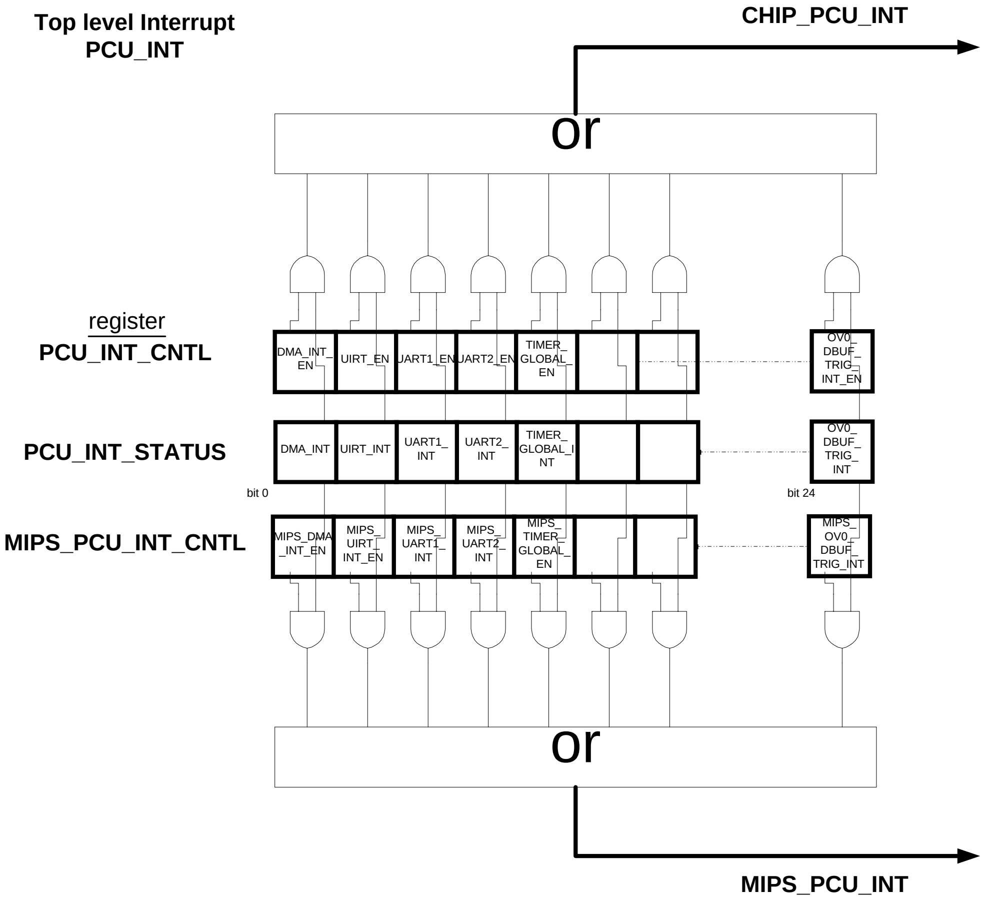


Figure 2-7 Control Logic for PCU Interrupt

2.7.3.3 South Bridge Interrupt

The Southbridge group of interrupts contains interrupts from the Xilleon 220 internal functional blocks. These interrupts control such peripherals as IDE, USB, LPC and DAIO (AC97-audio). Two sets of control (enable/mask) registers exist: one for the MIPS, and one for the external CPU. The outputs are two interrupt lines: one is the Southbridge interrupt to the MIPS, and the other is the Southbridge interrupt to the external CPU. The Southbridge interrupt to the MIPS is hooked up to the MIPS INT4 port. The Southbridge interrupt to the external CPU is merged with two other interrupt groups to form a single interrupt output to the external CPU. *Figure 2-8,“Chip Level Interrupt Logic for South Bridge Group,” on page 2-24* is block diagram of the interrupt logic for the South Bridge group.

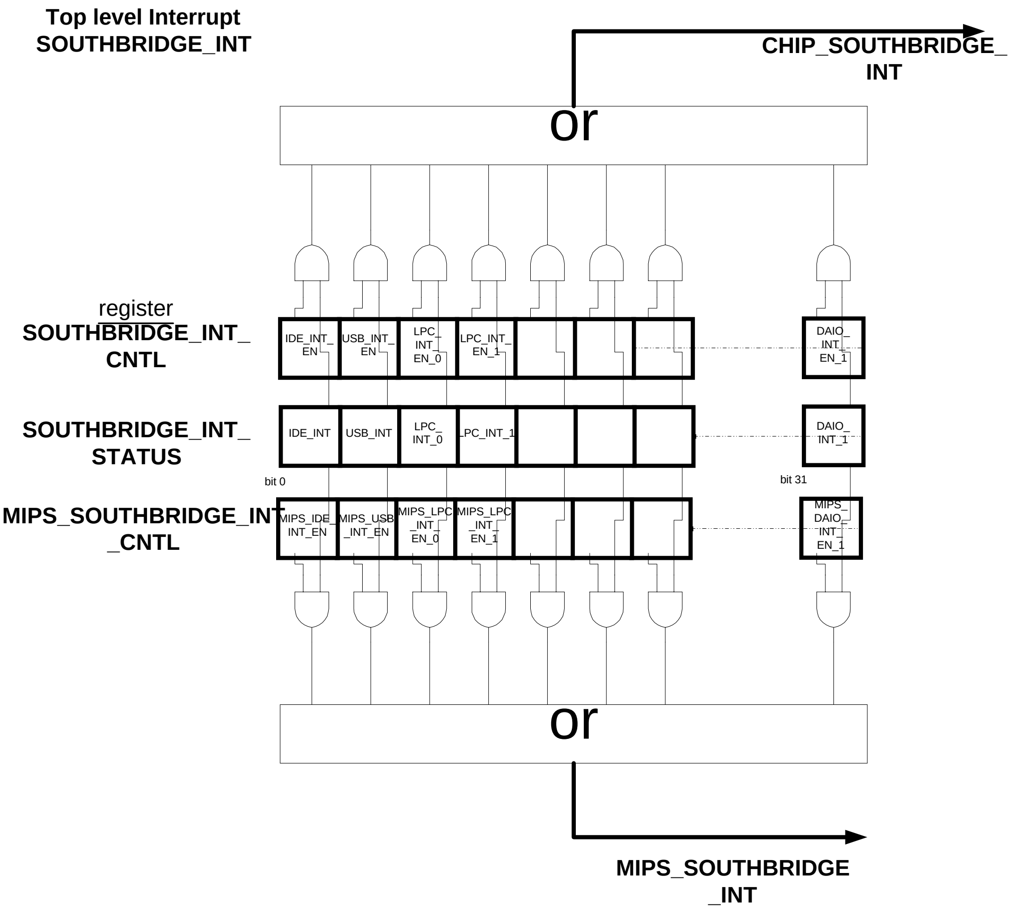


Figure 2-8 Chip Level Interrupt Logic for South Bridge Group

2.7.3.4 PCI Interrupt Group

The PCI interrupt group contains four PCI interrupt lines: INTA#, INTB#, INTC#, INTD#. These lines are defined by the PCI specs. They are directly coming in from four PCI interrupt pins to the Xilleon 220 chip. They are interrupts from PCI devices external to the Xilleon 220 chip. These interrupt lines are only used when Xilleon 220 acts as a PCI controller. The Xilleon 220 operates in solo mode when it acts as a PCI controller. When in peer mode, Xilleon 220 does not provide PCI controller functionalities; therefore, these four PCI interrupt lines are not used.

The four external PCI interrupts are merged into a single one by the PCI interrupt group control logic. The resulting single interrupt line is connected to the MIPS INT3 port. *Figure 2-9,"Chip Level External PCI Interrupt Control Registers and Status Registers," on page 2-26* shows the PCI interrupt group logic.

There is no corresponding interrupt that goes out to the external CPU for this interrupt group. The reason is that when there is an external CPU, the external CPU controls the PCI bus and all the PCI interrupts by-pass the Xilleon 220 chip and are routed to the external CPU. The mode of operation in such a case is called peer mode. In such a mode, all four bits MIPS_PCI_INT_CNTL should be set to disable, since no interrupt lines will be hooked up.

The INTA# pin on the Xilleon 220 chip has two functions depending on which mode Xilleon 220 is working on. When Xilleon 220 is working in solo mode, the INTA# pin is set only to accept external PCI INTA#. When Xilleon 220 is working in peer mode, the INTA# pin is set to output only. In such a case, the Xilleon 220 chip interrupt is sent out from this pin onto the PCI bus. *Figure 2-9,"Chip Level External PCI Interrupt Control Registers and Status Registers," on page 2-26* shows the chip level external PCI interrupt control registers and status registers. The output is MIPS_PCI_INT that hooks up to INT3 of the MIPS interrupt input port.

Top level Interrupt
EXT_PCI_INT

No PCI interrupt will be generated to the host CPU. We are interested in PCI interrupt in only SOLO mode

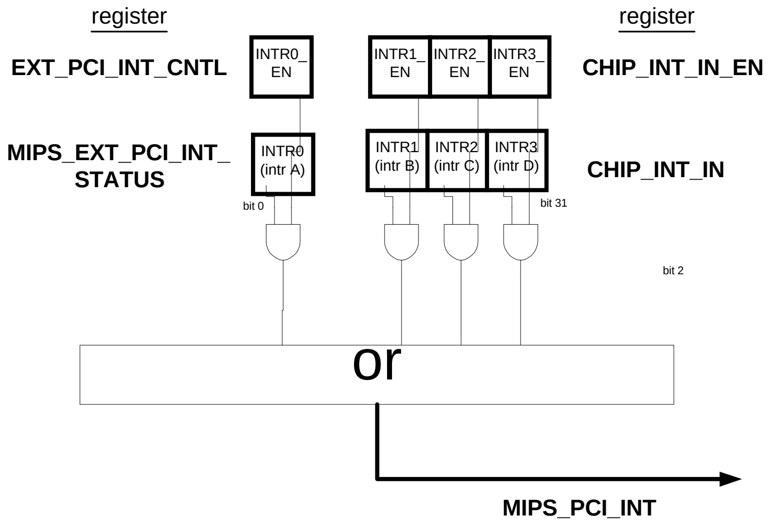


Figure 2-9 Chip Level External PCI Interrupt Control Registers and Status Registers

2.7.3.5 MIPS NMI Interrupt

MIPS can accept an NMI interrupt. It is generally used for system level service such as system error correction and initialization. However, in addition to system level error interrupt, Xilleon 220 can also route every one of the four group chip level interrupts to NMI. The MIPS_NMI_INT_CNTL register supports this feature. It has a similar bit-layout to the GEN_INT_CNTL (and MIPS_GEN_INT_CNTL) register. This register provides the enable/mask bit for each of the GEN_INT_STATUS interrupt bits to go to the MIPS NMI pin. Bit 31 of MIPS_NMI_INT_CNTL is the master enable bit for the MIPS NMI interrupt. *Figure 2-10,"MIPS NMI Interrupt Generation," on page 2-27* is the block diagram that shows the MIPS NMI interrupt control logic.

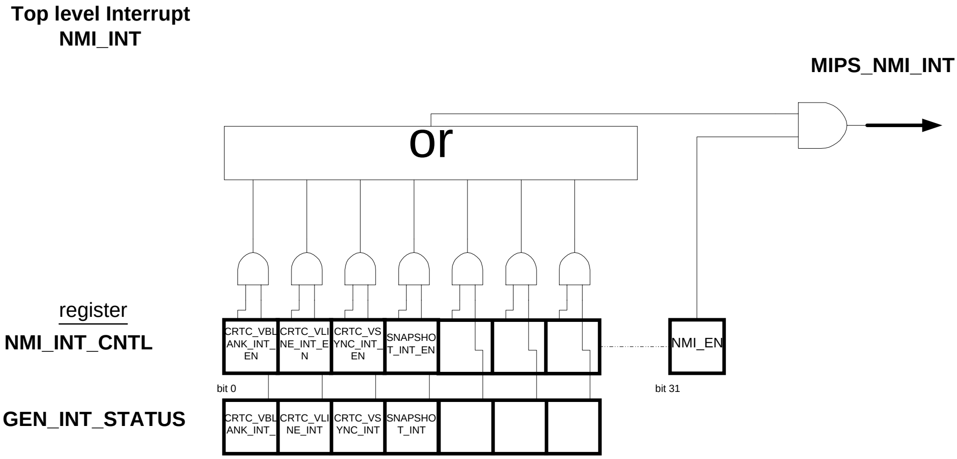


Figure 2-10 MIPS NMI Interrupt Generation

2.7.4 MIPS Interrupt Structure

The MIPS processor can accept up to six interrupt lines: INT0-INT5. There are two MIPS internal (CP0) registers that control the interrupt: the *cause* register and the *status* register. The cause register contains the interrupt status bit, where '0' means no interrupt, and '1' means an interrupt is pending. The status register contains the interrupt mask bit. The mask bit of '1' enables the interrupt and the mask bit of '0' disables the interrupt. Unlike the Xilleon 220 interrupt status register, there is no acknowledging bit for an interrupt in the MIPS internal interrupt registers. The MIPS requires the external interrupt source to pull the interrupt line to '0' after an interrupt get serviced. *Figure 2-11,"MIPS Interrupt Registers and Control Logic," on page 2-28* shows the MIPS internal interrupt registers and control logic.

Five MIPS interrupt lines are used in Xilleon 220. Except for the four interrupt lines discussed in the previous section, the INT5 is hooked up to the MIPS internal counter output. This connection is a MIPS convention.

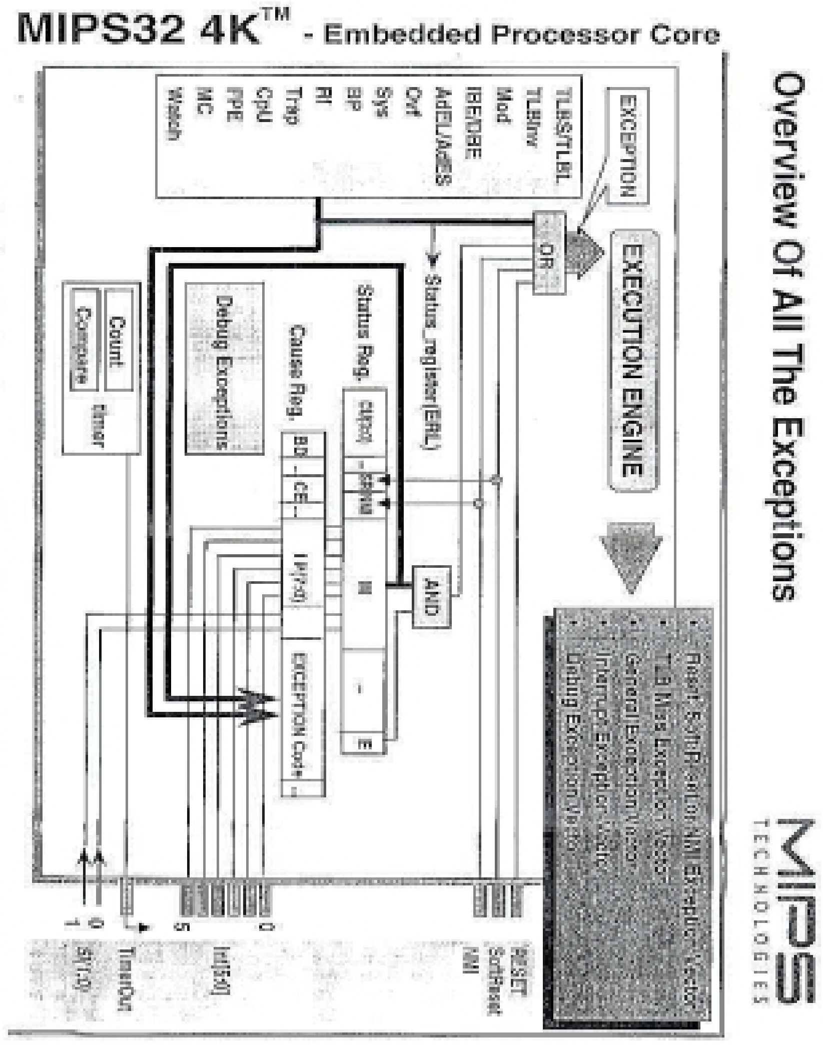


Figure 2-11 MIPS Interrupt Registers and Control Logic

2.7.5 Xilleon 220 Interrupt Service Flow

The software and hardware need to work together in order for a specific interrupt to work. The software is responsible for:

- Enabling an interrupt.
- Detecting an interrupt source.
- Disabling an interrupt upon an interrupt happening.
- Servicing the interrupt.
- Acknowledging an interrupt after servicing.
- Re-enabling the interrupt when exiting the interrupt service routine.

Each of the above steps is described below. Note that distinctions are made between the external CPU or the internal MIPS depending upon whether one, or the other, is servicing the block.

2.7.5.1 Interrupt Enable Sequence

In order to get a specific Xilleon 220 internal interrupt to the external CPU or the internal MIPS, the specific interrupt has to be enabled for that CPU. To enable an interrupt to the external CPU the following interrupt control register has to be programmed.

2.7.5.2 Interrupt for External CPU Enable

Block level

<Block>_INT_CNTL. <source>_INT_EN = 1 (specific interrupt source enable)

Group level

<Group>_INT_CNTL. <block>_INT_EN = 1 (block interrupt enable)

Chip_level

GEN_INT_CNTL. <Group>_INT_EN = 1 (group interrupt enable)

GEN_INT_CNTL.CHIP_INT_EN = 1 (chip interrupt enable)

2.7.5.3 Interrupt for Internal MIPS Enable

Block level

```
If (MIP_<block>_INT_CNTL exist) (there are two interrupt control register in the block level)
    MIPS_<block>_INT_CNTL.<source>_INT_EN = 1
Else( there is only single interrupt control register in the block level)
    <Block>_INT_CNTL.<source>_INT_EN = 1
End if
```

Group level

MIPS_<group>_INT_CNTL.MIPS_<block>_INT_EN = 1 (enable interrupt for the block)

MIPS internal

Status.IM(<group>) = 1(enable the interrupt for the particular group)

2.7.5.4 Interrupt Detection Sequence

When an interrupt is generated to the external CPU or the internal MIPS, the CPU will execute an interrupt service routine to respond to the specific interrupt. Before executing any particular interrupt routine, the software has to determine the source of the interrupt. Although the interrupt generation is a bottom up process, the CPU that determines the specific interrupt source uses a top down approach. Both the external CPU and the internal MIPS use a similar process to exactly determine the interrupt source.

2.7.5.5 Interrupt Detection by External CPU

The external CPU starts by reading the chip level interrupt status register and the interrupt control register. From the chip level the software will traverse three levels down to a specific block by reading each level of the interrupt status and the interrupt control register to figure out the exact interrupt source.

The procedure is as follows:

```
RegisterRead GEN_INT_STATUS
RegisterRead GEN_INT_CNTL
Case GEN_INT_STATUS&GEN_INT_CNTL (bit “and” of gen_int_status and gen_int_cntl register to mask out not enabled interrupts).
When <group>_INT_maksed = 1 (an interrupt group is detected)
RegisterRead <group>_INT_STATUS
RegisterRead <group>_INT_CNTL
Case <group>_INT_STATUS&<GROUP>_INT_CNTL (bit “and” of <group>_int_status and <group>_int_cntl registers to mask out not enabled interrupts)
When <block>_INT =1 (a block that generate the interrupt is detected)
RegisterRead <block>_INT_STATUS
RegisterRead <block>_INT_CNTL
Case <block>_INT_STATUS&<block>_INT_CNTL
When <source>_INT = 1(the exact source of interrupt is detected)
Break
End case
End case
End case
```

2.7.5.6 Interrupt Detection by MIPS

The MIPS figures out the interrupt source by using the same top down approach used by the external CPU. It starts from the MIPS internal interrupt control registers: cause register and status register.

```
RegisterRead Cause (MIPS internal CP0 register)
RegisterRead Status (MIPS internal CP0 register)
Case Cause&Status (apply the mask in status register to cause register to mask out not enabled interrupts)
```

```
    When INT_masked(<group>) = 1( detect a group level interrupt)
        RegisterRead <group>_INT_STATUS
        RegisterRead MIPS_<group>_INT_CNTL
        Case <group>_INT_STATUS&MIPS_<group>_INT_CNTL (bit “and” of mask and status to mask out not enabled interrupts)
            When (<group>_INT_STATUS.<block>_INT = 1) (detect a block that generate interrupt)
                RegisterRead <block>_INT_STATUS
                RegisterRead MIPS_<block>_INT_CNTL
                Case <block>_INT_STATUS&MIPS_<block>_INT_CNTL
                    When (<source>_INT_Masked = 1)
                        Break (the exact interrupt source is detected)
                End case
        End case
End case
```

2.7.5.7 Interrupt Disable Procedure upon an Interrupt

When an interrupt is generated, the first step is to disable all the interrupts to the CPU. This will ensure that no interrupts are generated before the current interrupt gets serviced. There is a master enable bit for each CPU. For the external CPU, this master interrupt enable bit is located outside of the Xilleon 220 chip. It is usually located in the system interrupt controller such as 8259A in an Intel system. The software should write to that specific interrupt master enable register to turn off all the interrupts to the external CPU. The CHIP_INT_EN bit in Xilleon 220 chip’s GEN_INT_CNTL register is not a master enable bit, since another chip in the system can still generate interrupts even if this specific bit is turned off.

For interrupts going to the MIPS, the master enable bit is in the MIPS internal interrupt control register. It is bit ‘0’ of the Status register. When an MIPS interrupt gets generated, the software should turn this bit off while servicing the interrupt.

2.7.5.8 Interrupt Acknowledge Procedure

When an interrupt gets serviced by an external CPU or an internal MIPS, the final step is to write the acknowledge bit of the specific interrupt to clear the interrupt. The interrupt acknowledge bit is the same bit as the interrupt status bit at the block level. At a higher level, there will be no acknowledge bit. As a result, a single acknowledge bit write to the lowest level of the interrupt status register will clear the interrupt completely. Write this bit to ‘1’ and the interrupt will be cleared. However, not all interrupts need an acknowledge. Some of the interrupt bits will clear themselves when the interrupt gets serviced. In such a case, no acknowledge is needed.

After the interrupt gets acknowledged, the master enable bit can be turned on again to enable interrupt.

2.7.6 Summary of Interrupt Groups

2.7.6.1 Firmware Interrupt Group (FW_INT)

Table 2-4 Firmware Interrupt Group (FW_INT)

Interrupt	Description
TPS_INT_EN	Transport processor interrupt
VD0_INT_EN	Video Decoder 0 interrupt
VD1_INT_EN	Video Decoder 1 interrupt
AUDIO_INT_EN	Audio Decoder interrupt
SMCA_INT_EN	Smart card A interrupt
SMCB_INT_EN	Smart card B interrupt
SMCC_INT_EN	Smart card C interrupt
MBUS_VIP_INT_EN	VIP interrupt
MBUS_I2C_A_INT_EN	I^2C interrupt
MBUS_I2C_B_INT_EN	I^2C interrupt
SI_MIPS_INT_EN	Stream interface interrupt
HM_COM_INT_EN	

2.7.6.2 Internal Peripheral Controller Interrupt Group (PCU_INT)

Table 2-5 Internal Peripheral Controller Interrupt Group (PCU_INT)

Interrupt	Description
PCU_INT_CNTL	
UIRT_EN	IR interrupt
UART1_EN	Serial 1 interrupt
UART2_EN	Serial 2 interrupt
TIMER_GLOBAL_INT_EN	Timer interrupt
RTC_GLOBAL_INT_EN	Realtime clock interrupt
FLEXBUS_INT_EN	Flexbus interrupt
DMA_INT_EN	PCU DMA interrupt

2.7.6.3 Internal PCI Interrupt Group (SB_INT)

Table 2-6 Internal PCI Interrupt Group (SB_INT)

Interrupt	Description
SB_INT_CNTL	
IDE_INT_EN	IDE interrupt
USB_INT_EN_0	USB interrupt 0
USB_INT_EN_1	USB interrupt 1
USB_INT_EN_2	USB interrupt 2
LPC_INT_EN_0	LPC interrupt 0

Table 2-6 Internal PCI Interrupt Group (SB_INT) (Continued)

Interrupt	Description
LPC_INT_EN_1	LPC interrupt 1
LPC_INT_EN_2	LPC interrupt 2
LPC_INT_EN_3	LPC interrupt 3
DAIO_INT_EN_0	DAIO audio interrupt
DAIO_INT_EN_1	DAIO modem interrupt

2.7.6.4 External PCI Interrupt Group (GEN_INT)

Table 2-7 External PCI Interrupt Group (GEN_INT)

Interrupt	Description
CRTC_VBLANK_INT CRTC	vertical blank interrupt enable
CRTC_VLINE_INT CRTC	vertical line interrupt enable
CRTC_VSYNC_INT CRTC	vertical sync interrupt enable
SNAPSHOT_INT	snapshot interrupt enable
SI_CPU_INT	stream interface interrupt to external host
TPS_INT	transport processor interrupt
I2S_FIFO_DATA_REQ_INT	I2S interrupt
BUSMASTER_EOL_INT	Bus master end-of-system-list interrupt
I2C0_INT	I^2C interrupt 0
I2C1_INT	I^2C interrupt 1
GUI_IDLE_INT	GUI 2D/3D engine idle interrupt
INTR0	External PCI interrupt 0
INTR1	External PCI interrupt 1
INTR2	External PCI interrupt 2
PCIC_INT	Host bridge interrupt
VIPH_INT	VIP Host interrupt
HDCP_AUTHORIZED_INT	HDCP interrupt
DVI_I2C_INT	DVI interrupt
FP2_DETECT	
MH_INT	
FW_INT	Firmware interrupt
PCU_INT	PCU interrupt
SB_INT	Southbridge interrupt
PCI_INT_EN	PCI interrupt enable

Chapter 3
Multimedia Subsystem

3.1 Introduction

The multimedia subsystem is loosely defined as the conglomeration of the following blocks: the Stream Interface, the Transport Demultiplexer, the Conditional Access Module, the Smart Card Interfaces, the MPEG Decoder, the Audio Subsystem, the Display Subsytem, the 2D Engine and the 3D Engine.

Together these blocks provide all the possible video/audio functionalities required in a modern HDTV set-top box, and in addition, the capability of Smart Card transactions, and 3D gaming. For details about a block of interest, use the following hot-links:

- *"The Stream Interface" on page 3-2*
- *"Transport Demultiplexer" on page 3-16*
- *"Conditional Access Module" on page 3-46*
- *"Smart Card Interfaces" on page 3-49*
- *"MPEG Decoder" on page 3-57*
- *"The Audio Subsystem" on page 3-67*
- *"Display Subsystem" on page 3-78*
- *"2D/3D Engines" on page 3-96*

3.2 The Stream Interface

3.2.1 Introduction

The stream interface (SI) is a centralized DMA engine which provides streaming data support between a hardware client, such as the transport or the video decoder, and the frame buffer or PCI memory space. The SI provides read/write pointer interrupt, scatter-gather, and fifo/buffer support to the clients. Hardware clients are mapped into a register port which is either the source or the destination for the streaming data. Each client is programmed in the SI for a burst size which the client must be capable of receiving or sending as a unit.

3.2.2 Features

3.2.2.1 General

- 128 independently programmable hardware buffers
- Buffers operate as linear (one-shot) or circular (continuous)
- Buffers are either input or output
- Stream controllable byte swapping logic allows for rearranging data to be in the "native" format of other processing elements.
- "Scatter gather" support to allow the use of non-contiguous memory blocks
- Maps buffers into either system (PCI mapped) or frame buffer memory space.
- "Linking" of buffers allows the creation of "virtual" FIFOs.
- FIFOs maintain a depth indicator.
- Programmable data fetch size on a per buffer basis.
- ### bytes of write buffering.

3.2.2.2 Interfaces

- Transport interface allows addressing of all 128 buffers.
- Client interfaces (exception of transport input) are configured with a 1:1 mapping to stream buffer.
- Client interfaces have a Request To Transfer handshake as well as a Transfer handshake
- Client interfaces are visible by direct host register access to allow for simplified client debugging activities.
- Two level client priority arbitration scheme where each level is treated as round robin.

3.2.2.3

-

3.2.3 Theory of Operation

The Stream Interface can be seen as a collection of 128 single-direction (either in or out) stream/video buffers, where the buffers can be either linear or circular. Each buffer is designed to minimize software interaction, and can be defined by one or more descriptor entries. Each descriptor can define an interrupt at a programmable point within it. Multiple entries are chained together and can be run as a loop. There is also an optimization mode for single descriptor entries to prevent the hardware from having to re-fetch the same descriptor.

3.2.3.1 Stream Interface Block Diagram

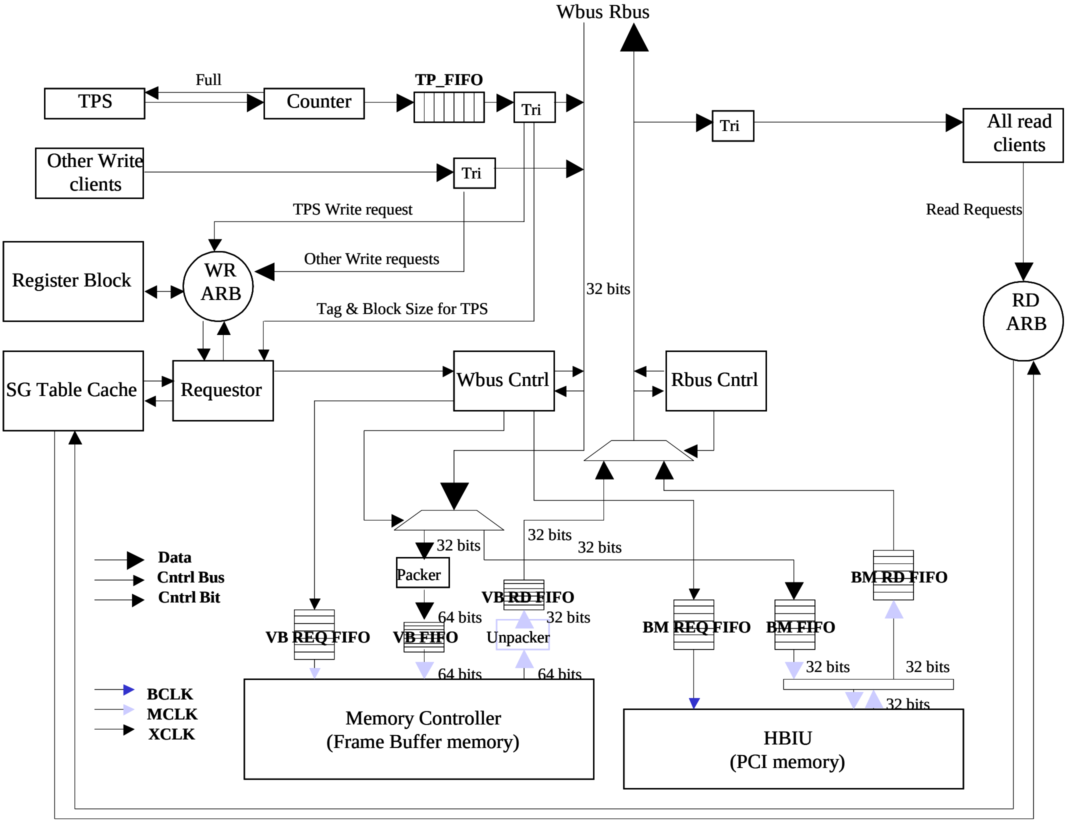


Figure 3-1 Stream Interface Block Diagram

3.2.3.2 Description of Buffers

Each buffer is identified by a unique 7-bit tag and is physically located in either the system memory or the frame buffer. The physical location of the buffer is determined by a

LOCATION bit and its respective SG-Table entry (For details on SG-Table entries, refer to section *"SG-Table Format" on page 3-10*. A bi-directional buffer can be realized by combining 2 stream buffers of opposite directions. *Table 3-1* below shows all of the possible buffer configurations in the SI.

Table 3-1 Buffer Configurations in Stream Interface

Buffer Type	Write Buffer	Read Buffer	Note
Linear Write Buffer	Linear	N/A	
Linear Read Buffer	N/A	Linear	
Circular Write Buffer	Circular	N/A	
Circular Read Buffer	N/A	Circular	
Linear Rd/Wr Buffer	Linear	Linear	The SG-Table entry list for the read and the write buffer must reference to the same physical location in the memory.
Circular Rd/Wr Buffer	Circular	Circular	The SG-Table entry list for the read and the write buffer must reference to the same physical location in the memory.

Linear Write Buffer

A linear write buffer is created by a linear link list of one or more SG-Table entries.

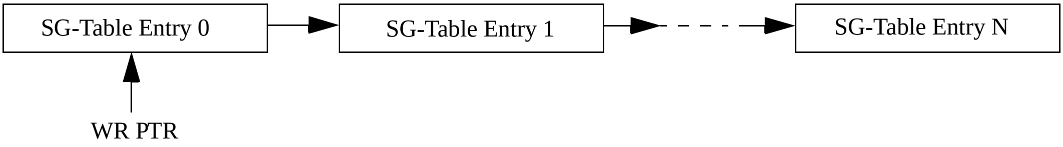


Figure 3-2 Linear Write Buffer Graphical Representation

Each contiguous block in the physical memory is represented by a SG-Table entry, which defines the block boundary, the current pointer, the pointer to the next SG-Table entry in the system memory, and status information of the buffer as a whole (for details refer to the section *"SG-Table Format" on page 3-10*).

As illustrated in *Figure 3-2*, the linear write buffer is created by linking a number of contiguous memory blocks together and using the current pointer of each block as the write pointer (WR PTR). Note that only one SG-Table entry is used at any given point in time.

Properties:

- Unidirectional -> Buffer can only be written to.
- Linear -> When the WR PTR reaches the end of the memory block defined by the last SG-Table entry in the list, the buffer will be disabled until it is re-initialized by the HOST.

Assess method:

The access method for each input source differs. Transport Parser is required to supply a tag for each byte of data. The other input sources are expected to supply the data only, since their respective tags and transfer sizes are already loaded in registers by the HOST.

Linear Read Buffer

A linear read buffer is similar to the linear write buffer except that it can only be read.

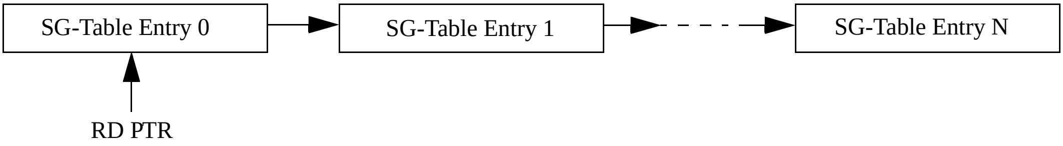


Figure 3-3 Linear Read Buffer Graphical Representation

As illustrated in *Figure 3-3*, the linear read buffer is created by linking a number of separate contiguous memory blocks together and using the current pointer of each block as the read pointer (RD PTR).

Properties:

- Unidirectional -> Buffer can only be read.
- Linear -> When the RD PTR reaches the end of the memory block defined by the last SG-Table entry in the list, the buffer will be disabled until it is re-initialized by the HOST.

Assess method:

In order to access a linear read buffer, an output source is expected to supply a request signal only, since its tags and transfer size are pre-loaded in its register by the HOST.

Circular Write Buffer

A circular write buffer is created by a linear link list of one or more SG-Table entries, with the next SG-Table pointer in the last SG-Table entry pointing to the first SG-Table pointer. *Figure 3-4* illustrates this process:

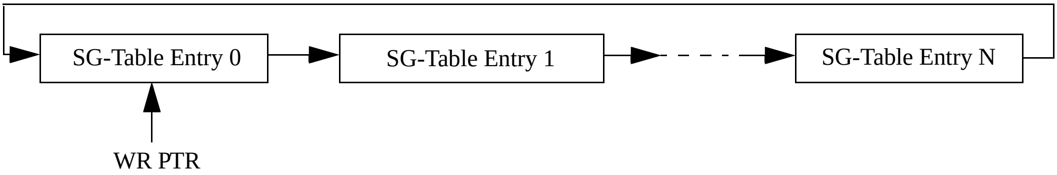


Figure 3-4 Circular Write Buffer Graphical Representation

The interesting property of this type of buffer, unlike the linear write buffer, is that the WR PTR can wrap around and start from the memory block represented by the first SG-Table entry in the list after the WR PTR gets to the end of the memory block represented by the last SG-Table entry in the list. With the property, the buffer appears "endless".

Properties:

- Unidirectional -> Buffer can only be written to.
- Circular -> After the RD PTR reaches the end of the memory block defined by the last SG-Table entry in the list, the RD PTR "wraps" to the starting location of the memory block pointed to by the first SG-Table entry in the list.

Assess method:

It is the same access method as the linear write buffer

Circular Read Buffer

A circular read buffer is similar to a circular write buffer, except that it can only be read. It is created by a linear link list of one or more SG-Table entries, with the next SG-Table pointer in the last SG-Table entry pointing to the first SG-Table pointer. *Figure 3-5* illustrates this process:

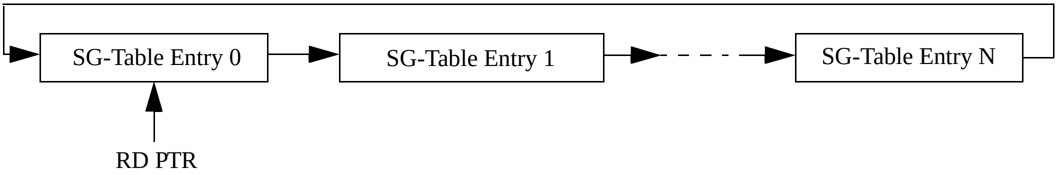


Figure 3-5 Circular Read Buffer Graphical Representation

Like the circular write buffer, the RD PTR can wrap around and start from the memory block represented by the first SG-Table entry in the list after the WR PTR gets to the end of the memory block represented by the last SG-Table entry in the list.

Properties:

- Unidirectional -> Buffer can only be read.
- Circular -> After the RD PTR reaches the end of the memory block defined by the last SG-Table entry in the list, the RD PTR “wraps” to the starting location of the memory block pointed to by the first SG-Table entry in the list.

Assess method:

It is the same access method as the linear read buffer

Linear Read/Write Buffer

A linear read/write buffer is created by 2 linear link lists of SG-Table entries. The two link lists should have the same number of SG-Table entries, and each pair of SG-Table entries in the same position from the two lists should reference to the same memory block in the physical memory. *Figure 3-6* illustrates this process:

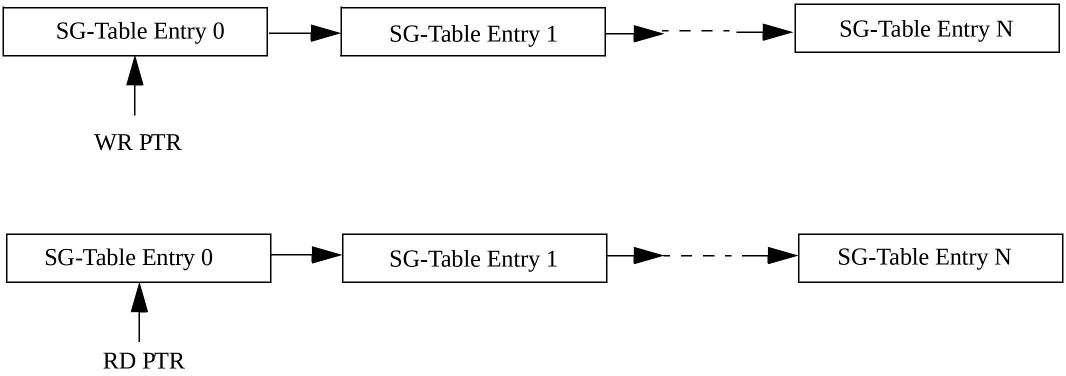


Figure 3-6 Linear Rd/Wr Buffer Graphical Representation

Since the two link lists define the same physical memory blocks linked in the same order, the two lists can be thought of as defining a single bi-directional buffer. The first link list is used to keep track of data that is written to the buffer using the WR PTR. The second link list is used to monitor the data that is read from the same buffer using the RD PTR. Please note that the WR PTR and the RD PTR can never pass each other.

When the WR PTR gets to the end of buffer, the buffer is marked as “write disable”, which rejects any further attempts to write to the buffer. When the RD PTR reaches the end, the buffer is marked as “read disable”. Accordingly, the buffer cannot be read anymore.

Properties:

- Bi-directional -> Buffer can only be written and read.
- Linear -> When the WR PTR reaches the end of the buffer, it becomes non-writeable.

Likewise, when RD PTR gets to the end of the buffer, it becomes non-readable. This status can be reset by the HOST re-initialization.

Assess method:

The access method for each input source differs. To write to the buffer, the Transport Parser is required to supply a tag for each byte of data. The other input sources are expected to supply the data only, since their respective tags and transfer sizes are already loaded in registers by the HOST. The tags used here must be associated with the link lists for write operation for the buffers. Similarly, an output source can request a read from a bi-directional buffer by supplying a read request signals only. The Stream Interface will automatically pick up from the pre-loaded register the tag that references to the link list for read operations.

Circular Read/Write Buffer

Similar to a linear read/write buffer, a circular read/write buffer is created by 2 linear link lists of SG-Table entries. The two link lists should have the same number of SG-Table entries, and each pair of SG-Table entries in the same position from the two lists should reference to the same memory block in the physical memory. *Figure 3-7* illustrates this process:

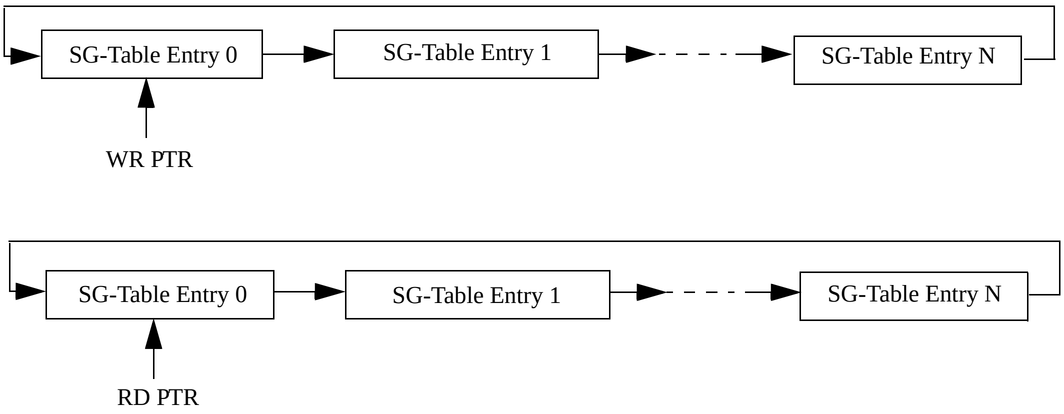


Figure 3-7 Circular Rd/Wr Buffer Graphical Representation

The only difference between a circular read/write buffer and a linear read/write buffer is that a circular read/write buffer is “endless”: both the RD PTR and WR PTR wrap around after they reach the end of the buffer.

Properties:

- Bi-directional -> Buffer can only be written and read.
- Circular -> Both pointers wrap around after they reach the end of the buffer.

Assess method:

It is the same access method as the linear read/write buffer.

3.2.3.3 Buffer Operations

Circular Buffers

On one end of the circular buffer the hardware either reads/writes while maintaining it's own pointer. When the end of the physical memory associated with the buffer is exhausted, the pointer wraps to the start of the physical memory and the process continues again. Data transfer is throttled by the associated data source/sink. When running in this mode, the software must ensure that it either stays sufficiently ahead of the reader or stays sufficiently behind the writer. Circular buffers are programmed in two ways.

- By setting the Loop_En control field in the SG_Table entry which enables a single entry SG_Table list which is circular.
- By setting the Link field of the last SG_table entry to point to the location of the first SG_table entry.

Linear Buffers

Once the end of the linear buffer is reached, the hardware will halt the processing of that stream until the stream is reprogrammed (i.e., the link portion of its SG-Table is written). In the case of data sources from the video decoder or transport parser, if the buffer reaches the full state, further data transfer requests are discarded along with the data. This ensures that other data services are not hampered. Linear buffering is not recommended for these data sources.

3.2.3.4 Stream Buffer Descriptors

The definition of where the stream buffer is located is virtualized to system memory by the use of SG-Tables. Each SG-Table entry describes a portion of the stream buffer wrt.:

- Physical start address – this defines the physical start address of the buffer.
- Physical end address – this defines the physical end address of the buffer.
- Physical ptr position – this defines the current ptr location within the buffer and is normally set to be equal to the Physical start address.
- Physical location of the next SG-Table entry – this defines the physical address of the next SG-Table entry.
- Auto increment enable on FB address (inc or non-inc) – this indicates whether the bus-master should increment the frame buffer address from DWORD to DWORD (or not).
- Selection of the FB address (1 of 6) – each buffer can src/sink from 1 of 6 frame buffer addresses. These may exist in a register or exist in the memory aperture.
- Loop on this table entry (Yes or No) – this indicates that this buffer will repeat upon itself upon wrapping.
- Valid Link (Yes or No) – this indicates that the link field contains a valid link entry which is fetched upon buffer wrapping.

Buffers may be defined by one or several SG-Table entries.

In the case of one SG-Table entry, the buffer is defined to consist of contiguous physical memory locations. The buffer may loop back on itself creating a circular buffer. This is selected by setting the loop control bit. If this bit is not set, the buffer will no longer accept stream requests until it is reprogrammed.

In the case of multiple SG-Table entries per buffer, the valid link indicator is used to indicate that the link field is valid. If the valid link indicator is zero, only one SG-Table entry is used to describe the buffer and depending on whether loop is set it may be a linear buffer.

Each stream buffer entry must be initialized prior to being enabled. This equates to essentially initializing the stream buffer to the first SG-Table entry through the register interface. Only buffers to be used need to be initialized. All other buffers are disabled. Input requests to disabled buffers is fatal and is reflected in the stream buffer status bits.

3.2.3.5 Scatter-Gather Support

Scatter-gather support is achieved by using the SG-Tables. Software can allocate blocks of memory in system memory space and point the stream buffer to them by using a SG-Table structure.

3.2.3.6 SG-Table Format

Table 3-2 0x<SG-Table Element>

Field Name	Bits	Description
Ptr_Addr	31:0	The physical address of the initial start position of the bus-master engine within the buffer. This can indicate an offset from the start of the block of data. Note: if the buffer wraps on itself (Loop_En = 1) then when it loops, this will reset to the start of the buffer. Field is Addr & 0xffffffff. Note 2: When programming the SG-Table Entry, the Ptr_Addr must be dword-aligned in order to support arbitrary byte alignment. Default Value : Start_Addr&"0000000"

Table 3-3 0x<SG-Table Element> + 1

Field Name	Bits	Description
Block_Left	31:20	The size of the request that remains to be completed in the next transfer. This field is used when the service of a transfer (either read or write transfer) is paused and needs to be continued in the next transfer. This field is used by the strm_int only. This field is applicable to bi-directional stream buffers only Default Value: 0
Block_Left_Status	19	0 – Last request is completed 1 – Last request is not completed, and the size of the remaining request is indicated in the "Block_Left" field. This field is used by the strm_int only. This field is applicable to bi-directional stream buffers only. Default Value: 0

Table 3-3 0x<SG-Table Element> + 1 (Continued)

Field Name	Bits	Description
Buffer_Status	18	The status of the buffer. This field indicates whether or not the buffer pointed to by the SG-Table Entry is in use. (Eg: This field must be set to 0 when a linear buffer has run out of space) 1 – Enable 0 – Disable Default Value: 1
Location	17	The destination/source of the transfer 0 – Video Buffer 1 – System Memory
C2S	16	Chip to system if set to a 1. System to Chip if set to a 0.
Burst_Size	15:4	Selects the burst size to be used for this requestor, in bytes. Note that read burst size of Video Buffer must be 128 bytes.
SG_Location	3	The storage locaton of the SG-Table. ‘0’->VB, ‘1’->PCI
Endian	2	Indicates whether or not to endian swap the data stream associated with this SG-Table (Stream Buffer). When set, the input DWORD “B3 B2 B1 B0” will be swapped to give the output DWORD “B0 B1 B2 B3”. ‘0’-> no endian swapping, ‘1’->swap
Flush_When_Full	1	When set, the SI flushes the write client data when the bi-directional buffer to which the client fills is full.
Reserved	0	For Future Use.

Table 3-4 0x<SG-Table Element> + 2

Field Name	Bits	Description
Link	31:25	The pointer to the SG-Table Entry counterpart in the SG-Table Cache. This is only applicable when the buffer pointed to by this SG-Table Entry is bi-directional.
Link_En	24	The valid flag for the “Link” field. Asserted when the stream buffer is bi-directional
Depth_Lag	23:0	The Depth (for write) or the Lag size (for read) of the stream buffer, in bytes. For write buffer: Default value = 0 For read buffer: Default value = Lag Size This field is applicable to bi-directional stream buffers only.

Table 3-5 0x<SG-Table Element> + 3

Field Name	Bits	Description
Start_Addr	31:7	The physical address of the start of the buffer, aligned to 128-byte boundary. The field is Addr & 0xffffff80.
Reserved	6:5	For Future Use

Table 3-5 0x<SG-Table Element> + 3 (Continued)

Field Name	Bits	Description
Bidir_Full_Dec_Cnt	4:2	Decrement Count after a bidirectional buffer becomes full. The number of read requests required before a write request on a bi-directional buffer, after the buffer becomes full. For the writer, this field is used by the hardware, and must be set to zero on reset. For the reader, this field indicates the number of read requests that must be made before any write request is allowed after the bi-directional buffer becomes full. Its maximum value is 7.
Buffer_Full	1	Indicates whether or not the stream buffer is full. This field should be set to zero by default, and is associated with the write component of a bi-directional buffer. This field should only be updated by the Stream Interface. This field is applicable to bi-directional stream buffers only.
Buffer_Empty	0	Indicates whether or not the stream buffer is empty. This field should be set to one by default, and it is associated with the write component of a bi-directional buffer. This field should only be updated by the Stream Interface. This field is applicable to bi-directional stream buffers only.

Table 3-6 0x<SG-Table Element> + 4

Field Name	Bits	Description
End_Addr	31:7	The physical address of the end of the buffer, aligned to a 128 byte boundary. The field is Addr & 0xffffff80.
Loop_En	6	Indicates that this table entry defines a complete circular buffer and that wrapping consists of going back to the current start address. 1 – Loop
Next_Table_En	5	Indicates that the link field is valid and should be used to navigate to the next SG-table.
Int_En	4	Indicates whether the limit value should be used to generate an interrupt. 0 – No Interrupt 1 – Interrupt when the Ptr_Addr crosses the Limit_Addr
Reserved	3:0	For Future Use

Table 3-7 0x<SG-Table Element> + 5

Field Name	Bits	Description
Limit_Addr	31:0	The physical address of the point at which an interrupt is to be generated. The interrupt is generated if the Ptr passes the point indicated by the Limit_Addr. Field is Addr & 0xfffffffE. Note: The maximum value of Limit_Addr of a block of memory is equal to End_Addr - 1

Table 3-8 0x<SG-Table Element> + 6

Field Name	Bits	Description
Next_Table_Ptr	31:2	The physical address of the next SG-Table Entry, aligned to DWORD. This field will be ignored if the field “Next_Table_En” is not set.
Reserved	1:0	For Future Use

Table 3-9 0x<SG-Table Element> + 7

Field Name	Bits	Description
Reserved	31:27	For Future Use
Pull_Mode_En	26	TPS Pull Mode Enable for this SG-Table: 0->Disable, 1->Enable Note: TPS Pull Mode can only be enabled for bi-directional buffers. Default Value = ‘0’.
TPS_Pipe_Sel	25:24	Pull Mode TPS pipe Select: Select one of the four TPS pull mode status bits (0, 1, 2, 3) to be updated by the TPS Pull Mode Logic. Default Value = “00”.
Pull_Mode_Thershold_Dep	23:0	The Pull Mode Thershold Depth: When Pull_Mode_En is set for a bi-directional buffer, the Pull Mode Logic within the Stream Interface updates the TPS pull mode status bit based on the value of TPS_Pipe_Sel as follows: Status Bit = ‘1’ when the Depth of the buffer > The Pull_Mode_Thershold_Dep Status Bit = ‘0’ otherwise Default value = 0xffffff

3.2.3.7 Defining the Frame Buffer Address

Table 3-10 0x<FBRegBase = n> [1]

Field Name	Bits	Description
FBAddr	31:2	The physical address of the frame buffer address to be used for the bus-mastering.
Reserved	1:0	For Future Use

[1] This entry is defined only for the first 6 elements. This is simply an addressing notation. Any of these entries are selectable through the Reg_Sel field.

These addresses are stored in the internal ram as well. Their addresses are defined as:

0x5 – Reg Addr 0

0xD – Reg Addr 1

0x15 – Reg Addr 2

0x1d – Reg Addr 3

0x25 – Reg Addr 4

0x2d – Reg Addr 5

3.2.4 External Interfaces

The stream interface connects to a multitude of read and write clients.

3.2.4.1 Read Clients

- Transport 0
- Transport 1
- MPEG decoder 0
- MPEG decoder 1
- VIP
- MPEG decoder context restore 0
- MPEG decoder context restore 1
- Audio 0
- Audio 1
- Audio 2
- Audio 3
- Smart Card 0
- Smart Card 1
- Smart Card 2
- DAIO

3.2.4.2 Write Clients

- Transport
- VIP
- MPEG decoder context save 0
- MPEG decoder context save 1
- Smart Card 0
- Smart Card 1
- Smart Card 2
- Audio
- DAIO0
- DAIO1

3.2.5 Performance

The following are the maximum data transfer rates when configured in a single mode of operation. Their verification is yet to be done on the real hardware.

- PCI write data transfer rate: 70 MB/s
- PCI read data transfer rate: 60 MB/s
- Frame buffer write memory transfer rate: 400 MB/s
- Frame buffer read memory transfer rate: 400 MB/s

3.3 Transport Demultiplexer

3.3.1 Introduction

The Transport Demultiplexer module in Xilleon 220 is based on a programmable micro-code engine (MCE), capable of parsing multiple transport formats. Programmability is required because there is no standard that covers all the different targeted markets.

The core, named Transport Processor System (TPS), is capable of handling different standards: ISO/IEC 13818-1, DVB (-S, -T, -C), DirecTV/DSS, and ARIB / BS/CS Digital. Each standard has its own microcode and input pipe configuration setting. The TPS can support two active formats at once. To support Picture-In-Picture and Time-Shifting, the TPS parses different sources by time-multiplexing the multiple pipes.

Directly coupled to the Transport Demultiplexer is the Conditional Access (CA) module. For CA details, refer to *"Conditional Access Module" on page 3-46.*

3.3.1.1 Features

General

- Complies with ISO/IEC 13818-1 based DVB, ATSC, OpenCable, JSAT, BskyB, and Canal + specifications.
- Complies with DirecTV standard.
- Parses the elementary components of a service, re-routes/splits PES, and sends them either to the hard-drive (Transport Packets) and/or to the frame buffer (i.e. MPEG2 A/V decoders (PES Packets) and/or MIPS (Transport packets, PES packets, Sections).
- Parses MPEG2 Program Streams (DVD playback) with leak rate controll for bit rate preservation.
- Time Shifting (transport stream playback) with time stamping of packets for bit rate preservation.
- Receives Transport/PES/Program streams through the PCI interface.
- Each transport demultiplexer input is capable of receiving serial (100 Mbps) or parallel (12.5 MBps) transport streams. The speed limitation of the CA (Conditional Access) block in case of DVB-CSA descrambling (7 cycles/byte) reduces the speed of the transport demultiplexer from 100Mbps to 60Mbps (serial) or 7.5MBps (parallel).
- Supports an MPEG2 stream from an external TS for the case of non-standard transport streams.
- Receives/transmits data from/to 1394 LINK layer devices in half-duplex mode.
- Simultaneous processing of 4 incoming streams (4 input pipes).
- Simultaneous support for two different standards on any combination of 4 inputs.
- Clock recovery per program ('event') supported (12 STC counters. 1 primary per pipe and 3 secondary ones on 2 pipes. Total of 2 audio STCs. Each STC group can be assigned to any input pipe. Audio STCs can be assigned to any video STC group).

- Six independent STC clcock generation circuits (4 video and 2 audio).
- Maximum of 32 PID filters per input pipe (all 4 input pipes can be attached to the same input stream in which case 128 PIDs are available for single input stream).
- One PID matching condition can be inverted (i.e., negative filtering).
- 32 HW section filters for Data Casting support that can be assigned to any PID filters on any input pipe. Section filtering can be performed on any combination of the 42byte(s) table header and payload, within micro-code hardcoded masking capability.
- 1 arbitrary condition per section filter feasible with negative filtering (i.e. inversion of matching condition).
- n SW section filters supported by TPS HW (CRC calculation) running on MIPS for low bitrate sections. n deppends on peer or solo mode. Sends transport packets of PIDs carrying sections to MIPS for SW section filtering.
- Supports time shifting on two input pipes in the system.
- Supports Packet substitution for recording/time shifting on total of 2 PIDs in the system.
- Lost Transport Packet detection via Continuity Count checks.
- CRC error checking and status reporting (PES and table sections for both HW and SW filtered ones)
- 32 interrupt types per input pipe.
- Extra serial transport port and capability to extract OOB data.
- Interface with two smart card modules (NRSS)
- Interface with internal CA module for scrambled streams. Supports TS and PES level descrambling.

Input Sources

The TPS receives data from the following sources:

- Transport Port A, configured as parallel or serial (12 pins).
- Transport Port B, configured as parallel or serial (12 pins).
- OOB interface, configured as serial (5 pins).
- 1394 in (13 pins).
- PCI interface (2 bus mastering channels).
- Two smart card interfaces / NRSS high speed input.

Note: OOB and serial 1394 can coexist in a system; otherwise only either OOB or 1394 can be present since they share GPIO pins.

Input Formats

The TPS receives data in the following formats:

- MPEG2 Transport packets.

- MPEG2 PES packets.
- MPEG2 Program streams.
- DirectTV Transport streams
- Other streams/packets (contact ATI for more information)

Output Destinations

The TPS can output data to several destinations:

- System memory.
- Frame buffer memory.
- 1394 LINK layer port configured as parallel or serial (Transport Packets).
- Transport Port C (HSDP) configured as parallel or serial (Transport Packets).
- I2S to audio block for internal or external decoder (PES Packets).

Output Formats

The TPS can output data in multiple formats:

- Full transport packets with or without time-stamp bytes.
- PES packets.
- AF (Adaptation Field) or some of AF data controlled by AF flags.
- PSIP sections.
- Some of PES header data controlled by PES flags
- Picture information notification (i.e., packet count and/or byte count) for trick mode playback.
- Other data based on micro-code programabillity on up to 8 PIDs per input pipe.

3.3.1.2 Applications

The four input pipes of the TPS allow a variety of applications. The table below summarizes some of the possible application combinations.

The *Application* column lists the applications per single live stream unless otherwise specified.

The *Data Movement* column lists data formats and their destinations. *V*- and *A*- stand for audio and video respectively. *PES* refers to MPEG2 PES layer packets. *PSI* is MPEG2 PSI/SI information. *MIPS* referses to the MIPS processor. *SPTS* is, from multi-program/event transport stream, demultiplexed single program transport stream "event" , i.e., transport packets.

The *Resources* column lists major resources and the number of processing passes necessary per packet per pipe. 1 pass/TP means that the MCE will process the transport packets (TP) only once. *Source* refers to live transport stream unless bus mastered (BM) input is used.

Table 3-11 Transport Demultiplexer Application Combinations

Application	Data Movement	Resources	Comments
View	• V-PES to V-decoder • A-PES to A-decoder • PSI* to MIPS	1 Input pipe single source 1 pass/TP µ code	Per pipe
Record**	• SPTS to hard disk • PSI* to MIPS	1 Input pipe single source 2 pass/TP µ code	Per pipe. To extract payload (PSI) and to send full packet to hard disk, we need two pass processing (both micro-code (MCE) and conditional access (CA) might be involved).
Play	• V-PES to V-decoder • A-PES to A-decoder • PSI* to MIPS	1 input pipe single (BM) source 1 pass/TP	2 pipes maximum.
PIP (Picture in Picture)	• V-PES1 to V-decoder1 • A-PES1 to A-decoder • PSI1* to MIPS • V-PES2 tp V-decoder2 • PSI2* to MIPS	2 Input pipes dual source 1 pass/TP µ code 1 pass/TP µ code	Clock recovery is performed both programs/channels. Core has to be ready to recover clock from program2, as well as A-PES2, in case the user switches the displays.
Time-shifting**	• SPTS1 to hard disk • PSI1* to MIPS • V-PES2 to V-decoder • A-PES2 to A-decoder • PSI2* to MIPS	2 Input pipes dual sources 2 pass/TP µ code 1 pass/TP µ code	2 live can be time shifted, i.e., a total of 4 pipes will be occupied. Clock recovery would be done on both live and stream coming in from hard disk. First one for time stamping, and the second one for stripping time stamp and for A/V decoder synchronization.
Viewing and recording**	• V-PES to V-decoder • A-PES to A-decoder • PSI1* to MIPS • SPTS to hard disk • PSI2* to MIPS	2 Input pipes dual sources 1 pass/TP µ code 2 pass/TP µ code	If recording and viewing the same program, then 1 Input pipe will be busy and no need for second navigation PSI extraction
PIP and recording**	• V-PES1 to V-decoder1 • A-PES1 to A-decoder1 • PSI1* to MIPS • V-PES2 to V-decoder2 • SPTS to hard disk • PSI2* to MIPS	2 Input pipes dual sources 1 pass/TP µ code 2 pass/TP µ code	Recorded program is the same as one of the viewed (either main display or PIP).
Time-shifting and recording**	• SPTS1 to hard disk • PSI1* to MIPS • V-PES to V-decoder • A-PES to A-decoder • SPTS2 to hard disk • PSI2* to MIPS	3 Input pipes 3 sources 2 pass/TP µ code 1 pass/TP µ code 2 pass/TP µ code	2 sources are from the tuner, the 3rd source is BM in.

Table 3-11 Transport Demultiplexer Application Combinations (Continued)

Application	Data Movement	Resources	Comments
PIP and Time-shifting **	• SPTS1 to hard disk • PSI1* to MIPS • V-PES1 to V-decoder1 • A-PES1 to A-decoder1 • PSI*1 to MIPS • V-PES2 to V-decoder2 • PSI2* to MIPS	3 Input pipes 3 sources 2 pass/TP μ code 1 pass/TP μ code 1 pass/TP μ code	All four pipes will be used if Time-shifting is done on both live stream inputs.
Recording and PIP and time-shifting **	• SPTS1 to hard disk • PSI1* to MIPS • V-PES1 to V-decoder1 • A-PES1 to A-decoder1 • PSI1* to MIPS • SPTS2 to hard disk • PSI2* to MIPS • V-PES2 to V-decoder2	3 Input pipes 3 sources 2 pass/TP μ code 1 pass/TP μ code 2 pass/TP μ code	Recorded program is the one displayed on PIP window .

*Note: PSI information can be sections coming from hardware section filter or transport packets coming from AUX_PID filter.

**Note: Any recording or time shifting will require time stamping and picture code tracking.

3.3.1.3 Bit Rates

Each processing pipe of the transport processor can handle unscrambled (clear) serial streams with bit rates up to 100Mb/s, or parallel streams with bit rates up to 12.5MB/s. The transport processor core thus supports up to four input streams 400Mb/s serial or 50MB/s parallel, all of which have to be unscrambled.

For scrambled streams, the CA block becomes a bottleneck for some scrambling/descrambling standards (refer to the CA block specifications for more details), the DVB case being the worst. Regardless of the number of scrambled input (i.e., whether all four or just one streams is scrambled), the TPS input FIFOs allow processing of a total of 240Mb/s (36.8MB/s).

Table 3-12 Maximum Input Bit Rates per Input Pipe Supported

Input Type	TPS Clock Frequency (MHz)	Serial Input Bit Rate (Mb/s)	Parallel Input Bit Rate(MB/s)
Not Scrambled	167	100	12.5
Scrambled (DVB)	167	60	7.5
OOB	167	100	n/a

3.3.2 Functional Description

3.3.2.1 Block Diagram

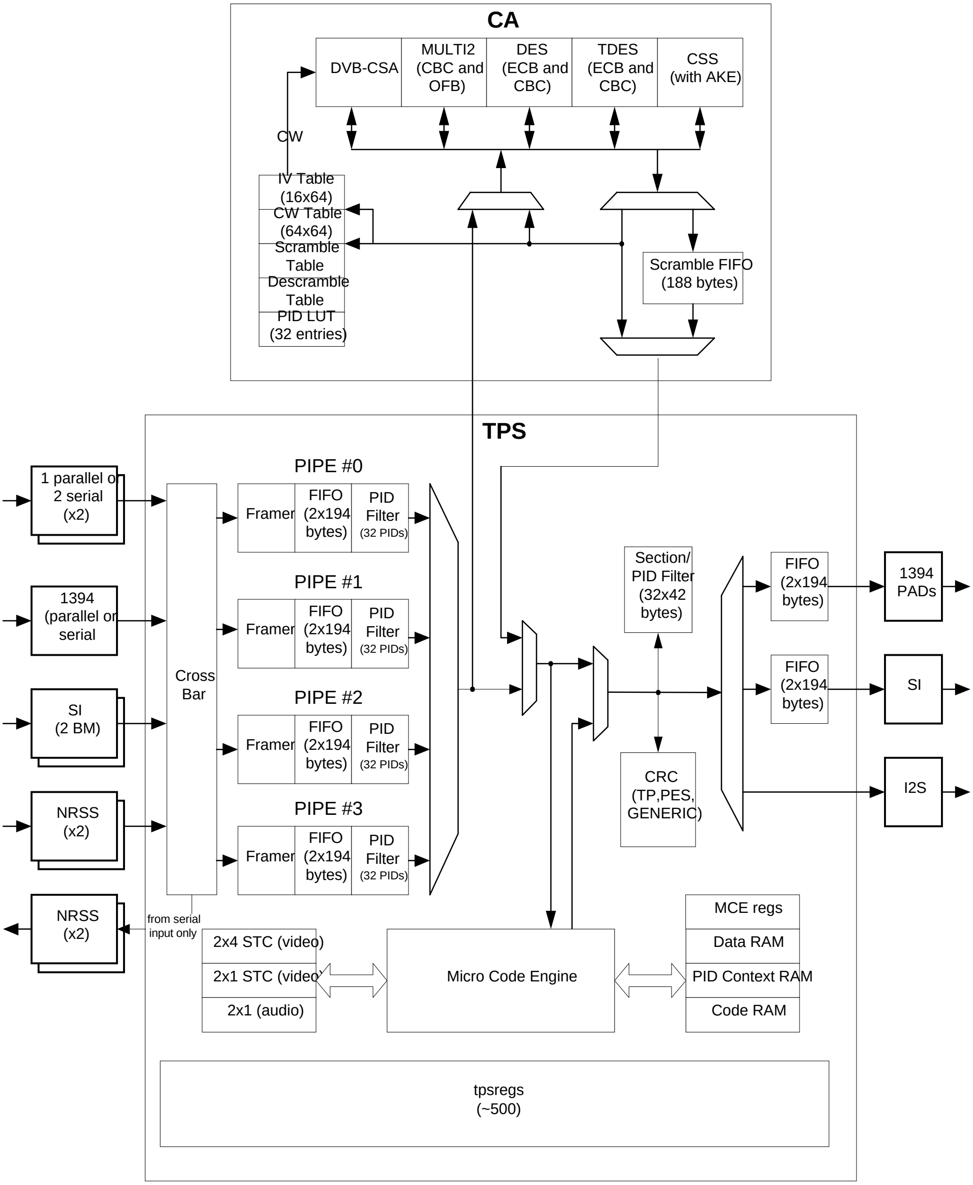


Figure 3-8 Transport Processor System (TPS) and CA Block Diagram

3.3.2.2 Description of TPS Sub-Blocks

Data comes into four framers from 4 out of 8 input sources . The framer establishes a lock on the incoming stream and realigns the data on byte boundaries if needed. After that, the data is passed through the PID filter bank. The matching condition determines the type of incoming data and the nature of processing that needs to be performed on it. Data is processed by one of the many parsers/extractors and is routed to one or more of the output queues. Serialisation of transport packets from all four inputs is performed here at the early stage of data pipe.

TPS is closely interconnected with the Conditional Access block and it shares the control over the currently processed input FIFO. Once TPS finds a packet is scrambled, it triggeres the CA to read the rest of the packet bytes. TPS waits for the descrambled data bytes at the output of CA.

To support features such as PIP and Time-Shifting, multiple input pipes of TPS core would be serviced in what will be the Transport Processor System (TPS). Since TP processing is done serially for all 4 inputs, there is only 1 CA system instantiated in Xilleon 220, at any given point in time, the TP of one stream input would be descrambled (if needed) by the CA module.

Transport Input Crossbar (TIC)
This is the input gateway to the TPS. All the possible input ports/sources come into this block, from there they are redirected to any one of the input pipes to be processed. The inputs to this block can be in many clock domains. The output is always a byte with all the additional control info, all on xclk. For serial input streams, the TIC packs the data in a byte and passes it to one of the framers which in turn determines the byte boundaries. It is possible to connect a single input port to multiple input pipes in order to increase the number of PID filters applied on that stream.

The TIC also hosts two bus mastering input ports. These ports are needed to suport time shifting of two live streams.

Bus Master Input FIFO
The bus master input FIFO is 64 DW deep. There are two instantiations of this FIFO. Each FIFO will start requesting data from stream interface as soon as bus mastering input is enabled. It will keep requesting as long as there is block of 32 DW empty space available in the FIFO. It will stop requesting if input pipe FIFOs are full. There is a leak rate control on the output of this FIFO which is used to control bit rate of the stream coming over the PCI bus.

Framers
There are four framers (each per input pipe). The framer is responsible for establishing sync on the input source. It receives byte wide data from the input crossbar along with the control (valid, req, sync), locates byte boundaries (in serial mode) and establishes sync.

The framer feeds the data to the input FIFO as long as the sync is established. The framer is also responsible for taking blind snap shots of STC counter(s) (controlled by the register SNAP_POS) the moment PCR enters the receiver (i.e., TS_FIFO).

The framer supports both MPEG2 transport stream framing and DirecTV transport stream framing.

Input FIFOs (TS_FIFO)

There are four input FIFOs. The input FIFO takes the data from the framer, and stores it until the MCE (Micro-code Engine) or the CA (Conditional Access) module is ready to consume the data — if the incoming stream is in the clear, the data is consumed by the MCE; for encrypted streams, the data is first sent to the CA module, then taken by the MCE.

Data consumption is performed on the TP (transport packet) boundary. One TP can be processed twice for the purpose of sending it to two different locations with two different indexes or in two different output types (TP and PES). The read side of the input FIFO(s) are controlled by flushtsfifo, rsttstmpptr and inreq instructions of the MCE.

The size of the FIFO is 2x194 bytes (188 + up to 6 time-stamp bytes when playing back from disk). This size allows the worst case scenario: 15µs (188*8/Bitrate) at 100Mbps for processing full transport packets on all four inputs, two of which in dual pass. At 60Mbps the time required would be 25µs.

Conditional Access (CA) Interface

The TPS block is closely interconnected with the CA block. Scrambled data go to the CA directly from selected TPS input FIFO and descrambled data are returned to the TPS block.

For details on the CA block, see *“Conditional Access Module” on page 3-46* .

Micro-code Engine (MCE)

The MCE is an 8-bit processor designed specifically for the application of digital stream parsing and demultiplexing. The same MCE with different micro-code is used for different stream standards/syntax (i.e., MPEG2 transport, DirecTV transport stream, Program stream, PES stream). The MCE contains a micro-code RAM, data RAM, datapath elements (comparator, shifter/rotator, ALU, etc), internal registers, byte counters, context RAM and registers, and Transport Backbone Bus (TBB) control.

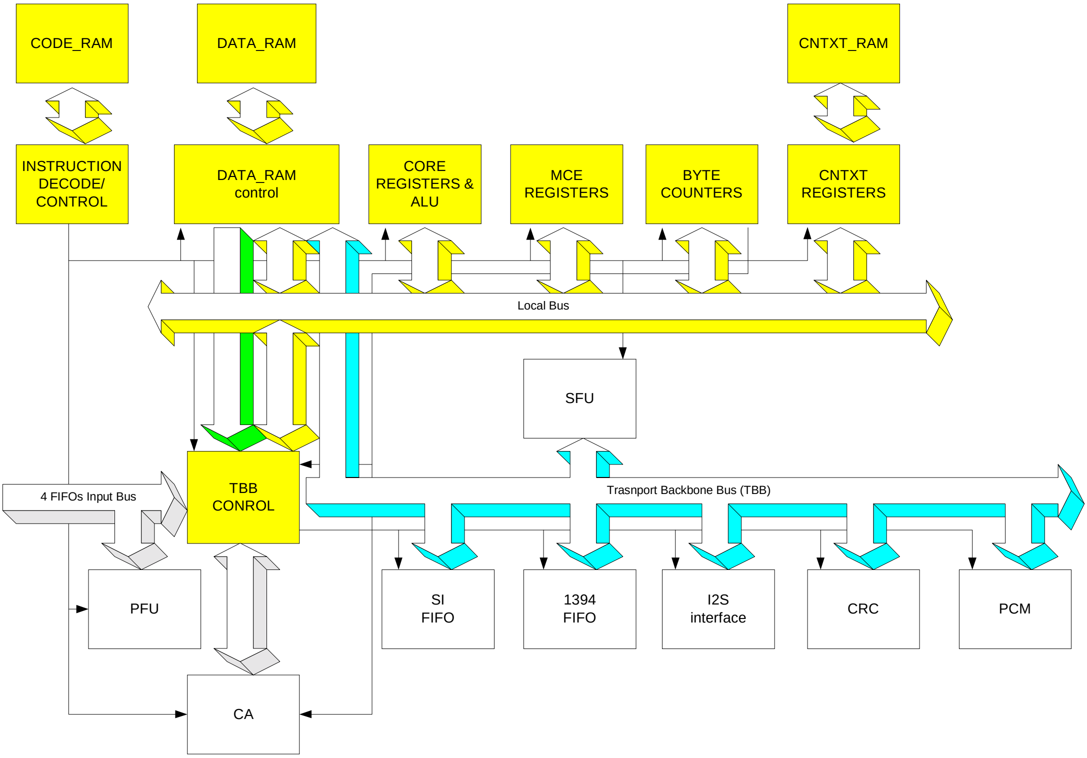


Figure 3-9 Micro-Code Engine Block Diagram

During the power up initialization sequence, the micro-code is loaded into the micro-code RAM by the CPU. The MCE waits (i.e., power saving mode - no instruction fetching cycles) for the first full packet to be ready in any of the pipes. As soon as there is at least one packet ready in any of the pipe FIFOs, it starts polling, in round robbin fashon, packet ready signals from all TPS input pipes. This way packets from all four pipes are serialized at the early stage of the processing pipe.

The MCE always processes the transport packet header bytes. PID filtering is done at this time and on PID filter match, the MCE continues with payload processing.

As a result of PID filtering, context is loaded from context RAM to the context registers. This context contains all TPS setting related to the particular PID as well as the states of the parsers from the previous packet carried by the same PID. At the end of each packet MCE stores the context registers back to the context RAM.

MCE parsers constantly monitor the transport_scrambling_control and PES_scrambling_control bits in the stream to determine if payload bytes are supposed to be descrambled before processing them. If descrambling is required, the MCE will pass necessary information (PID, scramble bits, etc) to the CA block and, based on TPS setting for scrambled packets, trigger CA block in corresponding mode to do the descrambling of the following bytes of packet payload. At that moment CA takes over control of the input FIFO and MCE switches its input to the output of CA block. Control of input FIFOs will be taken over by MCE again once byte counters indicate that all bytes of current packet have been processed.

Over the Transport Backbone Bus all bytes are transferred each to its corerresponding output client(s) (e.g., I2S is PES packet client so it will not receive transport header bytes, whereas 1394 is tranport packet client and therefore will receive all bytes of a packet). Bytes coming from the currently selected FIFO can be directed to multiple clients at the same time. This includes data RAM. MCE can transfer one byte at a time or it can burst multiple bytes (i.e. all payload bytes) to all the clients including the data RAM. The burst can be directly from input FIFO to the TBB (input and output directly hooked up), in which case the MCE is not fetching any new instruction until the burst is done.

Full control over every single incoming and outgoing byte is provided by micro-code parsers, i.e., micro-code programs executed by MCE. This gives full flexibility to the filter/demultiplexer hardware since it is possible to rewrite/readjust the parsers to the particular customer needs. This way it is possible to make trade-offs between functionality and speed of the micro-code/filter/demultiplexer.

For more on Microcode see *“Microcode Support” on page 3-28*.

PID Filters

The PID filtering unit has 32 PID filters per input pipe. All enabled PID filters of the currently processed input pipe are compared against PID bytes of the incoming stream using byte mask hard-coded in micro-code. The mask used for each of the PID value bytes is hard-coded per standard supported (i.e., 0x1F and 0xFF for MPEG2 PID, which is 13 bits wide, and 0x0F and 0xFF for DIRECTV SCID, which is 12 bits wide)

If there is a match on at least one PID, the MCE will parse that packet; otherwise it will discard the packet. In case multiple PIDs match, the PID filter with the higher number will be processed.

The type of processing required by MCE for matching PID is defined by the PID filter related registers as well as by the content of the PID context RAM (mainly by PID context entry 0).

MCE can perform some extra processing for the packets marked by certain groups of pointers, e.g.:

- On PIDs pointed by PCR PID pointers, the MCE will do PCR extraction from the adtapation field if present and generate appropriate interrupts.
- On PIDs pointed by check PID pointers, the MCE will do extra syntax checking, splice point handling and some other functios.
- On PIDs pointed by packet substitution pointers, the MCE will do packet substitution.
- On PIDs pointed by Extra PID pointers, the MCE will not do anything currently. These pointers are reserved for future upgrades where certain function can be assigned to them and micro-code can be added to support that function.
- On PIDs pointed by the Section Filter Unit pointers, MCE will perform section filtering,

i.e., section filters are assinged to those PIDs.

For example in *Figure 3-10*, for PID #15, MCE will perform, on top of functions required by PID context setting, PCR extraction, extra checking, and enabled extra functions.

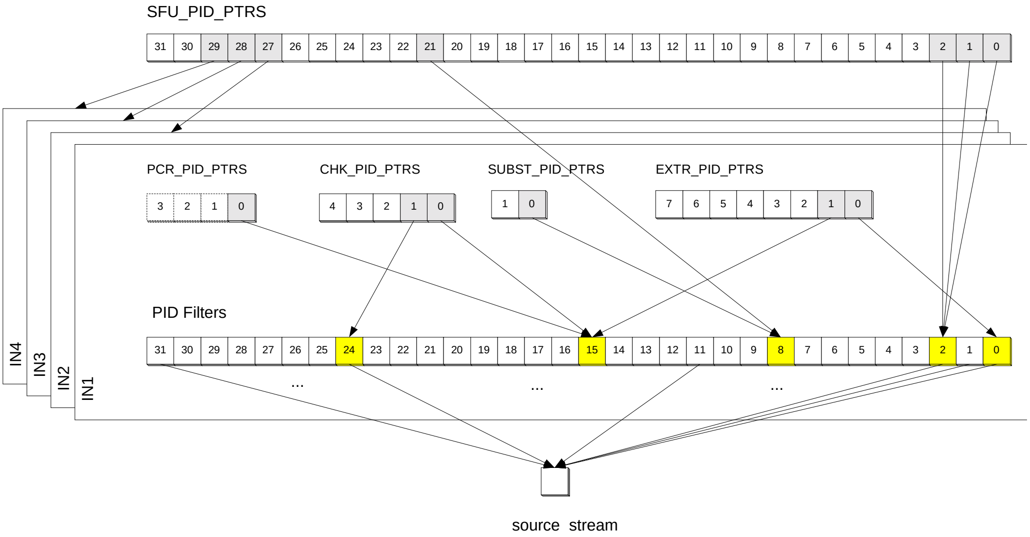


Figure 3-10 PID Filtering Unit and Special Functions done per PID

Section Filters

The four processing pipes share a pool of 32 hardware section filters. Each of the 32 sectin fitlers can be assigned to any of the 32 PIDs on any of the 4 input pipes. For example, all 32 section filters can be assigned to PID 0 of pipe 1, in which case no section filter is available for other 3 input pipes, or each pipe can be assigned 8 section filters all 8 being assigned to one PID on that pipe or single section filter assigned per PID to total of 8 PIDs per pipe.

TPS_SECTION_FILTER_0{..31}_CNTL and/or TPS_SFU_FIL_EN registers provide control over corresponding section filter.

Each section filter consists of up to 40 byte/bit conditions to support different section filtering depth requirements from different specs/customers. The number of conditions per filter is hardcoded in micro-code. By changing the micro-code one can adjust TPS to different filter depths and bit input bitrates for different customers. The conditions are stored in the Condition RAM (COND_RAM). Masks for each of the conditions are stored within microcode in CODE_RAM. One arbitrarry condition within each filter can be inverted, i.e., filter will match if condition is not satisfied.

Each section filter can generate an interrupt to the CPU and/or MIPS, depending on the selected mode of filtering, after each positive filtered section, or at the end, once all sections of a table have been filtered. The filtering mode can be one time or continuous. In one time mode, the section filter filters only one section/table and only once; in continuous mode, it keeps doing it until software changes the mode or disables the filter.

HW Assisted Software Section Filters

In addition to the 32 HW section filters sharable by all four pipes, it is assumed that filtering of low bandwidth sections will be done by internal MIPS processor. The number of software section filters is limited by the available bandwidth of the MIPS processor in a certain system configuration. The number of sections per transport packet that can be processed by MCE should go up to the theoretical limit of 60 sections per packet.

In this case of filtering, PID filtering and CRC checking is still done by TPS hardware while sections are filtered by software. PID carrying the sections (table_ids) that are supposed to be filtered by software should be set as AUX_SECTION filter. TPS AUX_SECTION parsing microcode will output whole transport packet with specified PID. It will trigger CRC engine at the proper time for each byte transferred within the packet. At the end of each transport packet payload it will append 2 bytes.

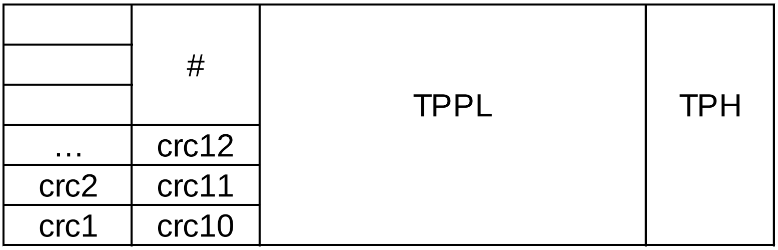


Figure 3-11 Transport packet containing multiple sections per packet as it appears in the frame/system memory

4 MS bits of the first appended byte represent the number of sections ended in the packet and the rest of the bits in this byte and the second appended byte are results of CRC calculation and its comparison against CRC bytes present in the sections. Bitx = 0 means CRC checked OK, bitx = 1 means CRC ERROR happened on the corresponding section X.

For networks and streams that will be putting more than 12 sections per transport packet, up to the theoretical 60 sections (3 bytes long each), for this mode of filtering we can rewrite/change the microcode to append 9 bytes. 6 bits in the first byte would be number of sections and then all other bits in the following 8 bytes would be the result of CRC comparison for the corresponding sections within the packet.

Microcode can be reprogrammed to append even more bytes if more sections are expected per packet, or if other information that might further assist to software filtering is needed.

Note that maintaining and outputting this information is adding overhead to the ~200 cycles necessary to process one transport packet by AUX_SECTION parser.

TPS Registers

There are four groups of registers in the TPS block

MCE Core registers — These registers make up the MCE. Thay can be accessed from CPU/MIPS as well as for reading during debugging through the TPS_MCE_CNTL_DBG register. In order to have meaningfull reading of this register, the MCE must be in single step mode.

MCE Internal/Status registers — These 8-bit registers are located in MCE’s address space. They have per pipe and other signals muxed into status registers as well as other registers and information used by MCE while processing a packet. All MCE internal registers can be accessed, for reading only, through the TPS_MCE_INDEX and TPS_MCE_DATA registers.

MCE Internal/Context registers — This group of 8-bit internal registers is also located in MCE's address space. They are loaded by MCE from PID context RAM at the begining of transport packet processing once PID match condition happens. They are stored by MCE in the PID context RAM at the end of current transport packet processing.

TPS configuration/MCE external registers — This group of 32-bit registers is accessible by CPU/MIPS only. Some of the fields of these registers are muxed in and hooked up/made available to the MCE status registers. They control the hardware arround MCE as well as MCE itself. Most of them are implemented one per TPS pipe, per PID filter or per section filter, few of them are for all four inputs.

CRC Block

The CRC block is designed to help the MCE calculate CRCs quickly and efficiently. It is composed of three CRC engines, two used by the MCE, and the third, the generic CRC engine, used for general purpose applications.

The CRC block is implemented in hardware as it is too computationally intensive to be implemented in software. It performs hardwired fast CRC computation for two polynomials (as specified in ITU-T H.222.0) on PES and section data in one clock cycle per byte, and it can use software programmable generic CRC engine for all the other polynomials (up to 32 bits) that need 2 cycles per byte. For all known polynomials, the CRC on a given byte can be computed in one clock cycle, for other combinations a 2-cycle implementation is used.

The generic CRC engine is flexible in its polynomial and vector size (both defined by registers), but slower due to the fact that it can only calculate one byte per every two clock cycles.

Normally, when the full data stream is fed in, the resulting CRC vector should be all 0’s. A signal indicating that the vector is all 0’s is also provided to the MCE.

3.3.3 Microcode Support

TPS and Microcide Engine (MCE) have hardware dedicated to speed up digital stream parsing. This hardware is controlled by special field in the MCE instruction and set of dedicated micro-instructions. Together with special microcode flow microinstructions they all provide very flexible means for parsing different digital stream formats.

Microcode for at least two different standards can be present in MCE's code RAM. Microcode that is not currently servicing any input pipe can be replaced by SW on the fly with another one.

Software loads microcode into MCE code RAM and sets PID context for each PID filter it wants to enable with the PID type and starting address of the parser that will be servicing that type of PID.

Special compiler is provided to compile microcode source file into binary file used to load MCE code RAM.

TPS microcode consists of Input pipe polling routine, standard selection jump table, transport layer parsers for different standards and PES/PSI/SI layer parsers (see *Figure 3-12*).

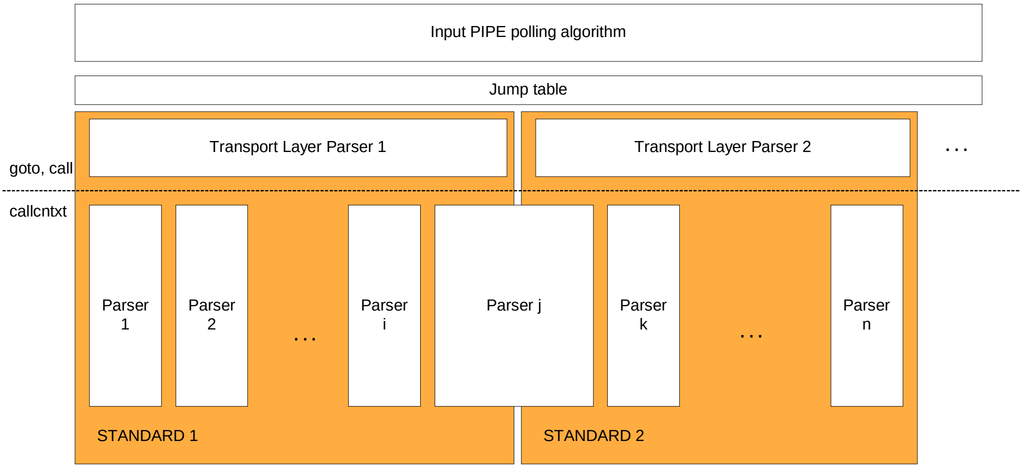


Figure 3-12   TPS Microcode Architecture

Input pipe polling routine will wait for full packet to be present on any of the input pipes and then through jump table start executing microcode for standard present on the corresponding input.

Jump table allows SW to chose/assign/reconfigure/reload on the fly which standard supporting microcode will be handling specific input pipe(s).

Transport layer parsers are standard specific. They are common to all PIDs i.e. all data types.

Different type of data carried by transport packets are handled by specific parsers. The same parser can be assigned to multiple PIDs of same type (i.e. PES parser is used to process audio, video PES packets). Some of these parsers can be shared between different standards (i.e. DVB and DirecTV have different transport layer but both are carrying video and audio PES packets). These parsers are called through special MCE instruction callcntxt. No parameters are provided with this instruction however they are all obtained automatically from context RAM. Context of each parser as well as its state is saved, at the end of each packet, per PID, per input, in the context RAM (please see on explicite retcntxt instruction or when byte counter 2 reaches 0 i.e. all bytes from the current packet are processed.

3.3.3.1 MPEG2 Transport Stream Support

All standards that are MPEG2 compliant (ATSC, DVB, JSAT, BskyB..) can be supported with single MPEG2 microcode. It is up to system SW to interpret different types of data being transferred by MPEG2 transport packets, and delivered to memory by TPS, with transport layer stripped off.

Transport Layer Parser

TPS configuration register setting Transport layer parser is responsible for: monitoring the transport packet header flags, for PID filtering, adaptation field parsing, redirecting transport layer scrambled bytes through conditional access block, and for preparing (loading context) for and executing the cntxtcall instruction (i.e. for calling the appropriate sub-transport layer parser).

Based on the transport header flags the parser decides to drop or not current packet, if adaptation parser needs to be called or if conditional access block should be triggered for payload descrambling.

Only packets that cause PID filter match will be processed. All others will be dropped.

Adaptation field parser is responsible for extraction of whole or part (i.e. private data) of the adaptation field, and for extraction of PCR field and loading it into the clock recovery circuitry of the processed input.

If transport level scrambling was applied to the incoming stream transport layer parser will trigger conditional access block at the moment first byte of scrambled packet payload is about to be processed. The parser that will be called after that will be processing descrambled data from the output of conditional access block.

ldcntxt instruction will load context of the matching PID from context RAM into context registers. callcntxt instruction will load next instruction pointer from that context as well. This instruction pointer was next instruction pointer of the parser automatically saved at the very end of the previous packet. This will cause the parser to continue from where it ended at the end of previous packet.

Automatic return back to transport layer parser and context save will happen at the end of packet, once byte counter that counts bytes being sent out reaches 0.

PES Packet Parser

PES packet parser is very simple one since in front of each decoder there is hardware PES parser that does full PES header parsing. Hardware one is stripping PES header and its output is elementary stream (ES). Microcode PES parser is responsible for outputing PES packets to the coresponding memory buffers, for monitoring scramble bits in the PES header in order to determine if PES payload should be sent through conditional access block, for picture code monitorring and for counting PES payload bytes in order to help with trick mode playback.

AUX Parser

AUX parser is just doing PID filtering. It is sending full transport packets to the corersponding buffer so that software parsers can process it. If packets are scrambled on transport level the parser will send them through conditional access/dscrembler block before sending them copy protected out.

PSI/SI Table/Section Parser

PSI/SI parser is responsible for controlling section filter hardware during section filtering. Only sections that match specified conditions will end up in the buffer. The parser will perform section filter condition comparison as the bytes are transferred to the temporary storage in the data RAM. At the end of comparison it checks the result of hardware comparator to determine weather to drop or to send the section. Current microde parser supports 16 condition (i.e. up to 18th byte) per filter. Hardware allows up to 42 conditions or 8byte/conditions + 8byte/condiions where the second 8 can be with the offset of up to 31st byte. However MPEG2 microcode has to be changed to support this at the expense of input bit rate reduction .

While section bytes of a matching section are being transferred to the output hardware CRC engines are triggered by the parser at the appropriate times. At the end of the section parser checks the result of CRC block and appends extra byte to the section in the buffer. Bit 0 of this byte will inform system software if there was CRC error or not during previous section transfer, so that software can determine if the section is to be dropped or not at that time.

Section parser will keep processing the sections from current packet until it finds stuffing byte or byte counter 2 signals the end of packet. In the second case we have situation of section spanning the transport packets. In both cases the state of the parser is being saved into the context RAM of the PID carrying the section, for the time next packet of the same PID arrives. Once the next packet arrives callcntxt instruction will load the context of the section parser and the parsing will continue from the point it stopped in the previous packet.

AUXPSI/SI Table/Section Parser

Auxiliary section parser is the same as AUX parser in that it is sending full transport packets with sections out to the buffer. However during the transfer the parser is triggering the CRC engines per section at the appropriate time. At the end of packet two bytes are appended to the packet in the buffer. This two bytes carry information about the number of sections that were present in the packet and result of CRC comparison for each of them. System software will be doing the filtering of sections and monitoring the CRC status from the two bytes.

3.3.3.2 DVD/MPEG2 Program Stream Support

The same TPS hardware is controlled by MCE and used to process DVD, i.e., MPEG2 Program Stream. TPS space PID registes are used as SID registers in order to have axes to context RAM per SID. Both PID register bytes have to be loaded with the same SID value of the PES packet of an elementary stream to be extracted from the program stream. SID comparators from MPEG2 transport stream support are used for audio sub SID filtering. Context RAM usage is shown in *Figure 3-13*. The remaining context RAM can be used as data RAM by seting PID_NUM register before doing ldcntxt stcntxt.

Program stream is processed by the MCE in packet chunks in order to alow the time for processing the other pipes. These packets can be of the similar size of that of MPEG2 transport packet.

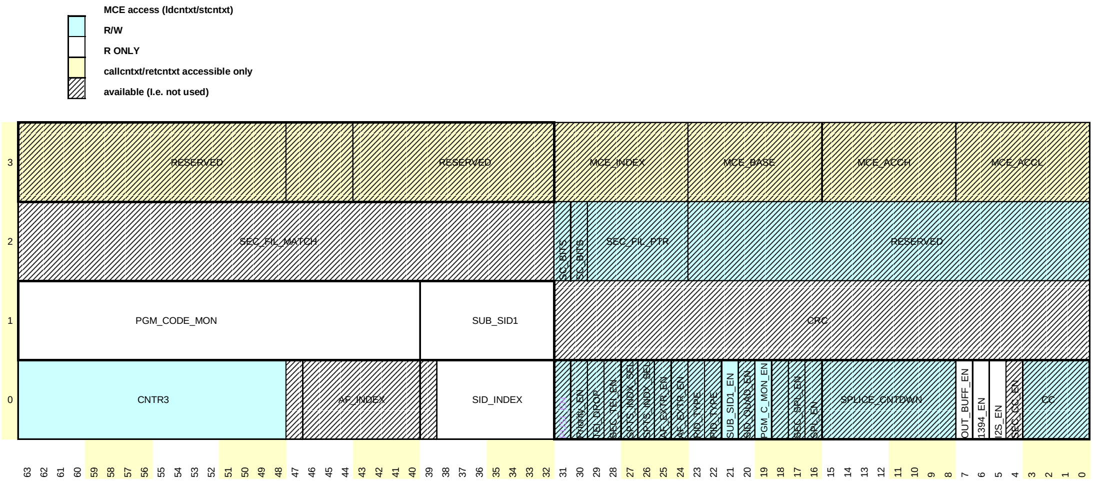


Figure 3-13 Context RAM Content for 1 SID filter in DVD MPEG2 Program Stream Support

3.3.3.3 PES Stream Support

Raw PES video and audio packets normally are sent directly to the buffers in front of the video and audio decoders through bus master inputs, i.e., there is no need for the packets to go through TPS. Only in case both audio and video packets are merged into the same file does one need to filter on video or audio SID in order to direct them to the corresponding buffers. This microcode would do that type of filtering. The register and context setup would be similar to DVD program stream setup, but microcode would be different.

3.3.3.4 Dual Pass Packet Processing

Recording of a single program transport stream filtered out of incoming multiprogram transport stream can be enabled by marking all pids that are to be recorded through setup of the PID context RAM fields. Since the recording buffer has its own INDEX, it is not possible to send the same byte of a payload/packet to two different buffers in the same cycle. Microcode will perform dual pass processing per packet, making sure the second pass is using buffer index specified for recording.

3.3.3.1 DIRECTV Stream Support

Figure *Figure 3-14* shows the usage of CNTXT_RAM for DirecTV stream support. Filtering on SCID is done using the same hardware used for PID filtering in MPEG2. Payload of all transport packets identified by the same SCID is routed to the buffer index specified by SCID_INDEX. Control words for scrambled stream support can be extracted and directed to the buffer index specified by CW_INDEX.

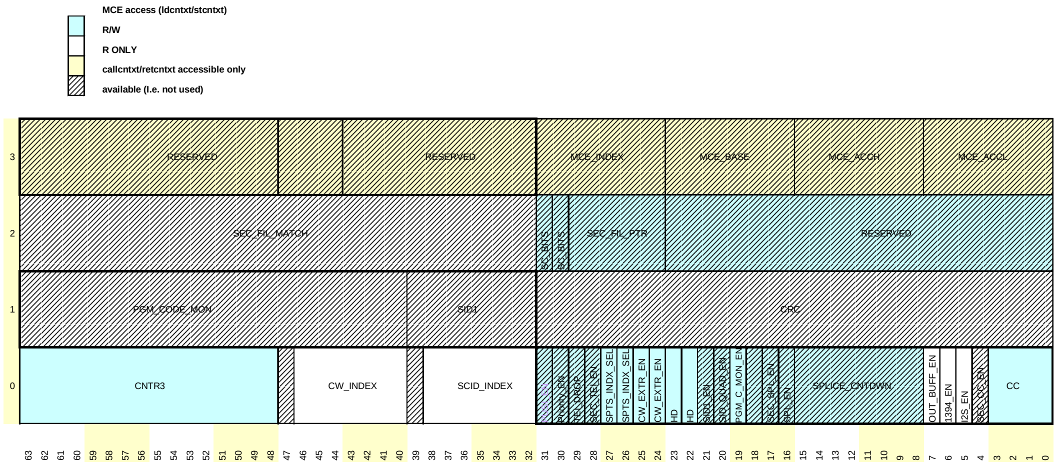


Figure 3-14 Context RAM Content for 1 SCID Filter in DirecTV Transport Stream Support

3.3.3.2 Support of Other Digital Streams

In general it should be possible to write microcode to process any type/format/standard of digital stream as long as electrical interface with transport inputs match or if the stream is being fed through bus master inputs. The only limits are curent TPS HW resources (i.e., number of comparators, filters, and RAM available) and possibly the number of cycles necessary for processing those packets.

3.3.4 Filtering on Transport Header Fields

Different filtering modes on these fields can be enabled by setting corresponding bits in the PID context RAM.

3.3.4.1 Transport_error_indicator

By default the microcode engine (MCE) will ignore this bit on all packets of all PIDs, so it will process all packets based on PID match condition. It is possible to force MCE to drop all packets with transport_error_indicator bit set to 1 in the stream on per PID bases. To do this SW has to set TEI_DROP bit in the context RAM for the PID of interest. In addition by setting SEC_TEI_EN bit in the PID context RAM MCE can be instructed to insert/send Error_Code into the output stream whenever packet has been droped.

3.3.4.2 Payload_unit_start_indicator

By default the microcode engine (MCE) will ignore this bit on all packets of all PIDs, so it will process all packets based on PID match condition and other processing requirements for that PID. MCE can be instructed to start acquiring and/or processing packets on certain PIDs starting from the packet that has payload_unit_start_indicator flag set to 1, by setting PUSI_EN bit in the PID's context RAM.

3.3.4.3 Transport_priority

By default microcode the engine (MCE) will ignore this bit on all packets of all PIDs, so it will process all packets based on PID match condition and other processing requirements for that PID. Setting Priority_EN bit in PID context RAM for a PID will force MCE to acquire/process only packets of that PID that have transport_priority flag set to 1.

3.3.4.4 Transport_scrambling_control

By default the microcode engine (MCE) will monitor these bits on all packets of all PIDs. If these bits suggest that packet is scrambled MCE will trigger CA block at the propper moment and it will wait for CA to deliver all remining bytes of the packet descrambled. Please note that this can cause MCE to hang if undetected error on clear packets exists and it happens to be on the transport_scrambling_control bits.

It is possible to change the behaviour of the MCE for certain PIDs by setting SCR_PKT_CNTL bits in PID context RAM for that PID. (1) Normal functionality - scrambled go to CA, (2) Force all scrambled packets to be dropped. (3) All scrambled packets can be forced out of the TPS without going through CA. This can help prevent hang in case it is known that the incoming stream is in clear. (4) Mainly for debugging purposes it is possible to force even not scrambled packets to the CA. This setting can also help prevent scrambled packet to go out not descrambled, if undetected error causes scrambled packet to be treated as not scrambled. The CA block has to be set properly whenever packets are being sent to it by TPS block.

3.3.4.5 Adaptation_fileld_control

Any of 32 PIDs on each input can be selected for transport packet level, adaptation field or PES level data extraction. AF_EXTR_EN, PID context RAM bits were assigned function of controlling extraction of adaptation field per PID. It is possible to extract full adaptation field, or just private data (but not both at the same time). In both cases extracted bytes are going to the bufer whose index is specified in AF_INDEX field in the PID context RAM.

3.3.4.6 Continuity_counter

By default the microcode engine (MCE) will monitor continuity counter.

MCE will drop duplicate packets unless original packet was missing or was dropped from the stream.

MCE will generate PCRx_DISCONT interrupt in case of PCR discontinuities.

3.3.5 Extraction of Flag Controlled Data - Adding Functionality (EXTR_PID)

For future upgrades pointers on 8 PIDs on each input pipe are provided. On these PIDs it is possible to enable up to 8 other functions. For each of the functions, the interrupt bit is provided in the interrupt register of each input pipe as well as enable bit for each of these interrupts.

For example, the transport stream data controlled by the different flags in the stream (i.e., in transport or PES packet header) can be extracted if enabled by the corresponding bit in the corresponding PIDx_flags register. Which flag from PIDx_flags register corresponds to which flag in the stream (not all will be, even though any of them can be extracted) is programmable at the time when micro code for the parsers is written. Which flag to extract data for is then enabled or disabled by the host CPU or MIPS at the run time. Where the extracted data goes is also predetermined by micro code. It can be local RAM, also accessible by host CPU/MIPS, or system or frame buffer (i.e. AF_INDEX can be used for that). One restriction applies in case full AF, PES header or table header has been selected for extraction on one PID: we can't request on the same PID at the same time extraction of some flag controlled data from those full data sections if they are supposed to go out to the same TBB client (i.e. SI) since they would have the same index and they would be interleaved on the client side (i.e., we would need multi-pass processing).

3.3.6 Stream Syntax Monitoring and Statistics (CHK_PID)

On 5 PIDs per pipe (please see TPS_INx_y_CHK_PID_CNTl registers in the register spec) it is possible to do extra checking and notifying the CPU/MIPS through interrupts about the results of the checking as well as about the error frequency.

Transport Packet Syntax is checked for some of the packet fields. If error occurred CHK_PID_TP_ERROR interrupt will be set for the corresponding pipe and error counter in MCE data RAM will be incremented. Which CHECK_PID was the source of the interrupt will be set in the TPS_IN1_CHK_PID_INT_SRC2 register and CHK_PID_TP_ERR_TYPE in TPS_INx_y_CHK_PID_INT_CNTL register will be set accordingly. Error types supported by current mpeg2 microcode (and can be changed by rewriting the microcode) are: dropped packet with scrambled bits set for stream that is in clear, dropped packets with illegal adaptation field flag, dropped packets with the same continuity counter (i.e. duplicate packets), dropped packets with illegal adaptation field payload length, dropped packets with illegal private data length.

Transport_error_indicator (TEI) flag is checked by micorode on all PIDs to determine weather to drop or to use arrived packet. Default behavior of microcode is to drop the packet with this flag set. This functionality can be changed on the fly by SW setting TEI_DROP bit in the context RAM. Weather CHECK_PID_TEI_ERROR_INT will be generated on some (i.e. CHK_) PIDs on TEI appearance is controlled by bit in TPS_Inx_y_CHK_PID_INT_CNTl register. One bit in the context RAM (SEC_TEI_EN) also controls weather sequence error code is going to be inserted by microcode into the decoders input buffer. Number of errors occurred is available in MCE data RAM.

The continuity_counter (CC) is monitored on all PIDs. The behavior of microcode is according to the ITU-T H.220.0 spec. CC error interrupts will be generated and counted only on CHK_PIDs if enabled through corresponding registers. Once interrupt is generated, which CHK_PID was the source will be available in TPS_INx_CHK_PID_INT_SRC2 register. Number of errors/interrupts will be available in MCE data RAM. Insertion of sequence error code in case of CC error is controlled by SEC_CC_EN bit in context RAM.

The discontinuity_flag is monitored by micorocode on all PIDs together with continuity_counter flag. Interrupt on discontinuity_flag=1 event is generated and counted on CHK_PIDs only. This interrupt is controllable per CHK_PID. Once interrupt is generated the source of the interrupt is logged in TPS_INx_CHK_PID_INT_SRC1 register. Number of errors will be available in MCE data RAM.

random_access_flag = 1 can cause interrupt on CHECK_PIDs if the interrupt generation is enabled for the corresponding CHK_PID. Which CHK_PID(s) are the source of interrupt is logged in TPS_INx_CHK_PID_INT_SRC1.

splicing_point_flag is monitored only on CHK_PIDs. Interrupt will be generated if enabled when this flag is detected to be 1 in the stream. SPLICE_COUNT_DOWN counter in context RAM is also updated on every following appearance of this flag until SPLICE_POINT happens.

Splice point (as explained in the ITU-T H.220.0 spec) will be monitored only on CHK_PIDs. When splice point happens, an interrupt will be generated and CHK_PID_SPLICE_PID will be automatically transferred into the PID register of the PID filter pointed by CHK_PID_PTR. From the first occurrence of the splice_point_flag to the moment splice point occurs, software has time to update the CHK_PID_SPLICE_PID register.

3.3.7 Transport Backbone Bus (TBB)

This bus consists of 8 bit data, 7 bit index, 8 bit client select and single write request. The bus is controlled by the MCE register writes and outreq micro-instructions.

From the MCE point of view, all clients on the bus are write only type: one output FIFO for all data going to memory through stream interface (SI) block, one FIFO for HSDP/1394 interface, I2S/audio block interface, CRC engines, local data RAM, picture code monitor (PCM) and section filter unit (SFU).

Each client accepts bytes that are intended for it. This is controlled by MCE writing to the TPSMCE_TBB_SEL register. Since the transport packet can be viewed as a sequence of byte blocks with the length being predetermined/fixed or coming from the stream bytes themselves, the number of MCE writes to the TPSMCE_TBB_SEL register will be small, i.e., only on region boundaries (9 times during 188 byte transfer).

Figure 3-15 summarizes the client selection/enable across the byte regions of a transport packet. Transport packet header (TPH) is fixed length. PSI header (PSIH) is fixed length, depending on the section syntax indicator. Adaptation field (AF), Adaptation field private data (AF - PRIVATE), PES header (PESH), PSI payload (PSIPL), PES payload (PESPL) start with the bytes carrying its length.

		STUFFING	PSIPL/PESPL	PSIH/PESPL	PSIPL/PESPL	PSIH/PESH	AF	AF - PRIVATE	AF	TPH
PSI/SI	**I2S**	-	-	-	-	-	-	-	-	-
	SI	-	EN	EN	EN	EN	EN*	EN**	EN*	-
	1394	EN	EN	EN	EN	EN	EN	EN	EN	EN
	CRC	-	EN	EN	EN	EN	-	-	-	-
	SFU	-	-	EN	-	EN	-	-	-	-
	PCM	-	-	-	-	-	-	-	-	-
	SI index	-	PSI	PSI	PSI	PSI	AF	AF	AF	-
PES	**I2S**	-	EN	EN	EN	EN	-	-	-	-
	SI	-	EN	EN	EN	EN	EN*	EN**	EN*	-
	1394	EN	EN	EN	EN	EN	EN	EN	EN	EN
	CRC	-	EN	EN	EN	-	-	-	-	-
	SFU	-	-	-	-	-	-	-	-	-
	PCM	-	EN	EN	EN	-	-	-	-	-
	SI index	-	PID	PID	PID	PID	AF	AF	AF	-
AUXPSI/S	**I2S**	-	-	-	-	-	-	-	-	-
	SI	EN	EN	EN	EN	EN	EN	EN	EN	EN
	1394	EN	EN	EN	EN	EN	EN	EN	EN	EN
	CRC	-	EN	EN	EN	EN	-	-	-	-
	SFU	-	-	-	-	-	-	-	-	-
	PCM	-	-	-	-	-	-	-	-	-
	SI index	PID	PID	PID	PID	PID	PID	PID	PID	PID
AUX	**I2S**	-	-	-	-	-	-	-	-	-
	SI	EN	EN	EN	EN	EN	EN	EN	EN	EN
	1394	EN	EN	EN	EN	EN	EN	EN	EN	EN
	CRC	-	-	-	-	-	-	-	-	-
	SFU	-	-	-	-	-	-	-	-	-
	PCM	-	-	-	-	-	-	-	-	-
	SI index	PID	PID	PID	PID	PID	PID	PID	PID	PID

Figure 3-15 TBB Client Enable and Index Scheme Across a Transport Packet Byte Regions

The expected behavior from all clients is that they are always ready, i.e., they have enough buffering on their side. REQ - RDY handshake is still implemented with all clients so that all of them can stall if one of them is not ready (Generic CRC engine needs 2 cycles per byte if ever used). If any of the clients is not ready for a period of time longer than the input FIFOs can accept, one or all of the FIFOs will overflow. This can cause the MCE to hang because data in the FIFO will no longer be packet aligned.

Most of the time the input FIFO is directly hooked up to the TBB bus through the TBB control block for byte bursting purposes, until some of the byte counters signal the end of burst. Only some bytes from transport packet, PES or PSI section header are examined by the MCE.

3.3.7.1 Stream Interface Client

Data from all four input pipes destined to memory is going out through single stream interface client. Any type of data (i.e., PES packets, PSI/SI sections, extracted AF, private data, extra information added by MCE) will be sent by different MCE parsers to this client. See *Figure 3-15* for more details on which byte will end up in this client.

This client consists of a15x388 output FIFO and logic to write/read to/from the FIFO and handshake with stream interface block and TBB. This client is provided with index/tag of the destination buffer with each byte transferred. Based on the index and setting of stream interface block (SG tables), each of these buffers can be in system or frame buffer memory.

3.3.7.2 1394 LINK layer Chip Client

This client is TPS output interface with TSB42AA4/TSB42AB4 (ceLynx), a 1394 LINK Layer controller in single stream half duplex mode. Modes A (CLK, DATA and VALID signals) and B (CLK,DATA, VALID and SYNC signals), with fixed transport packet length, of the ceLynx chip are supported (please see ceLynx Data Manual for more details).

One single program transport stream from any of the four pipes can be directed to this port. See *Figure 3-15* for more details on which byte will end up in this client.

In order to preserve bit-rate and to reduce jitter introduced by MCE processing four input pipes, time stamps are used internally to determine presentation time of the transport packet on this clients port. TS_FIFO will take snapshot of a free running counter, running of the corresponding pulled STC clock, on packet arrival. MCE will prepend that snapshot to the transport packet at the moment of processing/sending it to this client's FIFO. The packet will be sent out of the FIFO to the interface once the snapshot matches the client's local STC counter.

3.3.7.3 Audio (I2S) Client

Audio PES packets will be going to this TBB client. This client is alternative path from TPS to audio decoder block. See *Figure 3-15* for more details on which byte will end up in this client.

3.3.7.4 CRC Client

Three CRC engines are considered to be a single client since only one engine corresponding to the packet type (i.e. PES, PSI, please see "CRC Engines" section for more details) will be used during current packet processing. MCE will select this engine before sending the sequence of bytes that are supposed to be accumulated by CRC engine. See *Figure 3-15* for more details on which byte will end up in this client.

3.3.7.5 Picture Code Monitoring Client

This client will be typically selected during transfer of PES payload bytes. See *Figure 3-15* for more details on which byte will end up in this client . See *"Trick Mode Playback and Picture Code Monitoring (PCM)" on page 3-42* for more details on the functionality of this client.

3.3.7.6 Data RAM Client

Data RAM client is special client that can be source of TBB data or it can be destination client on TBB. Any part (i.e. type of data) of the transport packet can be stored to data RAM. The

MCE can modify or replace them and burst them out from data RAM to TBB.

3.3.8 General Purpose IO Pins

NRSS1, NRSS2 and TPA port pins can be used as GPIO pins. The TPS_GPIO_CNTL register selects between normal and GPIO function (please see register spec for details).

The TPS_GPIO_DIRECTION register determines the direction of a pin once in GPIO mode. The TPS_GPIO_DATA register drives values to the pins that are in GPIO mode and in output direction. Read path of this register is connected to the pins themselves. That way the state of the pins can be read regardless whether they are driven by the TPS_GPIO_DATA register, or by an external device.

3.3.9 Debugging

For debugging purposes, 4 banks x 16 groups x 11 signals from within the TPS block are muxed out to GPIOA pins for viewing with logic analyzers and oscilloscopes.

Through the TPS_MCE_CNTL_DBG register it is possible to check the state of microcode engine (MCE) once it is running in single step mode, after each instruction has been executed or after the highly unlikely case of an MCE hang. There are 25 groups of 16 signals that can be checked.

All MCE internal registers are accessible for reading by CPU/MIPS through TPS_MCE_INDEX and TPS_MCE_DATA. These registers are useful in MCE single step mode or after an MCE hang.

There is a register in the MCE address space (TPSMCE_RESERVED) whose bits can be muxed out to GPIO pins. This register can be used as "print" instruction from the microcode to notify certain conditions or states within the microcode/parser/stream.

The GPIOA pins can be used to drive the TPS transport parallel and serial input ports, once proper connection/cable is established. The TNF/1394 input port cannot be tested this way since the TNF_CLK pin is in output mode even when the port is in input mode.

The TPSGPIO port/registers can be used as well to emulate live feed to TPA, NRSS1 and NRSS2 ports without any cable connections. Every bit of the TPA input port and both NRSS ports are connected to the TPSGPIO control, direction and data registers.

All FIFOs in TPS can be put in debug mode. In this mode normal functionality of the FIFO is stopped and software can access the FIFO for read and write operations through the corresponding FIFO control register.

3.3.10 External Interfaces

3.3.10.1 Transport Input Crossbar (TIC)

This is the input gateway to the TPS. All the possible input ports/sources come into this block, from there they are redirected to any one of the input pipes to be processed. This way the whole design is very symmetrical, and there are no assumptions as to which pipe can perform what. The inputs to this block can be in many clock domains. The output is always a byte with all the additional control info, all on xclk. For serial input streams, the TIC would pack the data in a byte and pass it to one of the framers which would in turn determine the byte boundaries. It is possible to connect single input port to multiple input pipes in order to increase the number of PID filters applied on that stream.

Transport Input Ports A and B Parallel

These are 12 pin ports with CLK, SYNC, VALID, ERROR and DATA(7 :0) signals. They can be configured as single parallel or each being 2 serial ports. Polarity of each signal is selectable by software as well as enable of the control signal functionality (i.e. port can be configured as CLK and DATA only, CLK, VALID and DATA…)

Transport Input Ports A and B Serial

These are 5 pin input ports with CLK, SYNC, VALID, ERROR and DATA signals being the same signals/pins used for parallel except that only one DATA(0) bit is used for serial DATA transfer. Polarity and signal function enable is controlled by software.

Transport Input Port A and B Serial 2

These are 5 pin input ports with CLK, SYNC, VALID, ERROR and DATA signals. The remaining DATA lines not used in port A and B serial mode are mapped into port A and B serial 2 ports. Polarity and signal function enable of this ports pins is controlled by software too.

OOB Input Port Serial

This is another 5 pin input port with CLK, SYNC, VALID, ERROR and DATA signals. Polarity and signal function enable of this ports pins is controlled by software.

1394 (TNF) Input Port Parallel

This is 13 pin input port to interface with the TI Cilinx (and any other with the same/similar interface) 1394 LINK layer chip (i.e., in single stream/buffer mode - no ADDR pins are present to select the buffer/stream). It can be used in software controlled half duplex mode with 1394 output port. It has CLK, SYNC, VALID, R/W, AV and DATA signals.

1394 (TNF) Input Port Serial

This is 6 pin input port to interface with TI Cilinx (and any other with the same/similar interface) 1394 LINK layer chip (i.e. in single stream/buffer mode - no ADDR pins are present to select the buffer/stream). It can be used in software controlled half duplex mode with 1394 output port. It has CLK, SYNC, VALID, R/W, AV and DATA(0) signals (Please see HDTV2 Pinout Spec for more details).

DMA/Bus Master Input Port 1 and Port 2

Transport input cross bar also hosts two bus mastering input ports. These ports are needed to support playback and/or time shifting of two streams.

NRSS I/O Port 1 and Port 2 Serial

These are two smart card transport stream interfaces. Scrambled data go out and descrambled data come back in through this port. It is possible to select which serial transport input is hooked up to NRSS output. Also NRSS inputs can be selected as one of the sources for an input pipe.

3.3.10.2 Transport Output Ports

Two transport output ports are provided on shared GPIO A pins: High Speed Data Port and 1394 LINK Layer Controller Port. Both of them can work in parallel or serial mode, but only one at a time, since they share the same GPIO pins.

High Speed Data Port (HSDP)

High Speed Data Port is a 12-pin parallel or 5-pin serial standard (CLK, START, VALID, ERROR and DATA) transport output port. One single program transport stream from any of the four pipes can be directed to this port.

In order to preserve bit-rate information, time stamps are used internally to determine the presentation time of the transport packet on this port.

In bypass mode, the selected input port pins directly drive the output pins.

1394 LINK Layer Controller Port

This port is meant for interfacing with the TSB42AA4/TSB42AB4 (ceLynx) Link Layer Controller in single stream half-duplex mode. It is a 13-pin parallel or 6-pin serial (i.e. no address pins for buffer addressing necessary for multistream mode) I/O port. Modes A and B of the ceLynx chip are supported. Software has to decide when the port is in output mode and when it is in input mode to make sure no contention is caused.

One single program transport stream from any of the four pipes can be directed to this port.

In order to preserve bit-rate information, time stamps are used internally to determine the presentation time of the transport packet on this port.

- 1394 (TNF) Input/Output Port Parallel

  This is a 13-pin input port to interface with TI ceLynx, or any other 1394 chip with the same/similar interface, in single stream/buffer mode (no ADDR pins are present to select the buffer/stream). It can be used in software controlled half-duplex mode with 1394 output port. It has CLK, SYNC, VALID, R/W, AV and DATA signals.

- 1394 (TNF) Input/Output Port Serial

  This is a 6-pin input port to interface with TI ceLynx, or any other with the same/similar interface, in single stream/buffer mode (no ADDR pins are present to select the buffer/stream). It can be used in software controlled half-duplex mode with 1394 output port. It has CLK, SYNC, VALID, R/W, AV and DATA(0) signals.

3.3.11 Recording and Time Stamping

TPS supports extraction and recording of two single program transport streams (SPTS) in the system where one can have up to 4 multi-program transport stream (MPTS).

Software enables recording per PID by writing non-zero value to SPTS_INDX_SEL field in PID context RAM. All packets from selected PIDs will be output to one of the two record buffers (note: it is possible to mix packets from different pipes into the same output buffer, i.e., to function as a transport packet "multiplexer").

In order to support viewing the same program that is being recorded, the MCE has to perform dual pass processing on those packets in order to be able to send the same bytes to two locations. In the first pass the MCE removes the transport layer and sends the PES packets or PSI/SI sections to the corresponding decoder or buffer. In the second pass full transport packet is sent to the recording buffer.

Depending on the scramble_control bits of the input stream, the second pass/recorded packet will also follow the same rule, i.e., it will be copy protected (3DES) if input stream is scrambled.

During the recording, i.e., SPTS extraction from MPTS, no PCR, PTS, DTS values are changed. In order to be able to reconstruct the bit rate that corresponds to the PCR values stored in the SPTS, timestamps are stored with each transport packet. These time stamps are snapshots of the free running 16-bit counter running off recovered STC clock on the input pipe, from which the transport packet is stored. The snapshots are taken at the moment the packet enters the TS_FIFO. The two bytes of time stamp are prepended to each packet together with two more reserved zero bytes in order to make packets DWORD aligned (for trick mode playback).

3.3.12 Time Shifting and Bit Rate Recovery

Recorded transport streams can be played later or while recording is in progress. The latter is known as time shifting. In either case the TPS bus master inputs are used to play these streams. Up to two live streams can be time shifted.

The time shifted transport stream is delivered by stream interface block. At the moment the first packet arrives, time stamp is loaded into the free running (STC) counter. The time stamp bytes are stripped from the trasport packet, and the transport packet is placed in TS_FIFO of the corresponding pipe for normal processing by the MCE. The following packets will be sitting in BM_FIFO until their time stamp bytes match the value of the free running STC counter. At that moment the time stamp bytes are stripped and that packet is passed to the TS_FIFO. It is assumed that the stream interface is fast enough to keep BM_FIFO close to full so that no transport packet will miss its proper time for delivery to the TS_FIFO. This way the original bit rate of the stream is reconstructed so that PCRs, PTSs and DTSs are valid again, and the clock can be reconstructed from that stream as well.

The leak rate function alternatively can be used to recover the bit rate, in which case the time stamp bytes should just be stripped off by the pipe processing the stream.

3.3.13 Trick Mode Playback and Picture Code Monitoring (PCM)

In order to do trick mode playback with time shifted/recorded single program transport stream (SPTS), we need extra information to be stored on top of SPTS. This extra information has to be useful even if the stored SPTS is copy protected/ scrambled.

The Picture Code Monitoring (PCM) circuitry provides means to search for sequence (determined by microcode) of up to 4 bytes within the stream (i.e., to find picture start codes 00000100 on streams in clear or descrambled stream on elementary stream level). The MCE checks the result of the PCM at the end of current packet . We send the snapshot of packet counter for the packet that is carrying the requested sequence of bytes to separate frame buffer index.

We increment the packet count for every packet being sent to the disk (i.e. for all PIDs going to disk) but. Note that picture codes can span the packets so the packet count might be off for 1 with respect of the beginning of the sequence.

3.3.14 Picture-In-Picture (PIP)

From the TPS point of view, Picture-In-Picture is just two sets of PID and PSI/SI filters set on two input pipes (or even on a single pipe if all events being displayed are in the same input transport, i.e., coming from a single tuner). If both events displayed are timeshifted at the same time, then all 4 TPS input pipes will be used/programmed — two of them for live streams and two of them for playback streams.

3.3.15 Packet Substitution (SUBS_PID)

TPS supports transport packet substitution on up to 2 PIDs in the system. Through the TPS_SUBST_PID_CNTL register, the software can configure on which input and on which PID we want to do packet substitution.

Substitution can be done in continuous or one-time mode. In continuous mode all packets from the input stream that match marked PID will be replaced by the packet from special packet buffer. In one-time mode one packet from the stream will be replaced and then the function has to be triggered/enabled by software again.

The two buffers with the replacement transport packets are in the MCE DATA RAM. They are loaded by software before enabling the function on corresponding PID or after corresponding SUBST_DONE interrupt. The MCE will burst packet from the buffer and flush the one from the TS_FIFO, if PID match condition happens and the PID is marked as substitution PID.

SUBST_DONE interrupt will be generated by the MCE in the corresponding TPS_INx_INT register, according to the assignment of this function to the corresponding input pipe.

Potentially it is possible to replace the range of packet bytes if from and to count bytes are put in front of each packet and if the MCE microcode is modified to interpret those bytes accordingly.

3.3.16 Clock Recovery

Clock recovery is done by extracting the PCR fields from the incoming transport packets and checking them against the current STC. The value of the STC and PCR are latched for further processing. Low pass filtering can be performed in software or hardware to determine whether the STC clock needs to be pulled and the appropriate pulling is applied to the clock source.

3.3.17 Audio/Video Synchronization

3.3.17.1 Audio STCs

TPS also has two audio STC counters. These counters are assigned to one of the STCs in one of the video groups— the one we want audio to be in sync with. Assigning it to the video STC means referring it to the same PCR to which the video is referenced.

Audio STCs have their own clock recovery and clock pulling loop; however, the same PCR extraction circuitry will be updating/monitoring the corresponding audio and video STCs.

A snapshot of the audio STC counters is taken at the same time the corresponding video STC snapshot is taken, i.e., based on the signal coming from the pipe to which the video STC is attached.

3.3.17.2 Video STCs

TPS has four groups of video STC counters. Two of the groups have four STC counters each (for quad-video support), and the other two have one STC counter each. Each of these four groups can be attached to any one of the input pipes. Attaching a group of STCs to a pipe means selecting which pipe's PCR extraction circuitry will be controlling those STCs. In a group where there are four STCs, one of the STCs must declared as the primary STC. In a group where there is only one STC, the latter is the primary STC by default.

The primary STC is used in the hardware loop for selected clock source pulling. It can also be used for generating timer/early notifier (STC_VAL_REACHED) interrupts when it reaches preset value. The secondary STCs in a multi-STC group will be running off the same clock the primary STC is running off; however, no clock pulling based on a secondary STC is feasible.

To reduce jitter introduced by MCE processing four inputs, snapshots of each STC are taken at the moment the PCR byte enters the FIFO. This is done for every incoming packet regardless of whether it carries the PCR or not. The snapshots are put in the three-entries-deep snapshot FIFO. Only if the packet is a PCR_PID packet will the STC snapshot be used for clock phase error calculation; otherwise the snapshot will be discarded. The position of the byte at which the snapshot of the incoming packet is taken is controlled by software.

For genlocking purposes, a separate snapshot of the primary as well as of any secondary STC can be taken from the signal of any of the two display engines (i.e., there is a snapshot register and a corresponding STC selection per display).

STC counters are accessible to software for read and write operations in case software algorithm is used for clock pulling. Writing to an STC counter can be done in two modes— absolute mode and offset mode. In absolute mode the value being written ends up in the counter, whereas in offset mode, the value being written is added to the current value of the STC counter, and the result ends up in the counter.

STC counters can run on a fixed internal clock, an internal PLL clock, and an internal digital or external VCXO clock, the last two being pullable by hardware, software or a combination of both.

3.3.18 Interrupts

The TPS generates one interrupt to the internal MIPS and/or one interrupt to the CPU. These interrupts are OR functions of four interrupts, each coming from one of four stream inputs. Each stream input interrupt is an OR function of 32 interrupts that are the result of MCE monitoring many different things in the stream. Some of the interrupts are hardware generated while some are software interupts from the MCE point of view.

The CPU or MIPS will clear the interrupts that are responsible for servicing, at the beginning of the service routine or at the end of the service routine, depending on whether one wants to have re-entrant ISRs and how one wants to handle interrupt priorities.

Certain hardware generated interrupts related to one of the input streams can be cleared by by writing 1(s) to the position of the interrupt(s) to be cleared in the TPS_IN1{2,3,4}_INT registers. Other interrupts can be cleared by clearing their corresponding sources. Once all sources of interrupts of a particular type are cleared, that type of interrupt will be cleared. For example, to clear SFU_INT interrupt on certain input, software has to clear all section filter interrupts in the corresponding TPS_IN1{2,3,4}_SFU_INT register. These interrups are cleared the same way by writing 1(s) to the position of the interrupt(s) that is supposed to be cleared.

3.3.19 Clock Speed and Latencies

The TPS has three clock domains:

- Transport input clock (bit or byte)
- XCLK for all the core functionality
- 27MHz clock for the STC counter

There are several sources of latencies in the TPC:

- Accessing the filter conditions
- Saving/loading the context
- Processing the stream in the parser
- CRC engine (minimal)
- CA module (this module works on blocks of bytes)
- Flushing the client FIFOs (PCI and FB access priorities)

3.4 Conditional Access Module

The CA module incorporates conditional access descramblers and copy protection scramblers/descramblers. It directly couples to the transport demultiplexer for transport stream descrambling.

3.4.1 System-Level Description

As shown in *Figure 3-8*, when Xilleon 220 receives a TS, the latter is passed through one of the four transport demuxes. The transport demux extracts the transport packet header and adaptation fields and passes scrambled payloads to the CA module. Using the CW (Control Word) transferred from a CA smart card interface via the microprocessor, the CA module descrambles the payload and passes the descrambled payload back to the transport demux for further processing.

Depending on the type of cipher employed, CWs for the descrambling process are retrieved either from a CA smart card or from the host CPU. To retrieve CWs from the CA smart card the transport demux must filter the PID's corresponding ECM from the transport multiplex and pass them to the microprocessor. The microprocessor transmits the filtered ECMs and EMMs to the CA smart card through the smart card interface. The CA smart card transmits a CW back to the smart card interface in response to the EMM/ECM messages. The microprocessor then obtains the CW and passes it to the CA module so that the latter can generate the actual cipher key.

In systems that use an NRSS type interface, such as OpenCable, CP (copy protection) Keys are exchanged between the POD (Point of Deployment) module and the descrambler via the host CPU. The host CPU may either exchange CP Keys with the POD module or act as a conduit between the POD module and the descrambler. If the host CPU exchanges CP keys, then the CP keys must be passed to the CA module through another cryptographic key exchange in order to satisfy licensing requirements. If the host CPU acts as a conduit, the microprocessor must exchange CP Keys with the POD module.

3.4.2 CA Module Features

The CA module deciphers scrambled transport stream packet payloads. The features of this block are listed below:

- 32 control word pairs
- 8 initialization vector pairs
- 4 PID mask control word pair per transport stream input with matching initialization vector pair.
- 4 default control word pair per transport stream input with matching initialization vector pair.
- DVB-CSA (Common Scrambling Algorithm) Version 1.1 for embedded DVB CA.
- CSS (Content Scrambling System) for DVD playback.
- Single DES descrambler in CBC mode for CA with SCTE DVS 042.

- Single DES descrambler in ECB mode for NRSS, DirecTV and OpenCable copy protection.
- ABC EDE Triple DES descrambler in CBC mode for ATSC CA.
- Triple DES (TDEA) descrambler in ECB mode for copy protection.
- MULTI2 descrambler in CBC and OFB mode for Japanese CS/BS Digital CA.

3.4.3 Top-Level Description

The CA module is physically instantiated once within the Xilleon 220 and consists of a control block and five different types of ciphers: DVB-CSA, MULTI2, CSS, 3DES and DES. The DVB-CSA, MULTI2 and CSS ciphers can only descramble whereas the DES and 3DES ciphers may either scramble or descramble depending on their programming. Based on the fact that the transport demux transmits one transport packet payload at a time after the input mux (see *Figure 3-8*), the CA module only operates on one scrambled packet at time. This eliminates the need for context switching within the CA module.

When the transport demux sends a packet to the CA module it also sends the PID, the transport pipe, the scrambling_control bits and the operation to be performed. The CA module will use this information to select the corresponding CW and, if required, Initialization Vector (IV), from the CW and IV Tables. In addition to this the CA module will determine which cipher type and cipher mode to use on the packet payload. Once the cipher has been initialized with mode information, CW and IV it is ready for operation.

If a packet is to be both CA descrambled and CP scrambled, then the CA module will pass each block (64-bits) of payload to the CA descrambler and output descrambled block back to the transport demux. During this process the CA module will also send the descrambled blocks to the programmed CP scrambler for rescrambling. The blocks output by the CP scrambler are then stored in the Scramble FIFO. Once the entire packet has been CA descrambled and sent to the transport demux, the CA module will read the CP scrambled packet from the Scramble FIFO and send it to the transport demux. This two-pass method eliminates the need for transport demux to resend the descrambled packet back to the CA module for copy protection, thereby increasing the efficiency of the transport demux.

It is possible to bypass the CA module and record the CA scrambled transport packets onto a storage device (e.g. hard drive). The CA scrambled transport packets may be descrambled just prior to playback this stream. Note that this requires the CA software stack to send ECMs and EMMs to the smart card in a timely manner especially during trick mode playback.

3.4.4 Descrambling Keys Prior to Storage in the CW Table

The CA module has the ability to DES or TDES descramble keys prior to storage into the CW and IV Table with a set of keys selected from 16 secret internal keys embedded within the Xilleon 220. Since the CW and IV Tables are write-only from the software perspective, the previously scrambled key cannot be extracted from the CW and IV tables.

This feature is particularly useful if the STB manufacturer requires each STB to have a unique key for CP scrambling transport stream onto a hard drive. Scrambled CP keys may be stored in a ROM within the STB and sent to the CA module after power-up or scrambled CP keys may be sent via the network to each STB to maintain control over each box.

The 16 secret internal keys are published in a separate document.

3.5 Smart Card Interfaces

3.5.1 Introduction

The Xilleon 220 contains three fully independent ISO 7816 compliant smart card interfaces to control external interfacing to three smart card readers. Each smart card interface connects to the stream interface to bus-master data to and from the smart card and the frame buffer. Each smart card interface may also generate an interrupt to either the MIPS core or the PCI bus.

In general, two of the smart card interfaces may be connected to two independent CA smart cards, and the third may be connected to a banking smart card. However, if only one or two smart cards need to be connected to the Xilleon 220, the balls for the other smart card interfaces may be used as general purpose I/Os.

The smart card interface register names start with the prefix SMC_0, SMC_1, or SMC_2, and they correspond respectively to ball names SMyyyA, SMyyyB, and SMyyyS, where yyy is the function name of the ball. For example, register names that start with the prefix SMC_1 control the ball SMDATAB. For simplification, the register names have been written as SMC_x, indicating that all smart card interfaces have the register. Similarly, the ball names have been written as SMCyyyx. For example, SMDETECTx represents SMCDETECTA, SMDETECTB, and SMDETECTS.

3.5.2 Feature List

The smart card interface features are as follows:

- ISO/IEC 7816, EMV 2000 and EIA-679B (NRSS) Part A compatible.
- Asynchronous transmission and reception.
- Automatic card detection/removal with activation/deactivation.
- Programmable card detection level to support normally-open and normally-closed smart card connectors.
- Programmable smart card clock.
- 5 ball control (SMDETECTx, SMVCCx, SMCLKx, SMRSTx, SMDATAx).
- Wait and guard-time management and monitoring.
- Transmit/receive error detection with programmable number of character transmission retries.
- Clock stopping on high or low supported for low power.
- 32x8-bit FIFO with watermark levels.
- Maskable interrupts for the following: Character/Block wait time overrun, character transmission/reception, receive/transmit parity error, LRC/CRC error, card detect/remove, FIFO empty/full.
- Maskable balls for overriding smart card I/O balls
- Flow control support.

3.5.3 Functional Description

3.5.3.1 Block Diagram

The block diagram below shows the major blocks within the smart card interface.

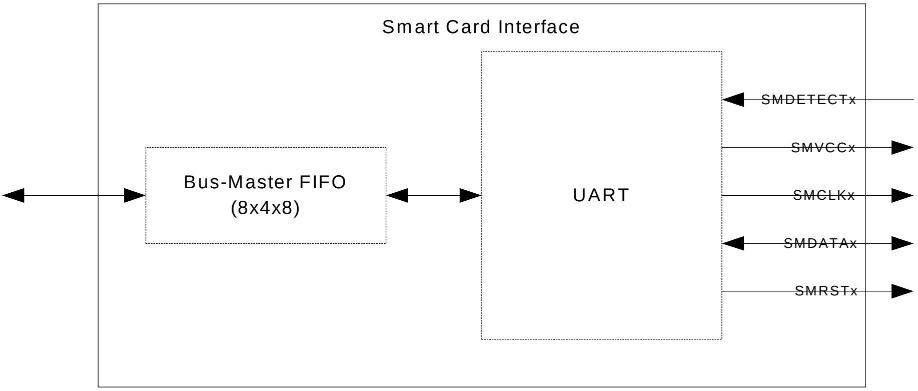


Figure 3-16 Smart Card Block Diagram

3.5.3.2 Theory of Operation

Each smart card interface provides the UART logic specified in the ISO/IEC-7816 standard to support protocol types T=0 and T=1. Smart card interfaces A and B may optionally be associated with one of the two high-speed data interfaces connected to a serial transport demux input to support EIA 679-B Part A (NRSS) compliant smart cards. The association with a serial transport demux input is only required to satisfy the smart card activation sequence described in EIA 679-B Part A.

Smart Card Interface Modes

The smart card interface operates in three different modes specified in the MODE field in SMC_x_CTRL. The table below describes the different modes.

Table 3-13 Smart Card Interface Modes

Mode	Mode Name	Description
0	GPIO	All balls of the smart card interface are controlled directly through GPIO_SMC_x_PAD_Y and GPIO_SMC_x_PAD_EN registers.
1	Smart Card	This mode provides all of the functions of a normal ISO/IEC smart card interface. Control over the smart card interface signals, however, may be overridden. If a field in GPIO_SMC_x_PAD_MASK is set to 1, the corresponding signal is controlled by GPIO_SMC_x_PAD_EN and acts like a GPIO. Otherwise the signal is controlled by the smart card interface logic.

Table 3-13 Smart Card Interface Modes (Continued)

Mode	Mode Name	Description
2	Analog IC	This mode still provides the UART functions described above however it uses an external chip, such as a TDA8004 IC card interface, to communicate to the smart card. The external chip controls the activation and deactivation sequence and provides the smart card clock through SMCLKx. In this mode SMVCCx controls the CMDVCC pin and the rising and falling edges of SMRSTx controls the activation of the actual clock/reset pin of the smart card.[a]

[a] See data sheet for TDA8004, IC card interface, Philips Semiconductors

Before the smart card interface is enabled the clock frequency, the default wait and guard times and the clock to reset and reset to ATR times must be set.

Smart Card Interface Initialization

Initializing the CLK_DIVISOR field in SMCx_CLK with the value that satisfies the following equation will set the smart card interface clock frequency (Fsci).

$$CLKDIVISOR = \frac{Fmemory}{2 \times Fsci}$$

Therefore, if CLK_DIVISOR is zero, SMC_x_CLK frequency will be half of the memory clock (Fmemory).

Bit duration time in SMC_x_CLK cycles unit is called elementary time unit (etu). The ETU_LEN field of SMC_x_CLK specifies the length of an etu in smart card clocks. An etu is initially measured and written to this field by the smart card interface by calculating the length of the first start bit of the answer-to-reset (ATR). If necessary, the ETU_LEN value may be changed after the ATR has been received.

The DETECT and VCC balls can be active low or high. The DETECT_LEVEL and VCC_LEVEL fields of SMC_x_CTRL specify whether the SMDETECTx and SMVCCx balls are active low or high.

Once the smart card interface is enabled and a smart card is detected, the SMVCCx, SMCLKx and SMRSTx balls are activated in a defined sequence (ACTIVATE field in SMC_x_CTRL must also be set to 1). The time interval between SMVCCx and SMCLKx activation is specified by the sum of the VCC2NRSS and NRSS2CLK fields in SMC_x_CONFIG. The field VCC2NRSS provides the time between SMVCCx and NRSSCLKx signal. The NRSSCLKx signal is controlled by the transport demux block. Similarly, CLK2RST specifies the time interval between SMCLKx and SMRSTx activation; note that the unit is in SMC_x_CLK cycles.

After the fields above are initialized, the smart card interface should be enabled to await smart card insertion. This is accomplished by setting the SMC_x_ENABLE field of the SMC_x_CTRL register.

Smart card insertion or removal is detected by the assertion of the STATUS_CARD_MOVEMENT field in the SMC_x_STATUS register. Once a smart card has been detected, the activation sequence may be started by setting the ACTIVATE field of the SMC_x_CTRL register. The cold reset activation sequence is shown in the figure below.

To warm reset the smart card, toggle the WARM_RESET field of SMC_x_CTRL from one to zero. Note that before performing warm reset, ETU_LEN must be set to the previously measured length (372 for most cards).

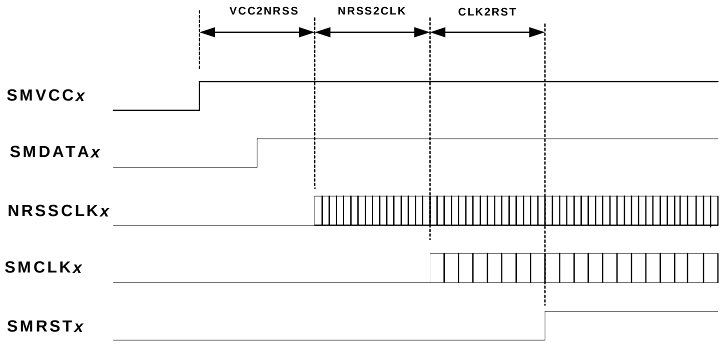


Figure 3-17 Smart Card Cold Reset Activation Sequence

Smart Deactivation

A smart card may be deactivated by clearing the ACTIVATE field. In emergency situations a smart card will be deactivated automatically by the smart card interface. The following conditions are considered emergency situations.

- If the smart card is removed.
- If number of transmit retries is exceeded.
- If an ATR is not received within RST2ATR_MAX smart card clocks.
- If ATR does not start with 3Bh or 3Fh.

If the smart card is deactivated in an emergency situation, the smart card interface will set the STATUS_REJECT_CARD field in SMC_x_STATUS. The deactivation sequence is shown below.

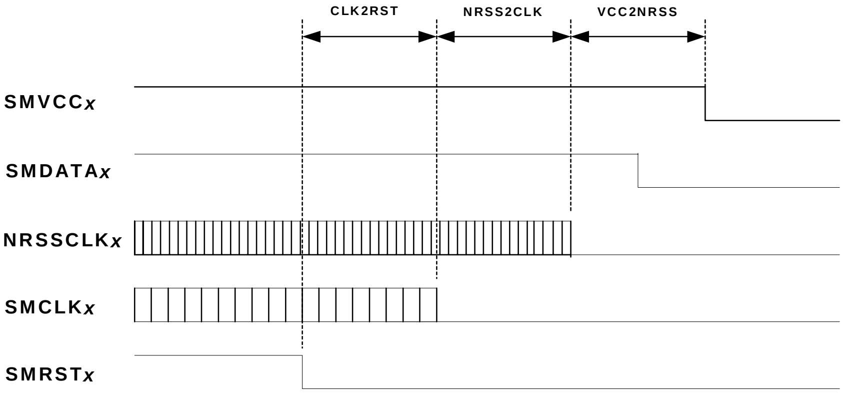


Figure 3-18 Smart Card Emergent Deactivation Sequence

Answer to reset (ATR)

After the SMRSTx signal has been asserted, the smart card should respond by sending an ATR. If it does not respond within the maximum amount of time after the rising edge of reset, the smart card interface will set the STATUS_ATR_NOT_RXED field in the SMC_x_STATUS register and reject the smart card.

The maximum amount of time between the rising edge of SMRSTx and the first start bit of ATR must be specified in the RST2ATR_MAX field of SMC_x_ATR. Similarly, if the smart card responds before the minimum amount of time after the rising edge of reset, the smart card interface will set the STATUS_EARLY_ANSWER field in the SMC_x_STATUS register. The minimum amount of time between the rising edge of SMRST and the first bit of the ATR is specified in the RST2ATR_MIN field. All units are in smart card clock cycles.

The format of the character byte (TS) of the ATR will define the convention the smart card supports. The convention is stored in the CONVENTION field of the SMC_x_STATUS register. The timing of the character byte defines the elementary time unit (etu). The smart card interface automatically calculates the etu in unit of smart card clocks and stores the value in ETU_LEN of SMC_x_CLK.

Guard and Waiting Times

Character guard time specifies the minimum amount of time permitted between to subsequently transmitted characters. The CGT field in SMC_x_GUARD_TIME sets the character guard time. Block guard time specifies the minimum distance between two subsequent blocks in protocol T1, when the interface is in transmit mode. The BGT field in SMC_x_GUARD_TIME sets the block guard time. Both fields are measured in etus.

Character waiting time specifies the maximum distance between two subsequent start bits. The CWT field in SMC_x_CWT sets the character wait time. Block waiting time specifies the maximum distance between two subsequent blocks. BWT field in SMC_x_BWT sets the block wait time. If the interface is in the receive mode and the delay between two characters or two blocks is exceeded the corresponding flag of SMC_x_STATUS register will be raised.

Interrupts and Status Clearing

All the fields in the SMC_x_STATUS register, except FIFO_DEPTH and CONVENTION, may interrupt either the MIPS core or the PCI bus. A particular status field can only generate an interrupt if the corresponding field of the SMC_x_INTR_MASK register is set. To determine the source of an interrupt, the GEN_INT_STATUS (or MIPS_STATUS, if using the MIPS) register is read to see if the PCU_INT field has been set. If it has been set, then the PCU_STATUS register must be checked to determine which smart card interface generated the interrupt. If it is determined that one of the smart card interfaces generated the interrupt, the individual fields of SMC_x_STATUS register must be checked.

Status fields in the SMC_x_STATUS register are set only by the smart card interface, and are cleared afterwards only by a register write. To clear a status field, a '1' must be written to that field. If a status field is '0', writing a '1' to that field does nothing. The STATUS_RX_PARITY_ERROR and STATUS_BYTE_RXED fields in the SMC_x_STATUS register will also be cleared by reading the SMC_x_RX register. The STATUS_TX_PARITY_ERROR and STATUS_BYTE_TXED fields in the SMC_x_STATUS register will also be cleared by writing the SMC_x_TX register.

3.5.3.3 Data Flow

This section describes the methods for communicating to the smart card. Communication with the smart card is performed through direct register accesses or through bus-master accesses. Direct register access mode is enabled when BM_ENABLE is cleared, and bus-master access mode is enabled when BM_ENABLE is set.

Direct register accesses must service each character as they are transferred to or from the smart card. This method requires the MIPS core or the peer processor to be interrupted upon the reception or transmission of each character. Bus-mastering allows larger character blocks to be directly written to, or read from the frame buffer or system memory via the stream interface block. Only after a character block has been transferred, will the smart card interface interrupt the MIPS or the peer processor for servicing.

To maintain memory efficiency while in bus-mastering mode, each smart card interface contains one 8x4x8 bus-master FIFO. Since communications with the smart card are half-duplex, the FIFO is used in both receive and transmit modes. The FIFO stores data until it has been filled, until an end of data block signal is indicated, or until an error is detected. The FIFO_DEPTH field in SMC_x_STATUS indicates the current number of DWORDS stored in the FIFO. The STATUS_FIFO_EMPTY and STATUS_FIFO_FULL fields are set by the smart card interface to indicate FIFO emptiness and fullness respectively.

The smart card interface supports protocol types T=0 and T=1, as controlled by the PROTOCOL field in the SMC_x_CTRL register. The default is set to T=0 so that the smart card interface may receive the ATR from the smart card. If inverse convention is used for transmission, the smart card interface will invert and reverse the order of the bits, so that bytes only need to be read or written in direct convention.

Direct Register Receive (RX) Mode

TX_REQ field of the SMC_x_TX register determines the receive/transmit mode of the smart card interface. To set the smart card interface to receive mode, the TX_REQ field must be set to a '0'. Direct register receive mode is set when the BM_ENABLE field in the SMC_x_CTRL register is cleared. This mode is the default mode for the smart card interface.

When the smart card interface receives a byte from the smart card, it will set the STATUS_BYTE_RXED field of the SMC_x_STATUS register. The smart card interface clears this field when the received byte has been read. If another byte is received before the previous received byte has been read, the STATUS_RX_OVERRUN flag will be raised. This field may be cleared only by a write to SMC_x_STATUS.

If a parity error is detected while receiving in protocol T=0, the smart card interface will set the STATUS_RX_PARITY_ERROR field in SMC_x_STATUS, and automatically trigger the smart card to resend the character. If a read parity error occurs more than NUM_OF_RETRIES times, the smart card interface will set the STATUS_REJECT_CARD field, and carry out the deactivation sequence. Parity errors are not checked for while receiving in protocol T=1; however, STATUS_CRC_LRC_ERROR field is set by the smart card interface if the calculated LRC/CRC does not match the received LRC/CRC of the block.

STATUS_WT_OVERRUN is set when either character waiting time or block waiting time is overrun by the smart card, and it may only be cleared by writing to the SMC_x_STATUS register.

Direct Register Transmit (TX) Mode

To transmit a byte to the smart card in protocol T=0, the byte must be written to the SM_TX_BYTE field and a '1' must be written to the TX_REQ field. Both fields are part of the SMC_x_TX register. When the byte has been transmitted to the smart card, the STATUS_BYTE_TXED field of SMC_x_STATUS register will be set by the smart card interface. Writing in SMC_x_TX field clears the STATUS_BYTE_TXED field automatically. TX_REQ field is cleared by the smart card interface every time a byte has been sent to the smart card.

If a parity error is detected by the smart card, the smart card will report this by setting the STATUS_TX_PARITY_ERROR field. The same byte must resent by setting TX_REQ field to a '1'. If the parity error is detected more than NUM_OF_RETRIES times, the smart card interface will set the STATUS_REJECT_CARD field, and carry out the deactivation sequence. Parity errors are not checked for while transmitting in protocol. Parity errors are not checked while transmitting in protocol T=1.

Bus-Mastering Receive (RX) Mode

Before transmitting FIFO contents in this mode, the SYSTEM_BUFFER field of the SMC_x_RX register and the SG table describing the destination system buffer must initialized. The SYSTEM_BUFFER value tags the character block to the SG table describing the location, depth, and other parameters of the destination system buffer. See the stream interface block for more information on system buffers and their initialization.

While receiving a character block from the smart card in bus-mastering mode, the smart card interface stores the characters in the bus-master FIFO, instead of sending them individually to the RX_BYTE field of the SMC_x_RX register. When the FIFO is full, or the end of character block is indicated, or when an error has been detected during reception, the FIFO contents are transferred to the stream interface block for storage in the destination system buffer. In protocol T=1, the end of character block is indicated by a comparison between the number of characters read and the LEN field in the block. In protocol T=0, the character waiting time overrun signal is used for indication of the end of a character block. When the end of a character block is indicated, the STATUS_BYTE_RXED will be set.

Bus-Mastering Transmit (TX) Mode

Before transmitting a character block to the smart card, the BM_COUNT field and the SG table describing the source system buffer must be set. The BM_COUNT value indicates the number of DWORDs to be transmitted to the smart card, and the SG table describes the location, depth and other parameters of the source system buffer. See the stream interface block for more information on system buffers and their initialization. Note: The burst size set in the stream interface's SG-table must be set to 4 or 32, or the total number of bytes to be bus-mastered to the FIFO if it is less than the depth of the FIFO.

To initiate transmission, the TX_REQ field must be set. This will trigger the stream interface block to read the source system buffer contents and store the characters into the bus-master FIFO. The smart card interface will then start transmitting the characters one at a time to the smart card. Once all the bytes in the FIFO are sent to the smart card, the STATUS_BYTE_TXED field is set by the smart card interface. The STATUS_BYTE_TXED field or interrupt may be used to reset the BM_COUNT field for subsequent transmissions.

3.5.4 External Chip Interfaces

In order to voltage convert and ESD protect the smart card, interfaces should be connected to an analog IC card interface such as a TDA8004. They could also be connected to discrete components to provide similar conversion and protection. The external interface connections are shown in the figure below.

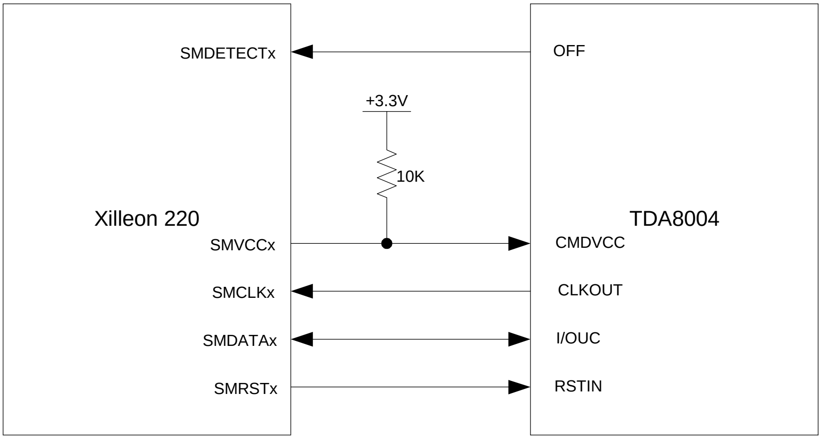


Figure 3-19 Analog IC Card Interface

3.5.5 Performance

The smart card interface only supports clock frequencies between *(1/2048)xFmemory* and *(1/2)xFmemory*, by increments of *(1/2048)xFmemory*. Since smart cards currently transmit only at a maximum of 20 MHz, the programming of CLK_DIVISOR must be done carefully.

3.6 MPEG Decoder

3.6.1 Features

Xilleon 220 supports dual high definition MPEG decoding using two MPEG decoders. These decoders are capable of decoding MPEG1 & MPEG2 video streams, and have the following features:

- PIP or up to 8 x standard definition MPEG-2
- Watch and record.
- Support for 1920x1080 30I and 24p non-lossy decode in 16MB memory.
- MPEG video passes to output with no color space conversion.
- Full support for user data and all other extension fields in the MPEG stream.
- Synchronization ability to allow locking software activities with the decoders' STC clock.
- Full hardware MPEG-2 MP@ML and MP@HL decode and MPEG-1 decode.
- Adaptive image compression techniques to support 1920x1080 decode in 8MB frame buffer configurations.
- Decode and display of all 18 (36) DTV ATSC MPEG-2 formats.

3.6.2 Functional Description

3.6.2.1 Block Diagram

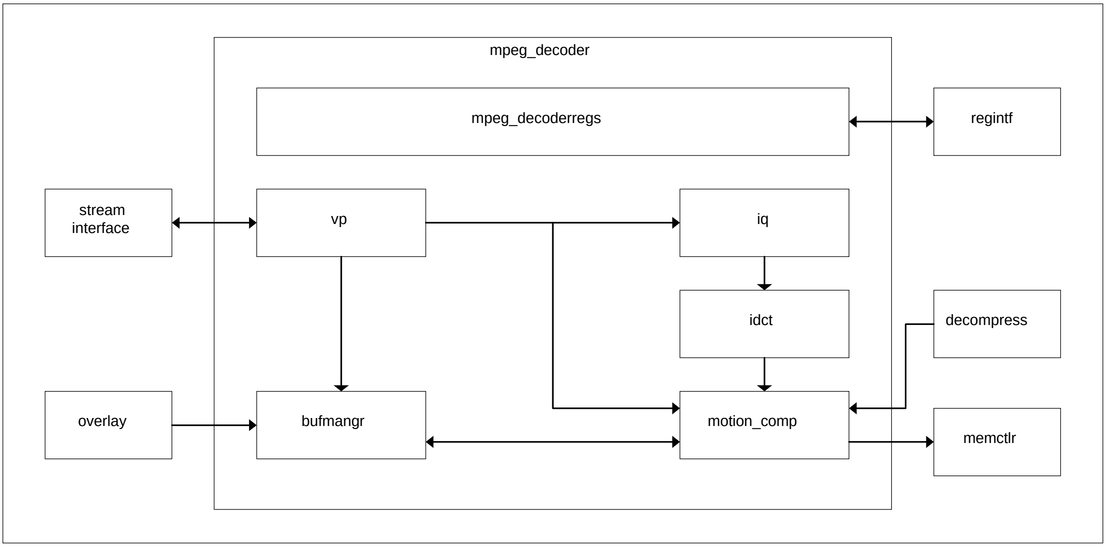


Figure 3-20 Top Level MPEG Decoder Block

3.6.2.2 Description of MPEG Block Components

The MPEG decoder, shown in *Figure 3-20* above, consists of the following major blocks: Video Parser (vp), Inverse Quantiser (iq), Inverse Discrete Cosine Transform (idct), Motion Compensation (motion_comp), Buffer manager (bufmangr), and MPEG Decode Registers (mpeg_decoderregs).

Video Parser (vp)
The Video Parser implements the ISO/IEC 13818-2 standard to the MP@HL specification. It is compliant with the ATSC/DVB video standards. The Video Parser accepts an incoming MPEG PES or ES stream from the stream interface. It is responsible for the following operations:

- Latches all fields that exist from the video sequence layer down to the picture layer of the MPEG-2 video specification. These bits are used to configure the picture decode operation and to configure the display sub-system.
- Parses out user_data from the incoming stream. It then puts this data into the User Data FIFO so that it can be accessed by the register reads.
- Triggers interrupts when user_data, extra picture information, or extra slice information is found in the incoming stream.

- Parses out MPEG extension data (pre-ambled by a reserved start code). It then puts this data into the User Data FIFO so that it can be assessed by the register reads.
- Detects and conceals errors. This includes such errors as skipped macroblocks and concealment vectors.
- Picture level decoding is completely hardware controlled.
- Decoder pipe has a maximum throughput of 1 pel per clock.
- Maintains comprehensive status of the decoding process. This includes latching the types of errors detected.
- Compressed video buffer is stored in the frame buffer.
- Latches the PTS from the PES packet.
- Supports trick modes using both HW/SW.
- Supports context saving and restoring the state of the parser to allow for <n>decode operation.

Inverse Quantiser (iq)
The Inverse Quantisation block implements Clauses 7.3 Inverse Scan, and 7.4 Inverse Quantisation of the MPEG-2 specification. This block takes the incoming stream of coefficients (runs and levels) produced by the Video Parser and dequantizes the values on the way through to the IDCT engine. This block input data from the Video Parser undergoes the following process: reconstruction, saturation, and mismatch control.

Inverse Discrete Cosine Transform (idct)
The IDCT block implements Clause 7.5 Inverse DCT of the MPEG-2 specification. It takes in 8x8 blocks from the Inverse Quantiser and performs the inverse discrete cosine transform function. This function converts the blocks from the frequency domain to the spatial domain. The resulting 8x8 block passes the IEEE 1180 IDCT test. It supports a throughput of up to 1 pel/clock. It also allows for pipe stalling from both ends.

Motion Compensation (motion_comp)
The Motion Compensation block performs the following tasks:

- Receives motion information from the Video Parser and determines the predictions.
- Generates the read addresses for the predictions, and the control signals for the memory controller.
- Creates tag information that is used to process and store the read data into a read buffer.
- Aligns the data as it comes back from the frame buffer, performs half-pel calculations, and writes the result into an internal read buffer using information that is stored in the tags.
- Combines the forward and backward predictions and adds in the error terms from the IDCT.
- Compresses the data if the system is in adaptive compression mode.
- Stores the decoded macroblock into an internal write buffer.

- Generates the write addresses and the control signals for the memory controller to write the decoded macroblock into the frame buffer.

Buffer Manager

The buffer manager provides hardware support for the SW driven buffer assignments. As such it performs the following tasks:

- Throttles the decoder at the end of the picture header processing pending the SW update for the decode buffer assignments.
- Throttles the decoder at the macro-block row level to allow for decoding behind the raster scan.

A minimal decoder requires 3 decode frame stores in order to decode an arbitrary I,P,B picture sequence. The Xilleon 220 allows a software element to determine these assignments.

At any time the decoder has 3 active frame stores. These are:

- Forward buffer: defines the frame store from which the forward predictions are taken from.
- Backward buffer: defines the frame store from which the backward predictions are taken from.
- Decode buffer: defines the current frame store to decode into.

During the decode process the allocation of buffer assignments proceeds as shown in the following diagram.

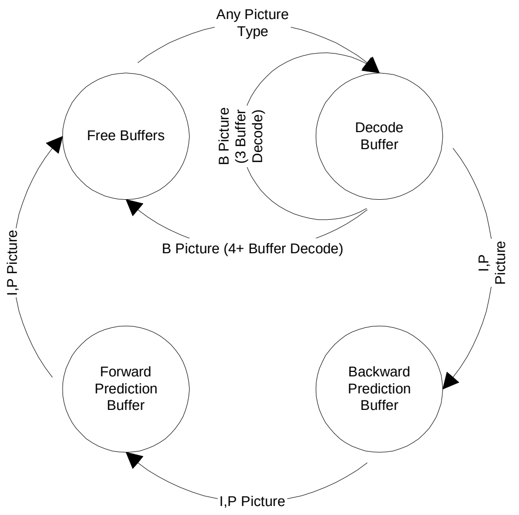

Any Picture Type
Free Buffers
B Picture (3 Buffer Decode)
Decode Buffer
B Picture (4+ Buffer Decode)
I,P Picture
I,P Picture
Forward Prediction Buffer
Backward Prediction Buffer
I,P Picture

Figure 3-21 Allocation of Buffer Assignments

3.6.2.3 Data Flow

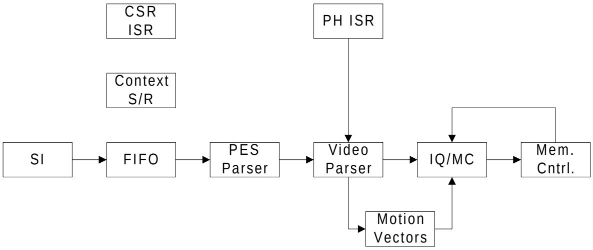


Figure 3-22 MPEG Decoder Data Flow

The flow of data through the decoder begins at its interface with the Stream Interface (SI). The input fetch mechanism makes requests of the SI in order to keep its input FIFO full. This input FIFO in conjunction with the buffer managed by the stream interface consists of the compressed video buffer.

From its input FIFO, data is passed through the PES parser. This element strips off the PES header and provides the decode algorithm with the PTS values to be associated with the next picture header. A status bit indicates the presence of new PTS information.

Data moves from the PES parser to the video parser. The video parser processes the incoming stream and verifies correct syntax and field values. Any deviation from the specification is noted, allowing for picture level error concealment algorithms to be implemented in software. It reports the presence of all stream elements above the picture header and latches their values in registers. It is also responsible for extracting all user/picture extra data and MPEG reserved extension data embedded in the incoming video stream.

At the occurrence of a picture header, the video parser will generate an interrupt to the software element in order to allow the decode buffer assignments to be made by the ISR.

Upon the ISR assignment of the decode buffers, the video parser continues processing the sub-picture layers of slice and macro-block. The elements parsed from the data stream are passed through an inverse quantization stage, the IDCT, and ultimately the motion compensation block. During this level of processing, any sub-picture errors are identified and actions are taken on these as specified by the enabled error concealment settings. These actions can include the use of concealment motion vectors when the stream is so enabled.

At the end of the picture decode, the context save/restore mechanism is triggered. This allows for the swapping of decode contexts required for time-multiplexing of the decoder. This swap action is triggered by the hardware and uses the settings configured during the picture header processing. At the end of the context save/restore, decoding resumes with the video parser looking for the next picture header. The Xilleon 220 requires an interrupt at the context save/restore point. Future implementations will remove this requirement to minimize software overhead.

3.6.2.4 External Interfaces

The MPEG decoder communicates with the following external blocks: register interface, stream interface, overlay interface, decompress interface, and memory controller interface.

Register Interface
The MPEG decoder core communicates with the host CPU through the register interface. The CPU monitors the status of the decoder core by reading the status registers, and manipulates the decoding process through the control registers. The decoder core provides access to all the MPEG specified stream fields above the picture level through the register level as status. Several of these fields can be software over-ridden. The Xilleon 220 register specification should be referenced for details concerning this operation. The register interface supports a 32-bit wide data bus and operates at the XCLK speed. The register block re-synchronizes the data to the BCLK speed.

Interrupts — The MPEG decoder core can generate interrupts based upon a variety of sources. These are selected by configuring the interrupt control register. Using these interrupts, software is provided with a flexible interface allowing for low latency servicing of data processing. The interrupt status register reflects the current status of the interrupt sources. The status bits remain set until the software clears the relevant interrupt status bit.

Stream Interface
The MPEG decoder receives its compressed stream from the Stream Interface block. The source of this data can be from either the transport parser or from a software generated source. The stream interface block manages the input buffer. It can be either circular or linear. The burst size for the compressed MPEG data is 128 bytes. There is a mechanism to flush the data within the stream buffer if it doesn’t contain a full 128 bytes of data.

Overlay Interface
There are several overlay/display interfaces. Some are direct hardware connections, while others are software implemented.

- The video decoder is slaved off the refresh timing of one of the display outputs. This locks the picture rate to that of the display, which in turn is locked to the STC time base. In the case of multi-decodes on a single hardware decoder instance, the non-focal decodes will frame slip in order to achieve lock with the display, which is locked to the focal decode. This locking and slipping is achieved through a software link.
- The video decoder ISR creates a display list which consists of the display information for each of the decode buffers in sequence (display sequence). This information is passed to the overlay ISR to guide its processing of the decoded data. The information passed consists of data related to field characteristics, chromacity, display duration and 3:2 pulldown characteristics.
- The overlay generates a signal that indicates the progression of the overlay through its active time. The decoder uses this signal to gauge its decode rate when running in 3 buffer

mode. This allows for decoding behind the display. The signal is hardware generated and acted upon, but the control for enabling both generation and processing is done by software.

Memory Interface
The MPEG decoder communicates directly to the memory controller when fetching references for motion compensation, and when writing the results of the decoding process. The memory interface supports a 64-bit wide data bus. The memory controller arbitrates the requests from the multiple requestors.

Decompress Interface
The decompress block returns 128-bit wide data to the motion compensation block. In quality mode, 64 bits of data are returned on the lower 64 bits of the bus. In memory_enhanced mode, 128 bits of data are returned.

3.6.3 Adaptive Filter Compression Decode Support

The Xilleon 220 MPEG video decoder supports reduced memory decoding of the video elementary stream. It does this in order to minimize memory bandwidth and footprint, both of which are critical set-top box design issues. Adaptive filter compression decode implies that the resulting image quality is somewhat less than that of a direct 1:1 decode. However, the quality level is still satisfactory for playback on SDTV devices. In the case of HDTV display devices, a 1920x1080 reduced memory decode may appear lesser in quality than a 1280x720 non-reduced memory decode, but the reduced memory decode is necessary in order to decode within a 8MB memory restriction. The Xilleon 220 implements a proprietary 2:1 adaptive filter compression algorithm that has been optimized to achieve the highest image quality over a wide range of different scenes.

Reduced memory decode does not change the throughput of the pipe. What it does change is the memory footprint (50% of non-reduced memory decode) and the memory bandwidth for prediction (~75% of non-reduced memory decode reduction in savings is due to higher memory granularity losses).

3.6.3.1 Decoding Sequence

Consider the following description of the data flow during a typical decoding sequence, assuming that B frames are being decoded and displayed using Adaptive filter compression:

- The stream interface picks up data from the frame buffer and sends it to the video decoder. The data could originate from the transport demux or from any other source.
- Inside the video decoder, the parser extracts the necessary information from the input stream. The parser then sends the parsed data through the decoding pipe and drives the motion compensation block.
- The motion compensation block reads reference macroblocks from the frame buffer (from decoded I and decoded P buffers). The read path goes through the decompress block because Adaptive filter compression is in use.
- The motion compensation block assembles the decoded picture, and writes it out to buffers inside of the frame buffer. Because Adaptive filter compression is in use, the

writing out of the decoded picture is compressed by the compress block inside of the video decoder.

- The display picks up the decoded picture from the frame buffer. Because Adaptive filter compression is in use, the display has to read through the decompress block as well.
- In *Figure 3-23* the decode and the display are using the same buffer. An interlocking mechanism is in use that prevents the decode from corrupting the data before it is displayed (labeled “decode behind display”). There is an option to use separate buffers for decode and display (4 buffers), but the memory footprint will increase by one third.
- When adaptive filter compression is not in use, the compress block will be bypassed. The decompress block will operate in pass-through mode.

To handle multiple streams with 1 decoder, the video decoder performs context switching. Its decoding alternates from streams to streams on a picture basis. One decoder can handle up to four streams from any source, (e.g., up to 4 streams from multiple transports).

Only the parser is context switched. Any decoding gone beyond the parser runs to completion without interruption. The context switching is accomplished by saving and restoring vital parser register values to the frame buffer. The decision over whether a switch will occur is made after each picture is parsed.

When a context switch is required, the MIPS processor is only responsible for setting up the transfers. The MIPS processor is not involved in the actual movement of context data. The data movement is handled by the stream interface.

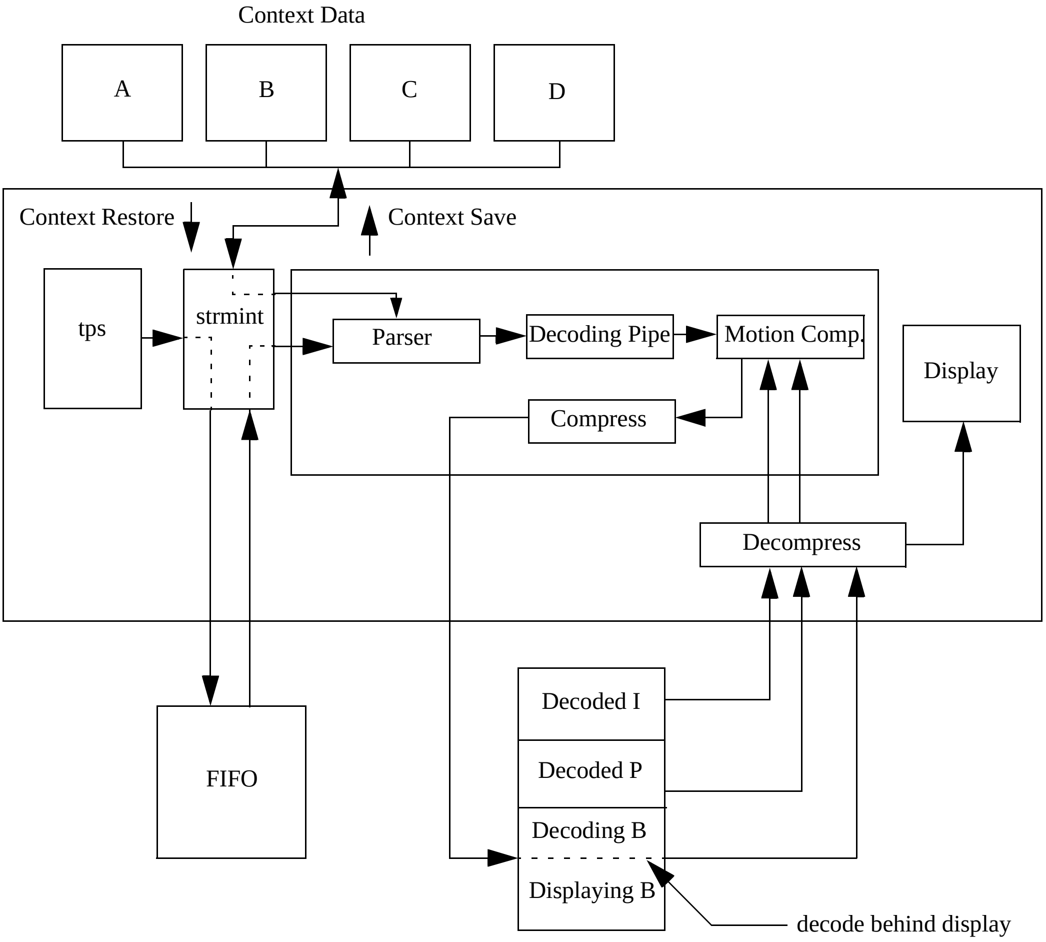


Figure 3-23 Decoding Sequence Data Flow

3.6.3.2 Trick Mode

The decoder supports the following low level controls:

- SEARCH_SH — This control forces the video decoder to search for a valid sequence header by consuming the input data stream at up to 60MB/s. Once it is found, the decoder will proceed as defined by the PLAY control.
- SEARCH_GOP — This control forces the decoder to search for a GOP without errors by consuming [the input data stream] at up to 60MB/s. Once a GOP is found, the decoder will proceed as defined by the PLAY control.
- SEARCH_I_PICTURE — This control forces the decoder to search for a I - picture

header without error. Once it is found, the decoder will proceed as defined by the PLAY control.

- SKIP_TO_NEXT_PICTURE —This control forces the decoder to search forward until the next picture is found. Once it is found, the decoder will proceed as defined by the PLAY control.

It is important to note that without software stream navigation support, the input data rate of the decoder can become quite high (~60MB/s), although during that time there is no motion compensation read/write data. The decoder also allows for picture level control of the forward/backward and decode buffers. The use of modes which skip data looking for start codes requires transport based cataloguing management in order to minimize bandwidth.

3.6.4 Performance

The maximum sustainable rate of the decoder pipe is capable of 1 PEL / clock (1 pixel = 1.5 PEL). So at 167 MHz, the throughput is 111.33 M pixels per second. 1920x1080i material requires 62.208 M pixels per second; however, the following factors affect the decoder pipe's capability to achieve this data rate:

- Starvation due to limited prediction bandwidth (memory controller cannot keep up).
- Starvation due to high bit-rate portion of stream (input fifo empties).
- Syntactical elements in the stream (i.e., picture headers, user data, etc., can slow down processing).

3.6.5 Error Recovery Concealment

Hardware will check for errors in the following ways:

- SEC code (sequence error code) detection.
- Syntactical element out of range.
- Invalid marker bits.

Concealment techniques enabled through hardware include:

- Concealment motion vectors (only possible if included in stream).
- Skipped macroblocks.
- Skipped pictures.

The level of error incurring concealment is programmable, allowing for errors of only a specified level to invoke hardware concealment. The errors are reported in the MPEG_<n>_DECODE_ERROR_INT_STATUS register. This allows for software level concealment at the picture level and above.

3.7 The Audio Subsystem

3.7.1 Introduction

The Xilleon 220 audio subsystem is designed to decode AC3 and MPEG1 layer I and layer II. It supports standard audio interfaces including I2S in/out, SPDIF and AC-Link, as well as a number of audio inputs, i.e., compressed audio from transport streams, PCM audio from devices such as ATI's Theater 200 or a host CPU, and audio from the AC-Link.

The Xilleon 220 on-chip audio decoder decodes two channels of MPEG audio or down-mixes 5.1 AC3 to two channels encoded with Dolby Prologic Surround. Full MPEG2 or AC-3 multi-channel audio needs to be decoded externally using the I2STx port or the SPDIF TX port.

3.7.2 Audio Feature List

AC-3

- Coding standard: ATSC A/52
- Sampling Rates: 48 kHz, 44.1 kHz, 32 kHz
- Coding modes: 1+1, 1/0, 2/0, 3/0, 2/1, 3/1, 2/2, 3/2
- Multi-channel down-mixing
- Re-matrixing to preserve Dolby Prologic encoding
- Conformance: Group A (20 bits)

MPEG

- Coding Standard: ISO11172-3, layers 1,2
- Sampling rates: 48 kHz, 44.1 kHz, 32 kHz
- Conformance: Highest level (16-bit)

PCM

- PCM modes for non-compressed audio data
- Sampling rates: 96 kHz, 48 kHz, 44.1 kHz, 32 kHz, 24 kHz, 22.05 kHz and 16 kHz.
- I/O: 16-24 bit input PCM, 20-bit output PCM
- Modes: PCM by-pass, Mono, Mono-L, Mono-R
- Mixing: Two independent PCM streams are scaled by independent PCM scale factors.

SPDIF

- Frames compressed MPEG or AC-3 data into SPDIF IEC61937 stream by hardware.
- Frames compressed DTS or AAC data into SPDIF IEC61937 stream by software.
- Standards: ATSC/A52 Annex B., IEC 1967

3.7.3 Audio Decoder Block Diagram

This is depicted in *Figure 3-24*. The Xilleon 220 audio decoder contains the following elements:

- An audio decoder core
- An audio register file that contains all the configuration and control registers for the Audio decoder.
- Four audio PLLs to generate clocks for dual output audio streams.
- Single I2S receiver
- Dual I2S transmitters
- Dual S/PDIF transmitters
- Single AC-Link with dual AC’97 CODEC support

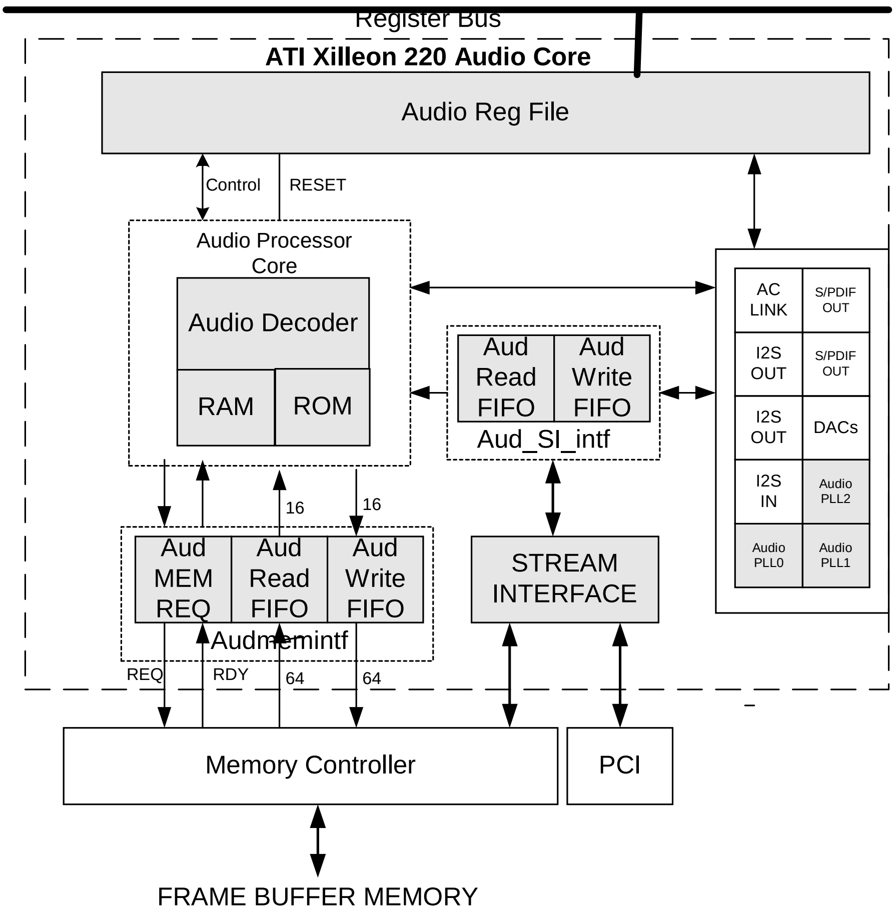


Figure 3-24   Audio Subsystem Block Diagram

3.7.4 Audio Data Flow Inside Xilleon 220

The audio data flow and external interfaces are shown in *Figure 3-25* below. This architecture is able to send out data as explained above. The figure also shows an input data path from the RAGE Theater through I2S.

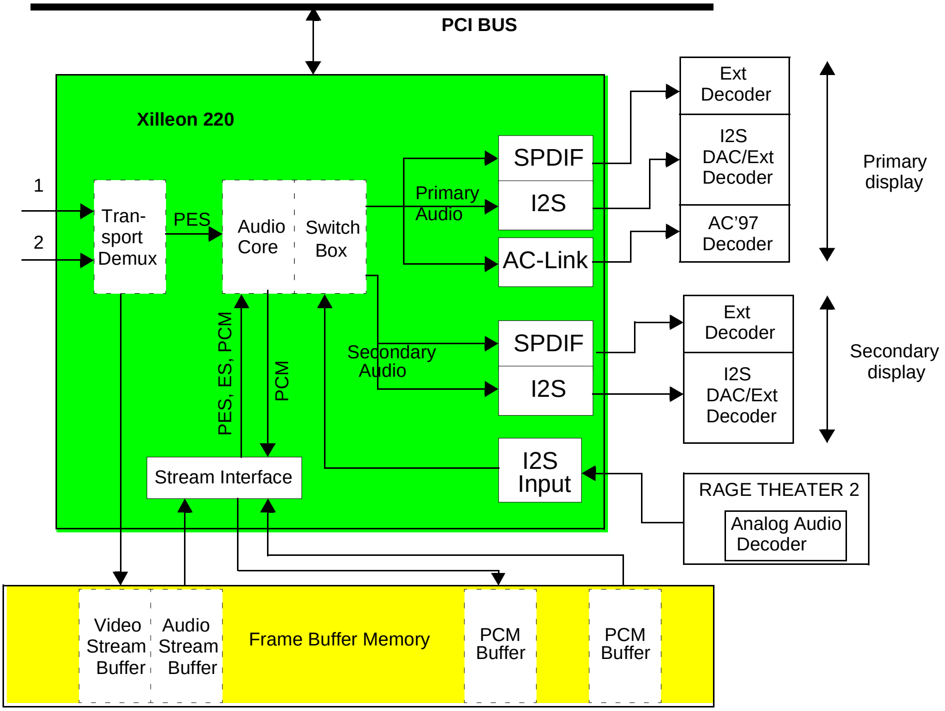


Figure 3-25 Audio Data Flow Diagram

There are two output streams: the primary and the secondary. The primary output stream is defined to be the one that an audience listens to. The secondary output stream is the one that an audience records or sends to a second TV. The following diagrams show all the possible output configurations for the primary and secondary streams

.

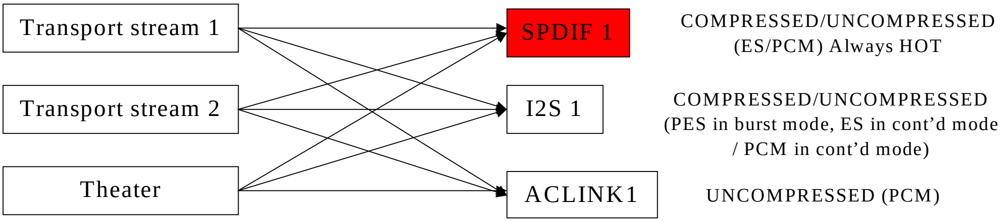

Transport stream 1
Transport stream 2
Theater
SPDIF 1
I2S 1
ACLINK1
COMPRESSED/UNCOMPRESSED (ES/PCM) Always HOT
COMPRESSED/UNCOMPRESSED (PES in burst mode, ES in cont'd mode / PCM in cont'd mode)
UNCOMPRESSED (PCM)

Figure 3-26 Primary Audio Output Format Scenario

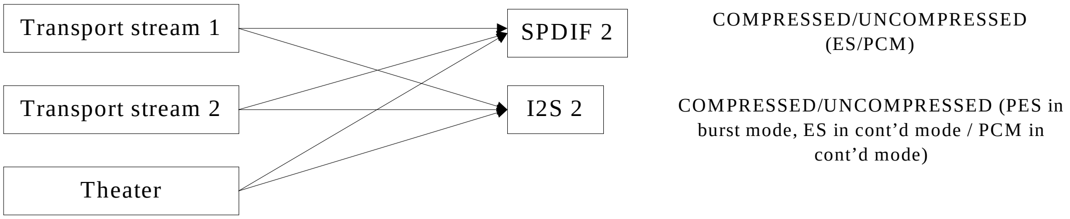

Transport stream 1
Transport stream 2
Theater
SPDIF 2
I2S 2
COMPRESSED/UNCOMPRESSED (ES/PCM)
COMPRESSED/UNCOMPRESSED (PES in burst mode, ES in cont'd mode / PCM in cont'd mode)

Figure 3-27 Secondary Audio Output Format Scenario

Each arrow represents a possible connection. There can only be one audio source for each stream. For instance, primary audio may come from TS1 or TS2 or Theater, but NOT TS1 and TS2 at the same time.

In the primary audio, the SPDIF can be left ON all the time. Either I2S1 or ACLINK1 can be ON. In the secondary audio channel, only one output device is allowed to be enabled.

3.7.5 Audio Mixing

Xilleon 220 contains a mixer to mix the primary PCM output audio with the host audio. This functionality enables the host CPU to alert the user with audio messages. Xilleon 220 does not directly support audio mixing of streams with different sample rates; instead, software must convert the host audio to the sample rate of the main audio before mixing in hardware.

Ramp-up and ramp-down logic provides a smooth volume control, in that the latter is changed only at zero crossings. It also takes care of sudden changes of volume such as muting and audio FIFO underflow.

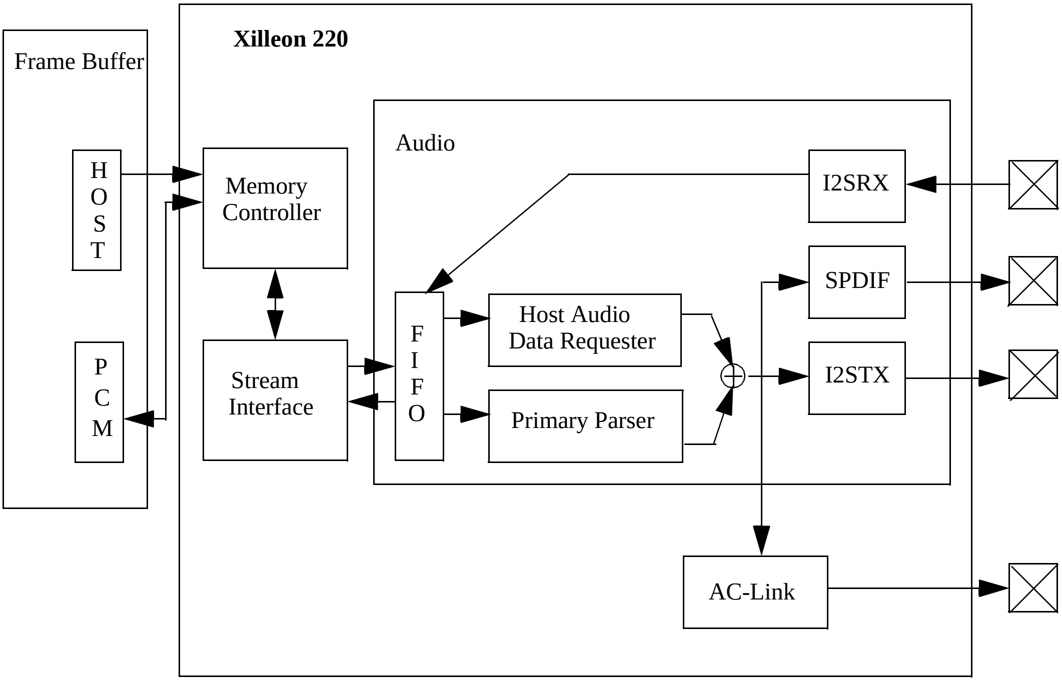


Figure 3-28 Host and Analog Signals in PCM Audio Mixing

Xilleon 220 supports two audio output streams. There is no mechanism to change balance or volume if an output stream is not PCM. If a PCM output stream is primary, balance can be changed by independently adjusting the L/R volume for each stream in separate registers inside the mixer. If a PCM output stream is secondary and this stream does not come from the audio decoder, neither the balance nor the volume can be controlled.

3.7.6 Error Detection Recovery/Concealment

Audio is muted when there is a CRC error in the encoded audio stream. The audio decoder checks the CRC before decoding. It conceals for the first frame that it encounters a CRC error. Concealment means that it repeats the previous frame’s audio samples.

Concealment sounds better than muting for only a very short period of time. So if there is a CRC error due to a drop-out on a single frame, the error is concealed rather than muted. In the case of multiple errors, the first error is concealed, then further errors are muted.

Audio is also muted when the audio decoder does not find valid sync words. Valid sync is considered a sync word at both the beginning and end of a frame.

3.7.7 I2S Interface

Figure 3-25 above shows how the Xilleon 220 outputs a playback and a record stream. Xilleon 220 supports I2S input, AC97 CODEC, and external audio decoder with SPDIF/I2S interface.

3.7.7.1 I2S Out Interface

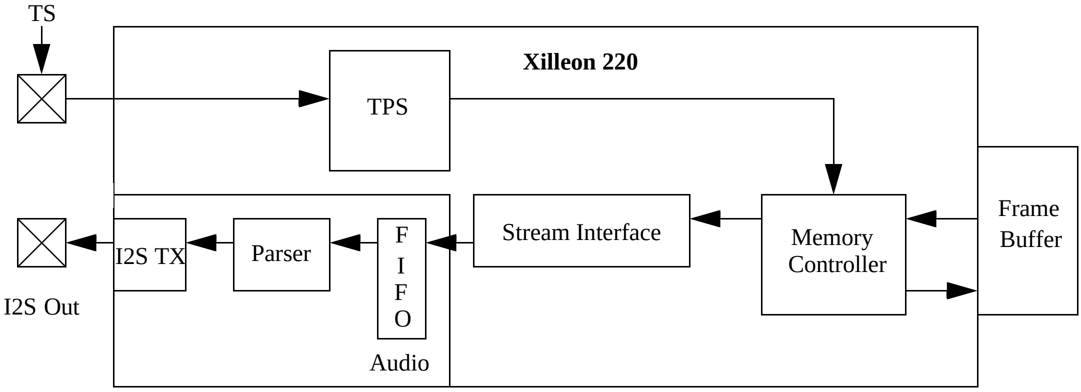


Figure 3-29 Data Flow From Transport to I2S Out

There are two I2S out blocks. These built-in I2S interfaces of the Xilleon 220 are able to drive I2S DACs or external audio decoders. Xilleon 220 supports Crystal CS492x family of decoders.

Table 3-14 I2S Out Interface

Signal Name	Type	Description
SCLK	O	I2S clock. When I2S port work in master mode, it drives clock out. When I2S port is in slave mode, it accepts audio clock from external audio decoder.
SDATA	O	I2S data. Bit serial audio data. MSB first.
RLCLK	O	Word select. In I2S, RLCLK= 0 is for right channel, RLCLK= 1 is for right channel.
DREQb	I	Data request input, when DREQb is 0, the audio decoder is ready to accept data.

The I2S transmitter works in two modes: PCM mode and compressed data mode. The signal DREQb does not apply in the PCM mode. The RLCLK does not apply in the compressed data mode.

In PCM mode, the transmitter outputs data in the format of I2S, right justified or left justified. These formats are well known protocols for transmitting audio PCM samples. The resolution of the data and the width of the RLCLK can be adjusted by programming registers. Supported resolutions are 16-bit, 18-bit and 20-bit, and RLCLK width is fixed at 64 I2S bit clock (SCLK) wide.

In compressed data mode, the DREQb signal comes from an external device. Data is sent when DREQb is asserted low, when DREQb is logic high, no data is sent and the SCLK is stopped. There is no invalid data or extra zeros inserted into the SDATA in this mode. Valid data is present on the SDATA whenever there is a rising edge of the SCLK. In case there is no DREQb, I2STX stops the SCLK and hence data transmission is stopped.

Figure 3-30 and *Table 3-15* below show the I2S Tx timing requirements.

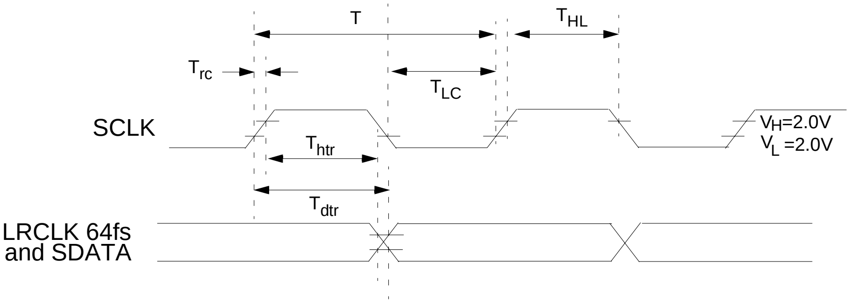


Figure 3-30 I2S Transmitter Interface Timing Diagram

Table 3-15 I2S Tx Timing Requirements

Symbol	Requirement	Description
T	Min allowed clk period for Tx: 0.9T= T_{tr}	Typ. Clock period
T_{HC}	Min > 0.35T	Clock high
T_{LC}	Min > 0.35T	Clock low
T_{dtr}	Max < 0.8T	Delay
T_{htr}	Min > 0	Hold time
T_{rc}	Max > 0.15T_{tr} (only relevant in slave mode)	Clock rise time

3.7.7.2 I2S Input Interface

There is a single I2S input interface which is intended to support analog audio from Theater^{TM} 200 or other I2S sources. This port is able to operate in slave mode only. *Table 3-16* below shows the pins for this interface.

Table 3-16 I2S Input Interface

Signal Name	Type	Description
SCLK	I	I2S clock. When I2S port work in master mode, it drives clock out. When I2S port is in slave mode, it accepts audio clock from external audio decoder.
SDATA	I	I2S data. Bit serial audio data. MSB first.
RLCLK	I	Word select. In I2S, RLCLK= 0 is for right channel, RLCLK= 1 is for right channel.

Figure 3-31 shows the I2S format. For I2S, data is presented most significant bit first, one SCLK delay after the transition of the LRCLK and is valid on the rising edge of SCLK. From the I2S format, the left sub-frame is presented when LRCLK is low and the right subframe is presented when LRCLK is high. SCLK is required to run at a frequency of 64 frames on the input ports.

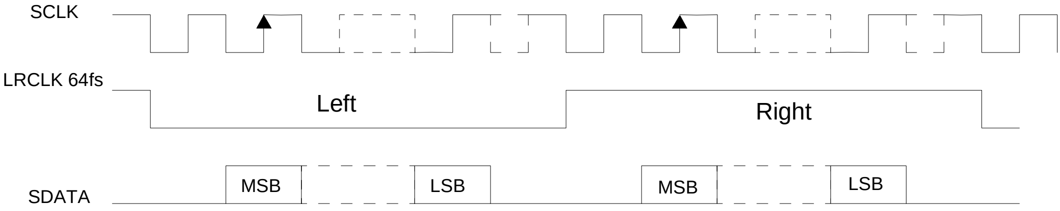


Figure 3-31 I2S Format

Figure 3-32 and *Figure 3-33* show the left and right justified format with a rising edge SCLK. Data is presented most significant bit first on the first SCLK after an LRCLK transition and is valid on the rising edge of SCLK. For the left justified format, the left sub-frame is presented when LRCLK is high and the right subframe is presented when LRCLK is low. The left justified format can also be programmed for data to be valid on the falling edge of SCLK. SCLK is required to run at a frequency of 64Fs on the input ports.

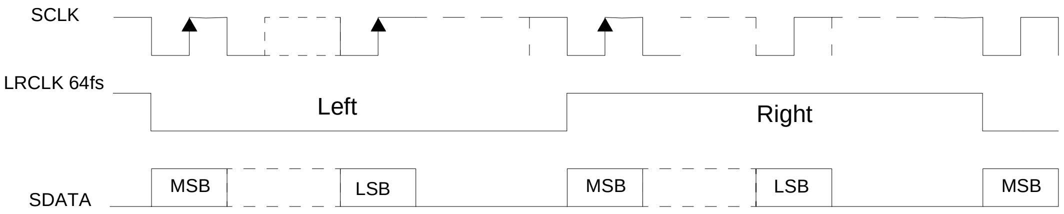


Figure 3-32 Left Justified Format (Rising Edge Valid SCLK)

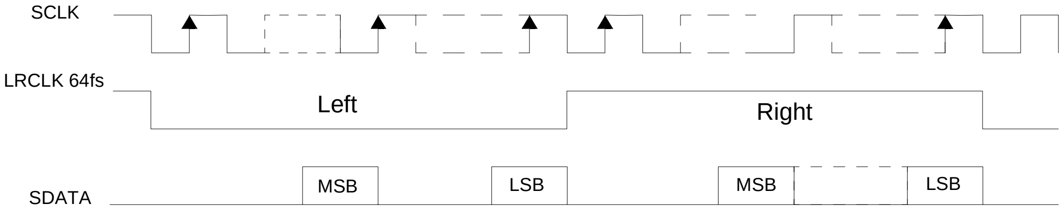


Figure 3-33 Right Justified Format (Rising Edge Valid SCLK)

Figure 3-34 and *Table 3-17* below show the I2S Rx timing requirements.

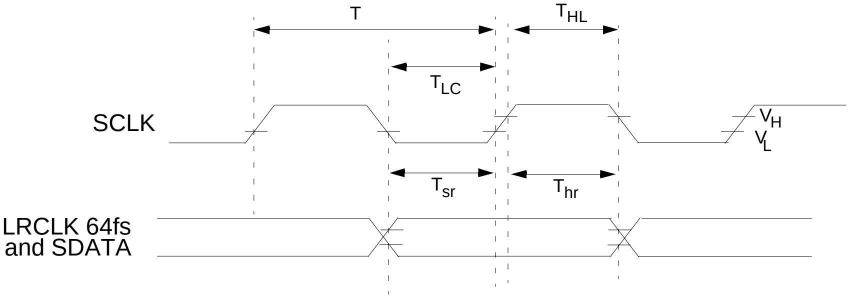


Figure 3-34 Timing Diagram of the I2S Receiver Interface

Table 3-17 I2S Rx Timing Requirements

Symbol	Requirement	Description
T		Typ. Clock period
T_{HC}	Min > 0.35T	Clock high
T_{LC}	Min > 0.35T	Clock low
T_{sr}	Max < 0.2T	Setup time
T_{hr}	Min < 0	Hold time

3.7.8 AC-Link Interface

See *"AC-Link Interface" on page 6-21* for details.

3.8 Display Subsystem

3.8.1 Introduction

Xilleon 220 supports a primary and a secondary display (e.g., a VCR or a second TV). The primary display is able to drive HDTV (e.g. 720p and 1080i) or SDTV (e.g. 480p, NTSC, PAL, and SECAM) standards while the secondary output drives SDTV on a composite/s-video signal. Picture-in-picture (PIP) is also supported on the primary display in the absence of a second display.

3.8.2 Features

Primary Display

- HDTV Encoder (supports HDTV TVs with tri-level syncs and correct signal shaping) and standard RGB monitors (VESA modes) up to 150MHz pixel clock rates).
- SDTV Encoder (drives all NTSC, PAL, or SECAM TV sets. It also supports Component 480i and 480p with Macrovision).
- Quad 10-bit DAC (useful for SCART-RGB or composite RF-mod + composite baseband + S-Video baseband) plus HSYNC and VSYNC.
- DAC sample rate is 148.5 MHz for all HDTV (e.g. 1080i) and SDTV modes. This allows a simple external filter on the board to suffice for all HDTV and SDTV modes. Signals are shaped with programmable digital filters to meet HDTV or SDTV specifications.
- The HDTV encoder, SDTV encoder, or VESA encoder can drive the Quad DAC.
- Termination load detection HW allows software to determine what connectors the user has hooked up.
- Internal DVI HDCP encryption hardware.
- Drives an external TMDS encoder/driver chip (e.g. SiI 168).
- CCIR656 style digital output port.
- Display refresh rate locks with the rate of the dominant video source.
- When the display is interlaced and the video’s vertical scaling is 1:1, the display field polarity automatically synchronizes with the field polarity of the dominant video source.

Secondary Display

- Standard VESA compatible RGB monitor support (up to 1280x1024).
- Target device is a VCR or a second TV.
- STDV Encoder (drives all NTSC, PAL, or SECAM TV sets).
- Triple 10 bit DAC plus HSYNC, and VSYNC.
- DAC sample rate is 148.5 MHz for all modes. This allows a simple external filter on the board to suffice for all HDTV (e.g. 480p) and SDTV modes. Signals are shaped with programmable digital filters to meet HDTV or SDTV specifications.

- Component Video (480p) output.
- Display refresh rate locks with the rate of the dominant video source.
- When the display is interlaced and the video's vertical scaling is 1:1, the display field polarity automatically synchronizes with the field polarity of the dominant video source.

3.8.3 Block Diagram

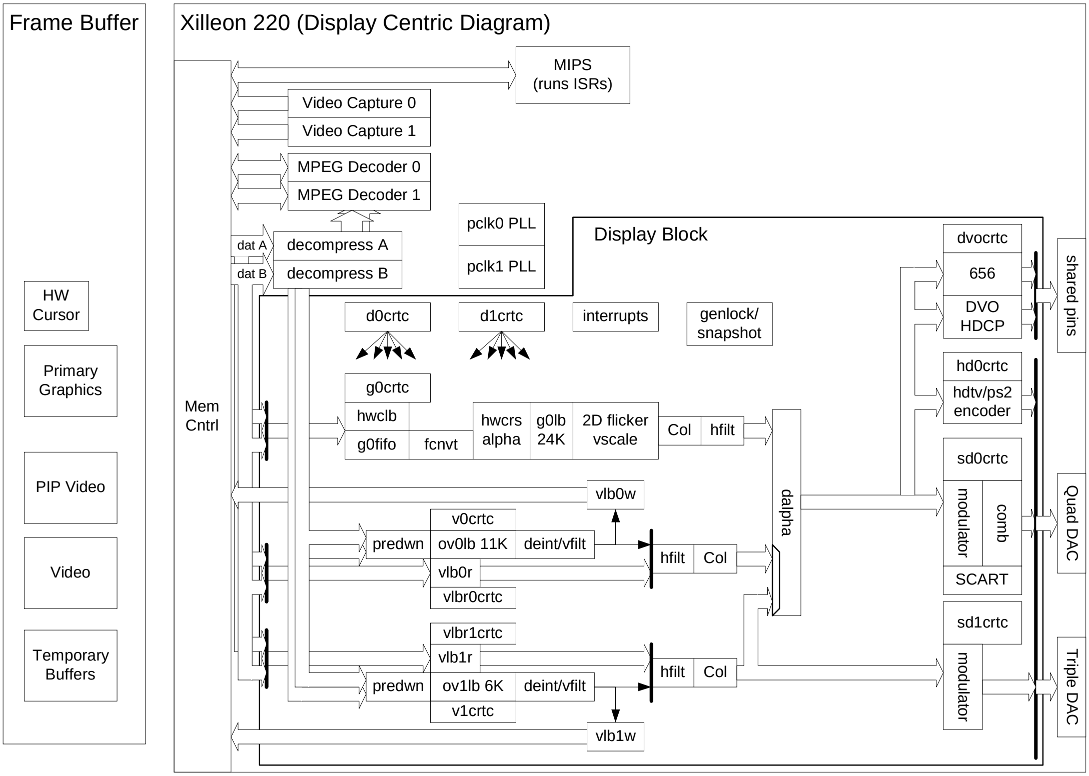


Figure 3-35 Display Top Level Block Diagram

3.8.4 Data Flow

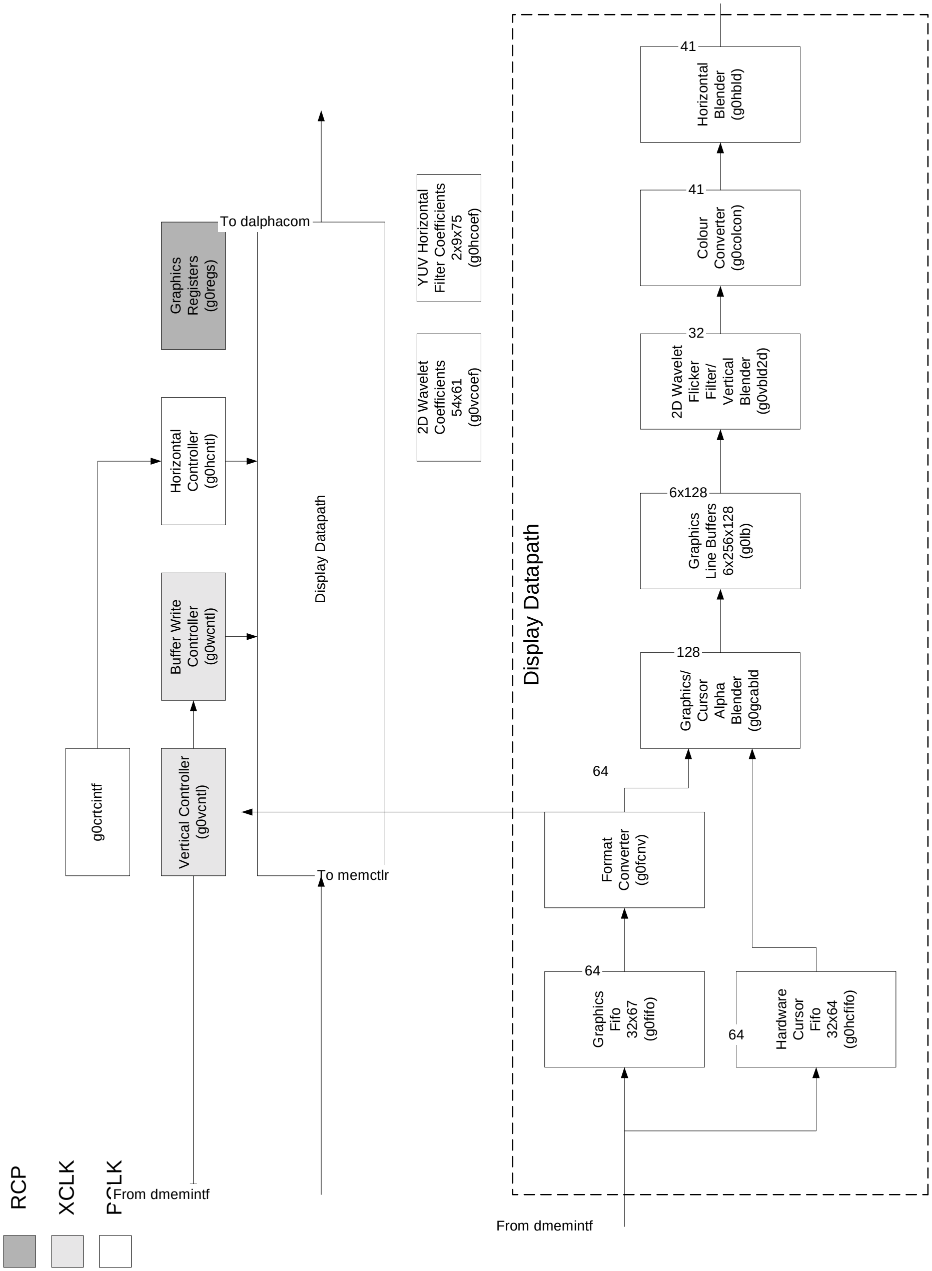

Horizontal Blender (g0hbld)
41
Colour Converter (g0colcon)
41
2D Wavelet Flicker Filter/ Vertical Blender (g0vbld2d)
32
Graphics Line Buffers 6x256x128 (g0lb)
6x128
Graphics/ Cursor Alpha Blender (g0gcabld)
128
64
Format Converter (g0fcnv)
Graphics Fifo 32x67 (g0fifo)
64
Hardware Cursor Fifo 32x64 (g0hcfifo)
64
From dmemintf
Display Datapath
YUV Horizontal Filter Coefficients 2x9x75 (g0hcoef)
2D Wavelet Coefficients 54x61 (g0vcoef)
To dalphacom
Display Datapath
To memctlr
Graphics Registers (g0regs)
Horizontal Controller (g0hcntl)
Buffer Write Controller (g0wcntl)
Vertical Controller (g0vcntl)
g0crtcintf
From dmemintf
RCP
XCLK
PCLK

Figure 3-36 Display Data Path

3.8.5 TV Encoders

3.8.5.1 Features

There are two TV encoders in the Xilleon 220 display block — The Standard Definition TV Encoder (sdtvo0) and the second encoder (sdtvo1). The Standard Definition TV Encoder (see *Figure 3-37*) produces a conventional NTSC, PAL or SECAM signal in Composite, S-Video, or RGB formats. It takes the proper vertically and horizontally rendered YUV data, encodes it, as well as the proper timing information, and puts it into the required standard.

The second encoder (sdtvo1) does not support SCART-RGB because it has a triple DAC and it does not support the 2 line comb-on-the-way-out feature.

The description below applies to the full featured encoder.

- NTSC, PAL, and SECAM encoding (all variants supported).
- SECAM signal shaping (such as pre-emphasis filter).
- Supports SCART-RGB (on the main display).
- Macrovision 7.1 (exact sub-revision to be confirmed).
- 2 line comb-on-the-way-out for luma-chroma cross-talk prevention on the main display.
- Resampling and pass through of raw VBI data received from an external device (see *“Vertical Blanking Interval Service Encoder (VBI Block)” on page 3-84*.

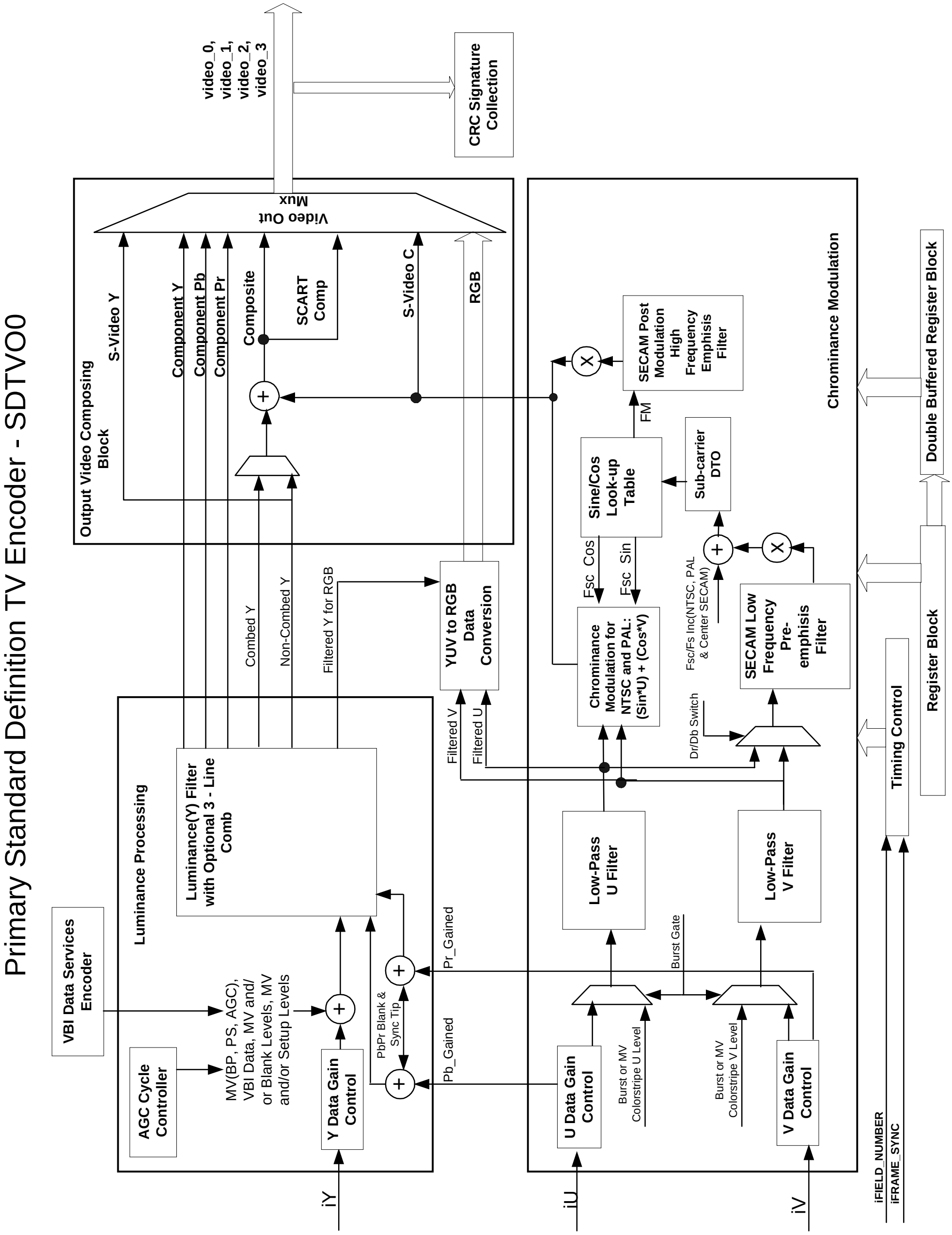

Primary Standard Definition TV Encoder - SDTVO0
video_0, video_1, video_2, video_3
CRC Signature Collection
Output Video Composing Block
Video Out Mux
S-Video Y
Component Y
Component Pb
Component Pr
Composite
SCART Comp
S-Video C
RGB
Chrominance Modulation
SECAM Post Modulation High Frequency Emphisis Filter
FM
Sine/Cos Look-up Table
Sub-carrier DTO
Fsc Cos
Fsc Sin
Chrominance Modulation for NTSC and PAL: (Sin*U) + (Cos*V)
Fsc/Fs Inc(NTSC, PAL & Center SECAM)
SECAM Low Frequency Pre-emphisis Filter
Dr/Db Switch
Double Buffered Register Block
Register Block
Timing Control
YUV to RGB Data Conversion
Filtered Y for RGB
Combed Y
Non-Combed Y
Filtered V
Filtered U
Luminance Processing
Luminance(Y) Filter with Optional 3 - Line Comb
VBI Data Services Encoder
AGC Cycle Controller
MV(BP, PS, AGC), VBI Data, MV and/ or Blank Levels, MV and/or Setup Levels
Y Data Gain Control
PbPr Blank & Sync Tip
Pr_Gained
Pb_Gained
Low-Pass U Filter
Low-Pass V Filter
Burst Gate
U Data Gain Control
Burst or MV Colorstripe U Level
Burst or MV Colorstripe V Level
V Data Gain Control
iY
iU
iV
iFIELD_NUMBER
iFRAME_SYNC

Figure 3-37 Sdtvo0 Block Diagram

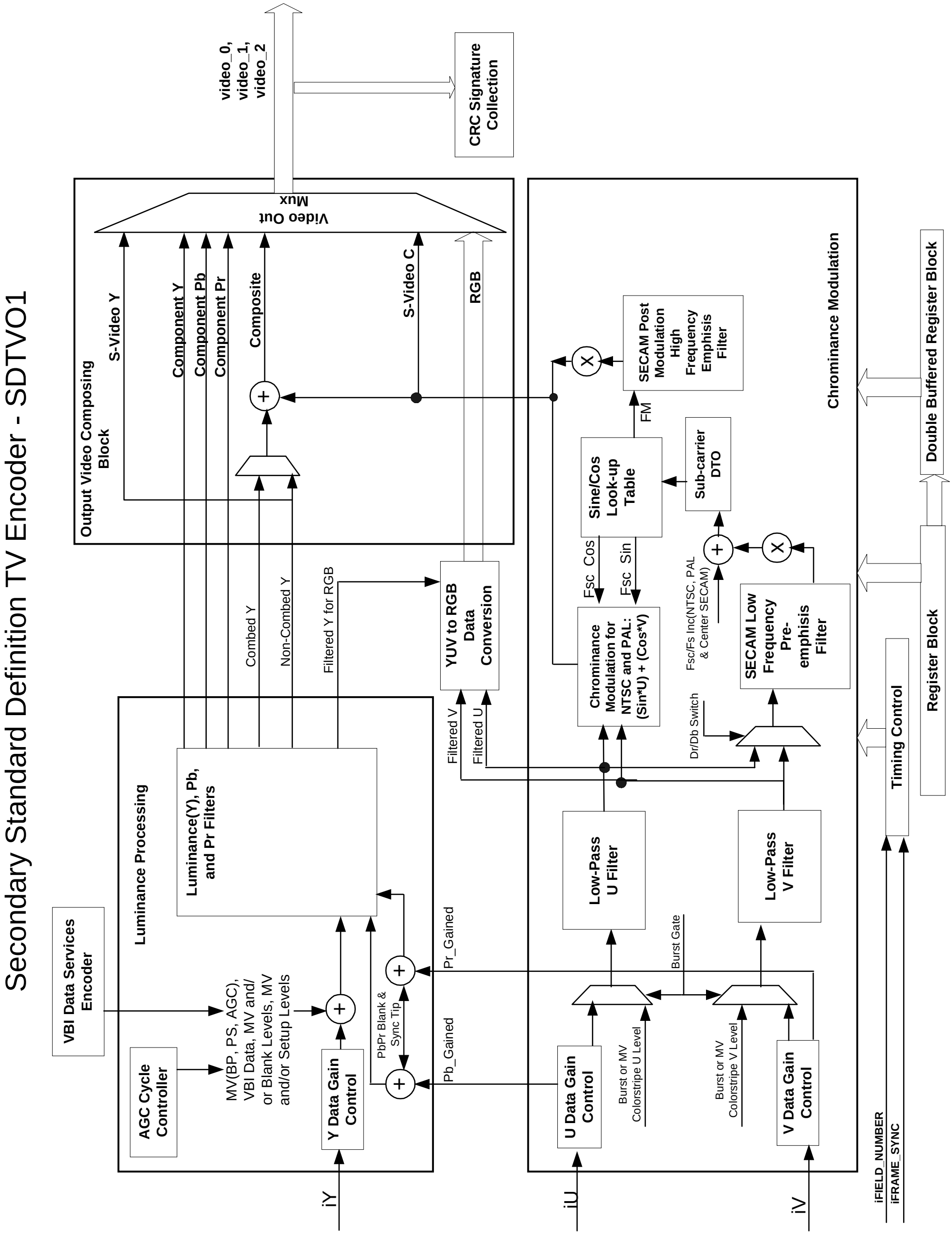

Secondary Standard Definition TV Encoder - SDTVO1
video_0,
video_1,
video_2
CRC Signature Collection
Output Video Composing Block
Video Out Mux
S-Video Y
Component Y
Component Pb
Component Pr
Composite
S-Video C
RGB
Combed Y
Non-Combed Y
Filtered Y for RGB
YUV to RGB Data Conversion
Filtered V
Filtered U
Luminance Processing
Luminance(Y), Pb, and Pr Filters
VBI Data Services Encoder
AGC Cycle Controller
MV(BP, PS, AGC), VBI Data, MV and/ or Blank Levels, MV and/or Setup Levels
Y Data Gain Control
PbPr Blank & Sync Tip
Pr_Gained
Pb_Gained
iY
iU
iV
Chrominance Modulation
SECAM Post Modulation High Frequency Emphisis Filter
FM
Sine/Cos Look-up Table
Sub-carrier DTO
Fsc Cos
Fsc Sin
Fsc/Fs Inc(NTSC, PAL & Center SECAM)
Chrominance Modulation for NTSC and PAL: (Sin*U) + (Cos*V)
SECAM Low Frequency Pre-emphisis Filter
Dr/Db Switch
Low-Pass U Filter
Low-Pass V Filter
Burst Gate
Burst or MV Colorstripe U Level
Burst or MV Colorstripe V Level
U Data Gain Control
V Data Gain Control
Double Buffered Register Block
Register Block
Timing Control
iFIELD_NUMBER
iFRAME_SYNC

Figure 3-38 Sdtvo1 Block Diagram

3.8.5.2 TV Encoder Components

Timing Control (TIMING Block)
This is a register directed state machine. It provides all of the control signals to generate a standard definition television signal. It provides the qualifier to designate the time of the following: horizontal sync, equalization and serration sync (vertical sync), color burst, limits of active data, actual active data, VBI pass through data, active signals for VBI encoder, active area for slew rate filtering, and all the Macrovision controls.

Vertical Blanking Interval Service Encoder (VBI Block)
Xilleon 220 is capable of supporting the following VBI data insertion standards:

- Line 21 for NTSC data on odd and even fields separately, including Closed Caption, Parental Rating information, ATVEF transport A, and other Extended Data Services (EDS).
- Wide Screen Signaling (WSS) for both PAL/SECAM and NTSC, including the Analog Copy Generation Management System (CGMS-A).
- Teletext data for PAL/SECAM.
- NABTS data for NTSC.

The flexible nature of VBI data insertion implemented in Xilleon 220 allows adopting new and/or proprietary VBI data standards as they emerge.

Luma Processor (LUMA_PROC Block)
This block processes all of the luminance active and synchronization portions of the signal. It provides gain to the active luma Y, muxes in the appropriate sync tip, blank level (Macrovision or normal), setup level (Macrovision or normal), Macrovision Back Porch Pulses, Macrovision Automatic Gain Control (AGC) pulses, Macrovision Psuedo Sync pulses, or VBI data. It also provides filtering for Digital to Analog conversion.

Luma Gain Control (YGAINCNTL Block)
This block provides programmable luminance (Y) gain from 0 to 1.25. It also applies a programmable break point level above which the luma gain is 2:1.

Macrovision AGC Cycle Controller (AGCCYCLE Block)
This is a state machine which produces the varying amplitude Macrovision AGC pulses for the various standards.

Luma Filter (YUPFILT Block)
This block filters the luminance portion of the signal in preparation for conversion to analog. The VBI material passes through the filter untouched. Everything else, except active data, must go through a fairly low pass slew filter to limit the hard transition ringing. Through the selection of a couple of low pass filters and a band pass filter, the active data can be filtered for a few degrees of sharpness and combed to remove false colour.

Luma Delay Line (LUMADELAYLINE Block)
This block contains memory to store two complete lines of pixels. This will enable the luma filter to have access to three lines of data to perform up to a three line comb filter function.

There is also logic to allow two pixel reads and two pixel writes during every period from a single port memory. The memory will also be used to provide a programmable pixel delay to allow for chroma/luma pipeline balancing.

Chroma Modulator (CHROMA_MOD Block)

The Chroma Modulator block creates the chroma or color portion of the signal including the reference color burst. It takes the active U and V data streams, it applies programmable gain, and low pass filters and modulates this into the required chroma portion for that standard. For SECAM the necessary pre-emphasis filters and FM modulation are provided to produce the chroma. Macrovision Color Stripe is also supported for the color burst.

Chroma Gain Control (UGAINCNTL & VGAINCNTL Blocks)

The Chroma Gain Control provides programmable chroma gain from 0 to 1.25 on the U and V data inputs. It also applies a programmable break point level above which the chroma gain is 2:1.

Low Pass Chroma Filters (UUPFILT & VUPFILT Blocks)

The Low Pass Chroma Filters provide a first order IIR 1.3 MHz low pass filter to the U and V data paths or burst gate. This is done in preparation for modulation in the case of NTSC and PAL, and as the first stage of the low frequency pre-emphasis for SECAM.

Discrete Time Oscillator (DTO_SCCLK Block)

This block contains two discrete time oscillators (DTO) which provide the phase to create the sub-carrier sinusoidal for NTSC and PAL, and the 2 center frequencies for SECAM. The sample frequency is TV_CLK operating roughly at 12 times the sub carrier frequency.

This block allows for a more accurate generation of different clock frequency, and eliminates the number of PLLs needed. Also, by performing the encoding and control signals insertion at one clock (TV virtual clock) and then upsampled to the Fs ensure a more accurate timing relationship between the control signals and the TV signals.

Below shows a simple block diagram for this design.

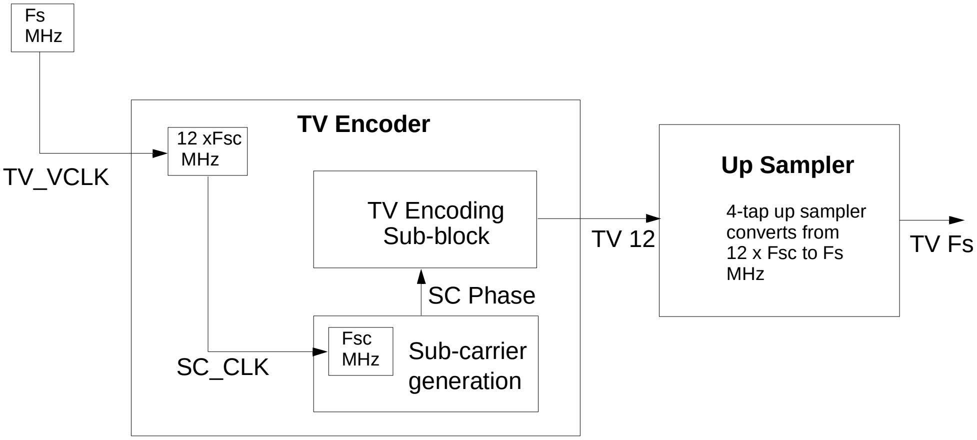


Figure 3-39 Discrete Time Oscillator Implementation

Sine and Cosine Look up Table (SINCOSLUT Block)

This block consists of a look up table with a 12 bit phase input producing a 13 bit sine and a 13 bit cosine value at that phase.

SECAM Low Frequency Pre-emphasis (SCM_LFPRE Block)

This IIR filter, cascaded with Uupfilt or Vupfilt, provides the necessary frequency response emphasis on the chroma data required before the SECAM FM modulation.

SECAM High Frequency Pre-emphasis (SCM_HFPRE Block)

This IIR filter shapes the post FM modulated chroma sinusoidal frequency response. It attenuates those frequencies above and below the two center frequencies, as required by the SECAM standard.

Chroma Delay Line (CHROMADELAYLINE Block)

This block contains memory to store one complete line of pixels to match the one line of delay that the luma comb filter will impose if enabled. The memory will also be used to provide a programmable pixel delay to allow for chroma/luma pipeline balancing.

Colour Converter YUV to RGB (YUVTORGB Block)

This block converts the properly filtered YUV data into a RGB format. This same YUV (except for Y which is allowed to have different macrovision settings for RGB) will also be converted into a proper composite format to act as sync for the RGB display.

Compose Output Video Format (VID_COMPOSE Block)

This block takes in luma, Y, for S-video, Y for Composite, chroma, C, for S-video, chroma for composite, red, green, or blue for RGB. It either adds or muxes these signals in any combination for up to four parallel outputs with Composite video, S-video Y, S-video C, Red, Green, Blue, or Composite for RGB sync.

Cyclic Redundancy Checking Signature Analyzer (TVCRC Block)

This block provides CRC signature analysis on the four data outputs from the video composer. It provides this analysis in the TV clock domain rather than in the PCLK domain provided by the Display block CRC.

Output Upsampler (PCLK_UPSAMP Block)

This block takes the four data lanes clocked at the slower TV clock rate and upsamples them to the higher 148.5 MHz PCLK rate. The entire encoder operates on the TV clock which is a DTO generated clock with variable period lengths that average to 1/12 of the Sub-carrier period of the required standard. The TV clock DTO is clocked with PCLK so this upsampler brings the data back into the proper clock domain for digital to analog conversion.

Resync Register Fields (DOUBLEB_REGS Block)

This block provides a buffering mechanism for those register fields (ex. Macrovision control) that are to be changed on the fly. However, the circuit that the register fields control will not tolerate asynchronous changes. There is a parallel bank of flops that gets updated from the register block (Sdregs) at controlled times. A lock bit will be set by software when the buffered fields are to be written. Only when this bit is low, and software has completed the re-write, will the Doubleb_regs version be updated. The circuit is controlled by the Doubleb_regs version. Because of this lock bit, it does not see the partially written bytes in the field and ignores the fact that the Sdregs block is on an asynchronous clock.

A second mechanism allows the update of Doubleb_regs only at a programmable time of end of frame, end of field, end of line or immediately. This prevents possible problems caused by changing the controls of the timing state machine in the middle of a state.

Register Block (SD0REGS)

This block provides the register field addressing, decoding, writing and reading.

3.8.6 Theory of Operation

3.8.6.1 Graphics Scaler

Compositing of the Graphics Image

The 2D/3D engine composites numerous sprites of numerous formats into an image for display. This composited image will generally be ARGB32(8:8:8:8), but the formats ARGB32(4:8:8:8), ARGB16(4:4:4:4), and ARGB16(1:5:5:5) are also supported for low cost systems or for applications such as gaming.

The source material for the graphics image (sprites, fonts, bitmaps, anti-aliased text, textures, etc.) can be provided in numerous formats including most varieties of RGB, CLUT, texture formats, and even pre-scaled YUV video. Colour keying, global alpha, or per pixel alpha can be used to control blending ratios.

Advantages of Front-end Compositing over Back-end Compositing

For graphics objects, front-end compositing has significant advantages over back-end compositing. For example, safe back-end compositing requires a complicated analysis of the memory bandwidth. If too many sprites are placed on a line, the back-end compositor will cause a spike in the use of bandwidth. This may cause artifacts on the display, or cause a catastrophic FIFO underflow or overflow condition on another memory client in the system. To avoid these back-end compositing problems, one must constrain how the developer can create the GUI and how the user can manipulate it. This is especially true if the product extracts the maximum value out of the memory sub-system. With front-end compositing, on the other hand (as is used in Xilleon 220), use of memory bandwidth is better amortized over time. Bandwidth usage is actually leveled overall because the less time critical front-end compositing operations use bandwidth left-over from the more time critical memory clients in the system.

Front-end compositing can be synchronized not only with the display refresh, but also with a multiple of the display refresh. The developer can thus create smooth animation effects at the most desirable update rate. At times when there is no animation occurring, memory bandwidth (and indirectly power) can be saved by only updating the GUI as often as events require it.

Front-end compositing is flexible and allows a mix of 2D scaling and 3D warping effects to be used to generate the best possible image. Back-end compositing tends to be two dimensional with no scaling. To achieve the same effects extra temporary surfaces would be needed.

With front-end compositing, images can be created directly with standard API calls. Back-end compositing, on the other hand, generally requires the developer to work through some kind of proprietary display list to achieve whatever the compositing effects are supported by the display and memory sub-system of the product.

The bottom line is that front-end graphics compositing simplifies middleware development. It is hard to keep on top of problems associated with back end compositing when incorporating third party SW, managing SW upgrades, and distributing SW across more than one generation of the HW. Successful and long lasting architectures, such as those upon which PCs and workstations are based, use front-end rather than back-end compositing for creating compelling GUI interfaces that are both flexible and scalable.

Alpha Blended Hardware Cursor

A 64x64 or 32x32 size, ARGB32(8:8:8:8) hardware cursor sprite is composited in the back end for two main reasons. First, it greatly simplifies the SW programming model for the cursor by separating it from the complexity of any other compositing operations; second, it ensures that the true colour alpha blended sprite is always available - even in very low cost systems with low pixel depth graphics.

The high quality cursor is important. Animation effects, such as the position of the sprite relative to its shadow, can help make up for missing tactile feedback and less precise control if the user must control the sprite with a remote rather than a mouse.

Scaling of the Graphics Image

The graphics scaler supports up to 1024x768 resolutions with 32bpp true colour ARGB. It will scale the graphics source to fit any display size. This means that the graphics content (sprites, fonts, menus, EPGs, etc.) for the product can be authored at *one resolution for all displays*. The hardware performs scaling and, if needed, flicker filtering to convert the common source resolution to the resolution of the user's display. Displays ranging from composite NTSC TV sets to 1280x720p DVI connected flat panel TVs can be supported with one consistent library of graphics content.

Square pixels can be maintained, or the graphics aspect ratio can be distorted slightly to better make use of available screen real estate - at the programmer's discretion. The graphics can be optionally under-scanned while the video is over-scanned and vice-a-versa.

To save on memory and memory bandwidth, the graphics can be programmed to cover only a small portion of the screen (for the purposes of displaying a logo, for example).

All graphics scaling is done with 10 taps horizontally and 6 taps vertically. The filter kernel is optimized for the scaling ratio and the type of display device. The one exception described below is that even more taps are used when performing scaling and flicker filtering for SDTV sets.

Flicker Filtering the Graphics Image

The graphics scaler flicker filters graphics for interlaced displays, *especially* for the SDTV TV sets that most people still use. For SDTV sets, a 15x6 tap wavelet based filter kernel removes flicker without unnecessarily blurring graphical details in the image that do not generate objectionable flicker. The resulting flicker and image sharpness is as good as expensive rack-mount studio equipment. Furthermore, the user can tune flicker suppression to optimize viewing distance and to account for personal taste.

In other solutions, flicker filtering is performed after other display operations such as scaling. The detail in text is unnecessarily blurred first by the scaling operation, and then again by the flicker filtering operation. Despite the fact that a flicker filtering has its own section in this document, in the Xilleon 220, flicker filtering and scaling are combined into one operation to prevent the accumulation of errors and achieve the best possible quality - especially with respect to preserving the detail in text.

The scaling plus flicker filtering is done independently for graphics and each video window. Because no compromises have to be struck, the signal processing can be optimized for each window's scaling ratio and type of content.

The upshot is that the high quality of the graphics scaler permits more information to be displayed on the TV. This, in turn, leads to better EPGs and a better web browsing experience. It then becomes easier to engage a consumer in new interactive services, such as on-line shopping and banking.

(Note: discussed later is a "comb-on-the-way-out" feature of the SDTV encoder that also enhances graphics image quality on SDTV sets by suppressing false color.)

Colour Converting the Graphics Image

Colour conversion uses a full 3x4 matrix multiply with 3.8 fixed point coefficient precision, and 11.1 precision for the constants. All colour conversions and non-unity gain adjustments are done only in the colour converters where full precision can be maintained throughout the calculation. Thus, there is no accumulation of rounding errors in the system.

The flexibility of the matrix multiply allows adjustment of brightness, contrast, saturation and hue. It also allows the colour components to be switched around so that surface formats such as ABGR can be supported (for RiscOS) instead of the typical ARGB.

Alpha Blending of Graphics with Video

Full 8 bit per pixel alpha blending is done between the graphics and the video. Alpha values are carried through the scaling process and blending is done after both video and graphics have been scaled and colour converted. Global alpha is also supported.

The accuracy of the alpha blending algorithm is equivalent to a SW algorithm that uses floating point precision to maintain its accuracy. Cheaper implementations that cannot make this claim lose one bit of precision. Thus, they actually have only seven effective bits of alpha support even though eight bits may be provided in the source material.

3.8.6.2 Video Scaler

There are two video scalers — V0 and V1. V0 displays video on the primary display device. V1 can display the primary display for picture-in-picture (PIP) or drive the secondary display.

One scaler can be reused in different vertical regions of the screen to create more windows. Settings, including scale ratios, surface format, deinterlacing, and colour conversion, can be reprogrammed for each window. Register double buffering allows programming for the next window to be done while the video scaler is still busy with the current window.

When decoding and displaying either analog or digital video, the video scaler is under firmware control. The middleware developer programs the firmware on which decoded stream to display, and where on the screen to display it. The firmware keeps track of the possibly changing resolution of the video, and programs the video scaler dynamically.

When downscaling, the firmware will automatically use multiple passes to more effectively amortize memory bandwidth and to achieve the highest possible image quality. The total number of taps used for the overall scaling operation depends on the scaler ratio and the number of passes. The minimum number of taps used is 10 taps horizontal by 4 taps vertical. The maximum depends on the number of passes and can be in the hundreds.

Both scalers have a PIP display FIFO. This FIFO can free the video scaler from display responsibilities, thereby increasing the number of quality enhancement operations the video scaler can perform per displayed image.

The video scalers can synchronize with the MPEG decoders, pick up the decoded video, downscale it to a manageable size (with multiple passes if necessary) and place the pre-downscaled image back into the frame buffer in a format that the 2D/3D engine can use as a texture. This allows the 2D/3D engine to create sweep effects with video.

Deinterlacing

Both video scalers support Per-Pixel Adaptive Spatial-Temporal Deinterlacing. The firmware will dynamically control deinterlacing options to achieve the best possible quality when the content arrives in an interlaced format.

Colour Converting the Video Images

Each scaler has its own colour converter. Each performs a full 3x4 matrix multiply with 3.8 fixed point coefficient precision, and 11.1 precision for the constants. All colour conversions and non-unity gain adjustments are done only in the colour converters where full precision can be maintained throughout the calculation. Thus, there is no accumulation of rounding errors in the system.

Direct Manual Control Of Video Scalers

It is possible for software to have direct manual control of the video scalers. The details of the video scaler's capabilities are interesting under these circumstances.

Surface Formats

Both scalers directly support all of the RGB and YUV formats described in *Table 3-18 on page 3-92*. The YcbYCr and CbYCrY formats are formats used by the video capture ports. The two plane YUV12 formats are used by the MPEG decoders. The MPEG decoders also use special tiling and interleaving options to achieve very high memory efficiency on random access motion compensation operations.

Many other surface formats are supported with help from the 2D/3D engine. For example, an off-the-shelf software MPEG decoder may decode to a YUV12 or YUV9 surface format. These surfaces have three planes, and the U and V planes are sub-sampled both vertically and horizontally (by two for YUV12 and by four for YUV9). To display the decoded image, the software decoder calls the driver's “Flip” API (the actual name of the API depends on the OS). This API automatically and efficiently converts the three-plane formats YUV12 and YUV9 to the newer and more bandwidth efficient two-plane YUV12 formats.

Note that the video scalers ignore the alpha component of the ARGB formats.

Size Restrictions

The only difference between the two video scalers is the size of the line buffers. V0 can store 11520 bytes while V1 can store 7808 bytes. This means that the maximum source zoom window widths are different for each of the two video scalers.

The maximum source zoom window height is always 8192 lines.

The video scalers can trade off scaling quality versus maximum size in a flexible manner. Unfortunately this makes the rules defining maximum width more complicated.

For example, the V1 line buffer (which stores 7808 bytes) can store two lines of Y and two lines of UV (2*1920 + 2*1920 = 7680) and thus perform two tap vertical scaling on a source of this width. If the lines are pre-downscaled by a factor of two, the line buffer can store four lines of Y and four lines of UV and perform four tap vertical scaling.

The video source surface and the surface's pitch can be wider than maximum, as long as the difference between the zoom window's left and right coordinates (rounded down and up to multiples of 64) are within the legal maximum width.

Table 3-18 Formats Directly Supported by the Video Scalars

Format Number	Format Name	FourCC name	Max Width for V0	Max Width for V1
3	ARGB16(1:5:5:5)	N/A	1920	1920
4	ARGB16(5:6:5)	N/A	1920	1920
6	ARGB32(8:8:8:8)	N/A	1440	976
11	CrYCbY (Cr is in byte 3)	YUY2	1920	1920
12	YCrYCb (Cb is in byte 0)	UYVY	1920	1920
13	Two plane YUV12 (in the UV plane, V is in byte 0)	NV12	1920	1920

If your application is unusual, you should contact ATI to discuss the possibilities. The video scalers are very flexible and may be able to go beyond the limits described above in certain cases.

Scaling Restrictions

There is no practical limit on how much the video scalers can upscale in a single pass. There are limits on how much they can downscale in a single pass. There are physical downscaling limits that prevent single pass downscaling scaling below ratios of approximately 32:1 - the exact number depends on the surface format. To achieve these extreme ratios in a single pass the video scalers will drop lines and/or pixels. An explanation of how the video scalers can be made to perform multiple passes when under direct manual control is outside the scope of this document.

3.8.7 External Interfaces

3.8.7.1 DVI Interface

DVI (Digital Visual Interface) provides a low-cost, industry-standard, high-speed digital link between the chip and a DVI display such as a panel display or a TV set with DVI inputs. HDCP (High-bandwidth Digital Content Protection) is employed on-chip to prevent unauthorized copying of material. The HDCP keys are encrypted and stored in the write-only key RAM inside the chip. Both the upstream link (application interface) and the downstream link (display interface) are encrypted. The upstream protocol (defined in HDCP upstream link specification) ensures security of exchange between software and hardware. The internal HDCP hardware solution not only enhances the link security but also reduces the system cost.

The integrated I^2C port supports EDID, DDC, and HDCP down stream link exchange. Hot Plug detection is supported by the on-chip circuit. To connect to a TMDS DVI input display device, only an external TMDS transmitter device (e.g. SiI164) is needed (see *Figure 3-40*).

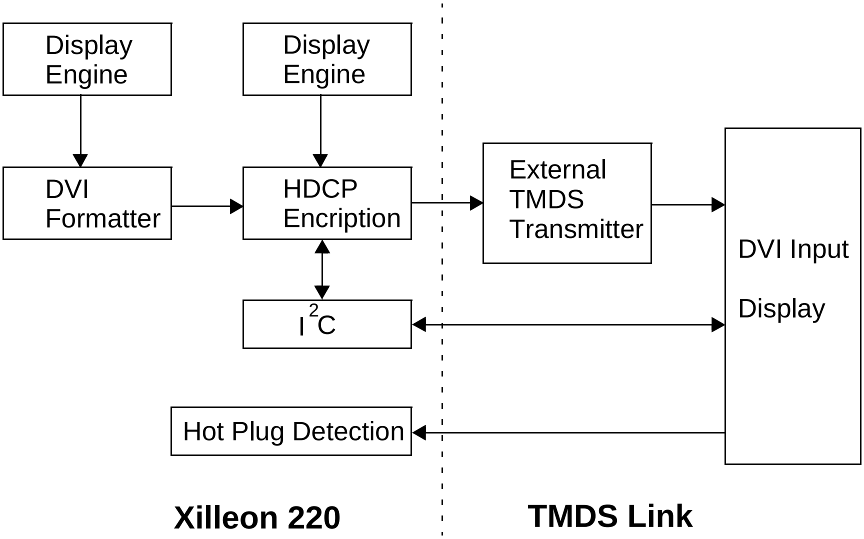


Figure 3-40 DVI Display Diagram

Xilleon 220 adopts the DVI 1.0 standard. The DVI formatter supports 18 bits pixel (RGB 6:6:6) and 24 bits pixel (RGB 8:8:8) output format. It also supports 2 or 4 gray levels frame modulation.

Interlaced timing is supported. The DVI display timing and pixel clocks are fully programmable. The maximum pixel frequency is 74.25MHz.

For detailed DVI output timing information, refer to *section 11.5: "DVI Out Timing" on page 11-32*.

The table below lists examples of modes that are supported.

Table 3-19 DVI Display Modes

Video Format	Refresh Rate
720x480i	60
640x480p	60
720x480p	60
1280x720p	60/50
1920x1080i	60/50
720x576p	50
720x576i	50

3.8.7.2 ITU-656 Out Interface

The ITU-656 digital output enables the Xilleon 220 to interface with an external display device such as an MPEG decoder, or a de-interlace device. The pixel rate is 13.5MHz. The output data stream rate and clock rate are 27MHz. The 4:2:2 output format is shown below.

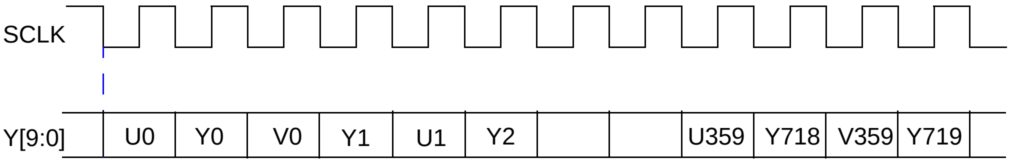


Figure 3-41 ITU-656 Output Format

The skew and the duty cycle of the output clock are programmable through the clock pattern registers. The ITU-656 interface supports both 8-bit and 10-bit data modes. The port is fully compliant with the recommendations of ITU-R BT.656-4. The ITU-656 data and clock output are multiplexed with DVI out pins and PCI bus pins.

For ITU-656 timing information, refer to *section 11.6.2: “ITU-656 Out” on page 11-34*.

3.8.8 Linearity, SNR, Jitter, Timing, and Color Considerations

TBA

3.8.9 Configuring for Different Broadcasting Standards

TBA

3.8.10 Mode Tables

3.8.10.1 Modes with 1 TV Out

TBA

3.8.10.2 Modes with 2 TV Out

TBA

3.8.10.3 Modes with DVI (656) Out

TBA

3.9 2D/3D Engines

3.9.1 2D Features

- Hardware acceleration of Bitblt, Line Draw, Polygon / Rectangle Fill, Bit Masking, Monochrome Expansion, Panning/Scrolling, Scissoring, full ROP support and hardware cursor (up to 64x64x2).
- Game acceleration including support for Double Buffering, Virtual Sprites, Transparent Blit, Masked Blit and Context Chaining.
- Acceleration in 8/16/24/32 bpp modes.

3.9.2 2D Engine Functional Description

This is a fixed-function unit that runs concurrently with the host processor. It is dedicated for drawing operations that include rectangle fill, line draw, polygon boundary lines, and polygon fill. A sophisticated pixel data path allows monochrome-to-two-color expansion, solid color fill, screen-to-screen bitblts, fixed pattern fill, general pattern fill, general patterns with rotation, and host-to-screen data transfers. Flexible bitblt trajectories allow hyper-efficient off-screen memory management, effectively increasing bitmap and font cache sizes and improving performance.

Other features include a quick setup GUI engine that off-loads draw engine setup from the host CPU. A 16-function ALU and a 4-function source/destination color comparator allow for the combinations of source and destination in a multitude of ways, useful for operations such as image overlaying or transparent blits. Bit masking and scissoring can protect memory regions from being written.

All internal draw engine data paths are 64-bit wide. Full drawing features are available in 8, 15, 16, 24, and 32 bits-per-pixel (bpp) modes.

All draw engine registers are 32-bit wide. A 512x32 command FIFO improves throughput over the expansion bus. The additional 128 entries are dedicated to busmastering of 16x64 source and destination FIFO improving memory bandwidth throughput.

3.9.3 3D Features

- Integrated triangle set-up engine reduces CPU and bus bandwidth requirements and dramatically improves performance of small 3D primitives.
- 4K on-chip texture cache dramatically improves large triangle performance.
- Complete 3D primitive support: points, lines, triangles, lists, strips and quadrilaterals.
- Comprehensive enhanced 3D feature set:
    - Full screen or window double buffering for smooth animation.
    - Hidden surface removal using 16-bit Z-buffering.
    - Sub-pixel and sub-texel accuracy.
    - Gouraud and specular shaded polygons.
    - Perspectively correct mip-mapped texturing with chroma-key support.

  - Support for single pass bi- and tri-linear texture filtering, vastly improving bi- and tri-linear performance.
  - Full support for texture lighting.
  - Special effects such as complete alpha blending, fog, video textures, texture lighting, reflections, shadows, spotlights, LOD biasing and texture morphing.
  - Dithering support in 16bpp for near 24bpp quality in less memory.
- Extensive 3D mode support:
  - Draw in RGBA32, RGBA16, and RGB16.
  - Texture map modes: RGBA32, RGBA16, RGB16, RGB8, ARGB4444, YUV444.
  - Compressed texture modes: YUV422.

3.9.4 3D Engine Functional Description

The 3D graphics engine offers a number of orthogonal pixel processing functions associated with the rendering of 3D images. These functions are chosen to accelerate features of drivers for major operating systems and APIs.

The Xilleon 220 graphics core includes a triangle setup engine. This engine needs only color, alpha, Z, U and V information at vertices of triangles to successfully draw Gouraud shaded, or perspectively correct, texture mapped triangles. The setup engine significantly reduces the CPUs load in 3D graphics applications, giving applications more CPU time to perform non-setup related tasks.

Pixels to be displayed can be further modified by alpha blending with pixels in the destination by fogging pixels with a fog color, and in the case of texture mapping by lighting them. Depth buffering is achieved by associating a 16-bit Z value with each pixel. The Z, alpha, and fog color for each new pixel is supplied from interpolators within the 3D coprocessor. In the case of texture mapping, the alpha factor may even be stored in the texture map on a pixel-by-pixel basis.

Pixels in the 3D engine are always operated on as 24-bit entities (8 bits each of Red, Green, and Blue). Other pixel sizes, i.e., 8-bit and 16-bit, are dithered by the 3D graphics subsystem to output at the desired pixel size.

The 3D engine contains a powerful texture mapping unit. This unit takes a series of precomputed maps (mip maps) and selects texels from these maps in a way that allows them to look perspectively correct. Texels can be filtered in a number of ways, and then lit (lightened or darkened). Once the texel is formed by filtering and lighting, any of the pixel processing modes mentioned previously can be applied to the texel.

A 4K texture cache greatly reduces the memory bandwidth needed to support texture mapping.

Chapter 4
MIPS Processor

4.1 Overview

Xilleon 220 has an integrated MIPS 4Kc processor core and associated North Bridge and South Bridge functionality to allow Xilleon 220 to run as a standalone set top system without the need for an external processor. The main function of the embedded core is to run firmware code that controls Xilleon 220’s various internal multimedia functional blocks, such as the MPEG decoder or the 2D/3D engine. It can also be used to run OS code (such as Linux) and user application code (such as internet browser) in configurations where the embedded MIPS is the only processor in the system.

The Xilleon 220 embedded MIPS top level block includes:

- MIPS 4Kc core and associated data and instruction cache.
- MIPS_MEINTF block that allows MIPS to access Xilleon 220 external memory and Xilleon 220 I/O devices.
- MIPS_HOST COM port that allows communication between Xilleon 220 internal MIPS and an external host processor through message passing.
- A Performance counter block that counts performance parameters in the MIPS core such as cache hits and misses.
- A MIPS support register file that contains all register bits that control the configuration of different blocks in the MIPS top level.

4.2 Features List

MISP 4Kc Core Features List

- MIPS 4Kc processor core supporting MIPS32 instruction set (MIPS3000/40000 compatible).
- 250 to 300 MHz operation.
- 16 KB 4-way set associative data cache.
- 16 KB 4-way set associative instruction cache.
- A TLB based memory management unit.
- Enhanced EJTAG interface.
- Fast multiply and multiply-accumulate instructions for DSP type of applications.

MIPS Memory Interface Features List

- Supports up to 128 MB system memory.
- Supports up to three concurrent outstanding memory accesses (1 instruction read, one data read, and one data write) for better memory access efficiency.
- External write buffer of 64 bytes deep for enhanced memory write performance.
- Data pre-fetch buffer of 32 bytes deep and instruction pre-fetch buffer of 32 bytes deep for enhanced memory read performance.
- Flexibility of running MIPS internal clock either synchronously or asynchronously with the memory clock.
- MIPS system memory can be allocated to a dedicated memory channel for a dual channel memory configuration or can be allocated to share memory channel with Xilleon 220’s MPEG decoder and 2D/3D engine in a single channel memory configuration.
- Programmable memory access priority for overall system performance.

MIPS I/O Interface Features List

- Fast read and write interface to Xilleon 220 internal registers.
- Full access to all the Xilleon 220 internal functional blocks and external buses.

MIPS_HOST Communication Port Features List

- Two-way communication port for passing messages between external host processor and Xilleon 220 internal MIPS in Peer mode.
- DWORD sending buffer and 1 DWORD receiving buffer for each processor.
- Operation either polling or interrupt driven.

Performance Counter Features List

- Six 32-bits independent counters to count six 4Kc core performance parameters at the same time.
- Can choose to monitor two sets of performance parameters, with one set concentrating on cache performance and the second set concentrating on TLB performance.

4.3 Xilleon 220 Embedded MIPS Functionality

4.3.1 Xilleon 220 Embedded MIPS Top Level Diagram

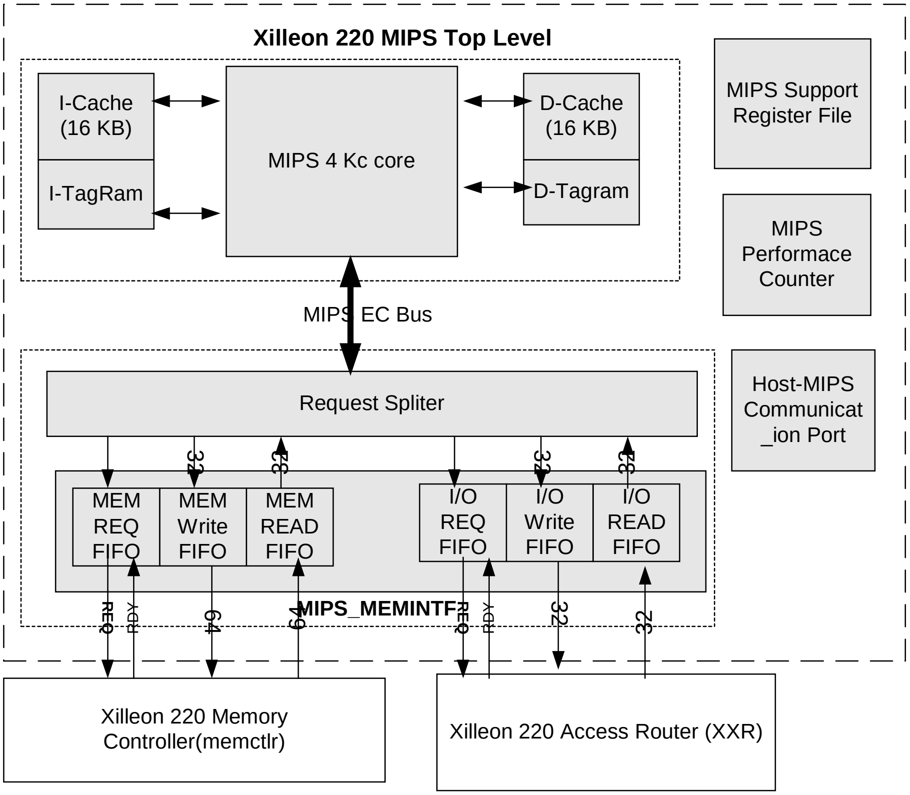


Figure 4-1 Xilleon 220 Embedded MIPS Top Level Diagram

Figure 4-1,"Xilleon 220 Embedded MIPS Top Level Diagram," on page 3 shows the five major functional blocks inside the top level of the ATI Xilleon 220 MIPS core. They are: (1) MIPS4Kc core with associated 16 KB data cache and 16 KB instruction cache; (2) MIPS _MEMINTF block that interfaces to the Xilleon 220 memory controller and Xilleon 220 access router; (3) MIPS support register file that contains control register bits for all blocks in the Xilleon 220 MIPS core top levels, such as address ranges for memory access and I/O access used by the MIPS_MEMINTF block; (4) MIPS performance counter that interfaces to the MIPS 4Kc performance monitor interface and counts performance indicators such as cache hits and cache misses; (5) Host –MIPS communication port block that is used to pass messages between the Xilleon 220 internal MIPS processor and an external host processor.

4.3.2 Theory of Operation

4.3.2.1 MIPS Core

MIPS 4Kc core is a general purpose 32-bit RISC CPU core, designed by MIPS Technologies, specifically for system on a chip (SoC) applications. It executes the MIPS32 Instruction Set Architecture (ISA). In addition to the base implementation of the 4 Kc core, the MIPS 4 Kc core in the Xilleon 220 chip also includes the 16 KB instruction cache, 16 KB data cache, and EJTAG debugging port which are optional for the original core. *Figure 4-2,“Block Diagram of MIPS 4 Kc Core in the Xilleon 220.,” on page 4* shows the major functional blocks of the MIPS 4Kc core in Xilleon 220 chip. Detailed description of the MIPS 4Kc core can be found in the MIPS32 4K Processor Core Family User’s Manual.

The execution unit implements a load store architecture with 32 general-purpose registers. The four-stage pipe lined execution unit has a throughput of one instruction per cycle for most instructions. Please refer to Chapter 2 of MIPS32 4K Processor Core Family User’s Manual for details.

The multiply-divide unit implements a fast multiplier, with 32x16 multiply instructions taking one clock cycle and 32x32 multiply instructions taking two clock cycles. The multiply-add and multiply-subtract instructions are also supported for DSP functionalities. The divide uses a simple 1-bit per-clock iterative algorithm, which requires 35 clocks to finish a single divide instruction.

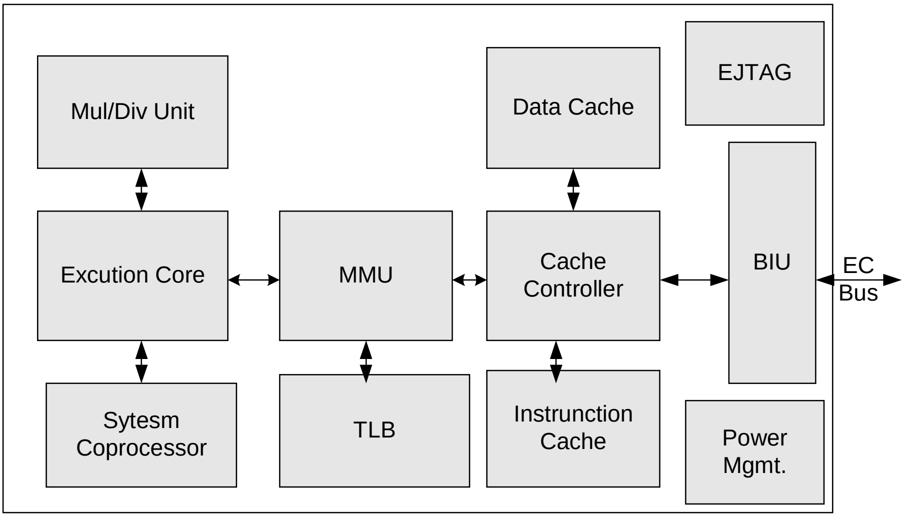


Figure 4-2 Block Diagram of MIPS 4 Kc Core in the Xilleon 220.

The system coprocessor 0 (CP0) implements system functionalities such as interrupt masking, CPU configuration and modes of operation (kernel vs. user) as well as EJTAG debugging. CP0 functionality is accessed through CP0 registers. Please refer to Chapter 5 of MIPS32 4K Processor Core Family User's Manual for more details.

The memory management unit (MMU) implements a TLB based address translation scheme, which is required by OS such as Linux and WINCE. It translates the MIPS virtual address used by the OS and application software into the MIPS physical address, which is used by hardware, and that appears on the MIPS external bus. Please refer to Chapter 3 of MIPS32 4K Processor Core Family User's Manual for more details.

The cache controller contains the logic to control separate data and instruction caches. Both the data and instruction caches are configured as 4 way set associative. The size of each cache is 16 KB. The cache line is 16 bytes. The cache writes policy is writing through. Please refer to Chapter 7 of MIPS32 4K Processor Core Family User's Manual for more details.

The bus interface unit (BIU) is the MIPS external bus controller which implements the MIPS EC32 bus protocol. The EC32 bus is a 32-bit high performance bus. It supports burst data transfer as well as concurrent outstanding transactions on the bus for high throughput and high memory access efficiency. The MIPS 4Kc EC bus allows up to three (one instruction read, one data read and one data write) transactions to be active on the bus at the same time. The BIU also contains 2 sets of 16 bytes write buffers. In conjunction with the 64 bytes (4 set of 16 bytes) external write buffer, it greatly reduces the core stall due to latencies writing to memory.

The MIPS 4 Kc core in Xilleon 220 supports two modes of power management:

- Register controlled power management
- Instruction controlled power management

With register controlled power management, three bits in the CP0 register status can be used by the software to control the power down mode as well as also to allow the interrupt to be serviced in the power down mode. With instruction controlled power management, the WAIT instruction puts the MIPS core into power down mode. Please refer to Chapter 8 of MIPS32 4K Processor Core Family User's Manual for more details.

The EJTAG port is supported in the 4 Kc core in Xilleon 220. It provides an industry standard 5 pin EJTAG debug port (the 5 pins are TCLK, TMS, TDI, TDO and T_RST_N). The EJTAG port can be used with popular on chip debugging tools (such as Green Hill's IDE) for on chip code debugging (OCD). OCD allows software developers to read and write MIPS internal registers, read and write memory, single-step through code, and set breakpoints without using a software monitor or operating system. Please refer to Chapter 9 of MIPS32 4K Processor Core Family User's Manual for more details.

4.3.2.2 MIPS_MEMINTF

The MIPS_MEMINTF block in the Xilleon 220 MIPS top level functions as a CPU's North Bridge. It is a bridge between the MIPS external bus interface, the EC bus interface, the Xilleon 220 memory controller as well as Xilleon 220 I/O interface, the Xilleon 220 access router.

Apertures

The MIPS_MEMINTF separates EC bus requests into either memory access requests or I/O access requests, based on a predefined memory address range (called an aperture). The separated requests are forwarded either to the Xilleon 220 memory controller or the Xilleon 220 access router. The aperture is an address range in the MIPS physical address space. An aperture is completely defined by a base address and a size. The base address is the starting address of an aperture in MIPS physical address space. The size of an aperture is the difference between the end of the aperture address and start of aperture address.

The MIPS_MEMINTF block uses three apertures for the purpose of splitting requests into memory requests and I/O requests. *Figure 4-3,"Apertures Used by MIPS_MEINTF," on page 7* shows the mapping between different apertures in the MIPS physical address space to the Xilleon 220 RAM and the Xilleon 220 access router. The first is the system aperture, which defines the random access memory (RAM) area that MIPS uses for run time code and data. EC bus requests that fall in the MIPS system aperture range are forwarded to Xilleon 220 memory controller. Normally, the system aperture base is the MIPS physical address 0x0. The aperture size is programmable with four options: 16 MB, 32 MB, 64 MB, and 128 MB.

The second aperture is the MIPS RAM boot aperture. This is a special MIPS system memory area in which the MIPS boot code resides. It starts at the MIPS physical address 0x1fc00000 (508 MB). In configurations such as the peer mode, where there is an external host CPU responsible for setting up the MIPS run time code and data, the boot code can also be placed in the Xilleon 220 RAM by the host CPU. However, the boot code address range, which starts at 0x1fc000000 (508 MB), is not in the same physical address range as the normal run time code and data, which normally starts at MIPS physical address 0x0. A separate aperture range called the MIPS RAM BOOT aperture is used to map the boot code into Xilleon 220 RAM, as shown in *Figure 4-3*. In this configuration, a separate boot ROM is not needed for MIPS. The RAM boot aperture can be enabled or disabled by the host by programming the RAM_BOOT_APERTURE_ENABLE bit in the RAM_BOOT_APERTURE register. By default, the RAM BOOT APERTURE is disabled. In solo mode configuration, the MIPS can only boot from ROM that hooks up on one of the Xilleon 220 chip's I/O bus. The RAM boot aperture is always disabled in solo mode.

The third aperture is Xilleon 220 internal register aperture. All Xilleon 220 internal registers, such as MPEG decoder registers and 2D/3D engine registers, fall in this range. The Xilleon 220 register access is defined as a special type of I/O access and requests are forwarded to the Xilleon 220 access router where it is routed to the register interface block. The base of the Xilleon 220 internal register aperture is defaulted to the MIPS physical address 0x18000000(256 MB), and the size of the aperture is 64 KB.

All other accesses that fall outside of these three apertures are considered I/O accesses by the MIPS_MEMINTF and all are forwarded to the Xilleon 220 access router where it is further divided into different apertures, such as the frame buffer aperture and PCIC aperture. The access router is responsible for routing access in different I/O apertures to different I/O devices, either internal or external of the Xilleon 220 chip.

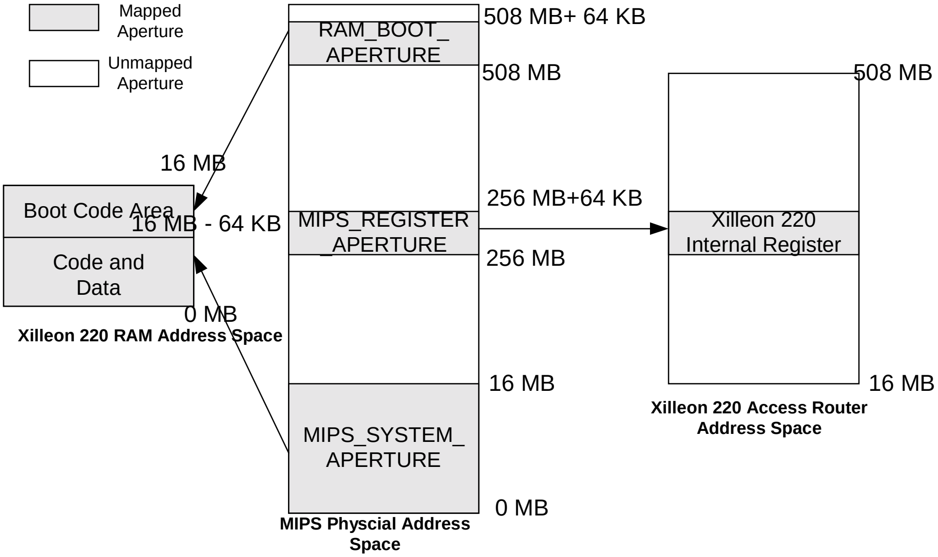


Figure 4-3 Apertures Used by MIPS_MEINTF

Figure 4-3 shows three apertures used by the MIPS_MEINTF block to route MIPS requests to either the Xilleon 220 memory controller or the Xilleon 220 access router. Requests that fall in the MIPS_SYSTEM_APERTURE or the RAM_BOOT_APERTURE are routed to memory controller. Requests that fall in the MIPS_REGISTER_APERTUE or that do not fall in any of the three apertures (in *Figure 4-3*) are forwarded to Xilleon 220 access router.

The Write Buffer

The MIPS_MEMINTF contains a write buffer and read pre-fetch buffers to optimize the MIPS memory access efficiency. The memory write buffer is configured as 4 sets of 16 bytes each. It can hold up to four burst write transfers (16 bytes each) from the MIPS EC bus. The write buffer improves memory access performance by allowing the MIPS to continue subsequent write or read transactions as long as the write buffer is not full. Without a write buffer, MIPS would have to stall and wait until a single write transaction is complete, which may take many clocks cycles depending on whether the memory controller is servicing other memory access clients in the chip, before starting the next transaction.

The write buffer has a control register that controls the flush point, which is defined as the number of valid entries in the write buffer before requests are made to the memory controller. The flush point register bits are located inside the MIPS_MEMINTF_CNTL register. The flush point value ranges from 1 to 4, where 1 corresponds to making a request after one valid entry, and 4 corresponds to making a request when the write buffer becomes completely full. The default value is two, which is the optimal value in the majority of cases. The write buffer can also be flushed by software control. In case the software wishes to move all valid data in the write buffer to memory, the MIPS instruction SYNC is used. The execution of the SYNC instruction would cause a flush of all the valid entries in the write buffer to memory.

Read Pre-fetch Buffers
The MIPS_MEMINTF block implements read pre-fetch buffers to improve sequential memory read access efficiency. When enabled, the MIPS_MEMINTF pre-fetches the next piece of data in the sequence into the pre-fetch buffer whenever a memory read request is made. If a subsequent memory read request falls in the pre-fetch address range, the read data will come from the read pre-fetch buffer with an access latency of only one clock cycle rather than many clock cycles if the data were to come from memory. The pre-fetch buffer is separated into data reads and instruction reads and each contains 32 bytes. The MIPS_MEMINTF_CNTL register contains an enable bit for each instruction pre-fetch and data pre-fetch for enabling and disabling the pre-fetch buffer functionality.

4.3.2.3 MIPS HM COM port

The HM_COM port is a bi-directional communication port that allows the Xilleon 220 internal MIPS processor and an external host processor to pass messages to each other.

The HM_COM port logically contains a one DWORD sending buffer and a one DWORD receiving buffer for each of the processors (total four one DWORD buffers). A processor can send a message by writing to its sending buffer and can receive a message by reading from its receiving buffer. Each buffer has an associated control flag. For each sending buffer, there is an empty flag associated with the message sending processor that checks whether the previous message has been retrieved by the receiving processor before sending any new data. For each receiving buffer, there is a valid flag that the receiving processor can check before retrieving a message. The flags are set and cleared by the HM_COM port hardware when a message is sent or retrieved.

The sending buffer of one processor is actually the receiving buffer of the other processor. Therefore, there are only two physical buffers existing in the HM_COM port. The two physical buffers are named following the host side convention. They are: HOST_WRITE_BUFFER (host side sending buffer and MIPS side receiving buffer) and HOST_READ_BUFFER (host side receiving buffer and MIPS side sending buffer). The block diagram of the HM_COM port is shown in *Figure 4-4,“Top Level of HM_COM port.,” on page 9*.

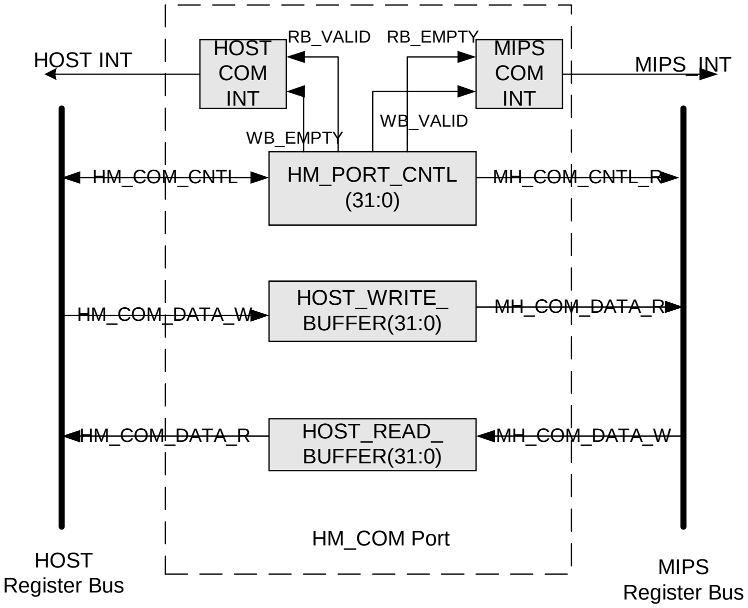


Figure 4-4 Top Level of HM_COM port.

Figure 4-4 shows the top level of the HM_COM port. The HM_COM port logically consists of one receive buffer and one sending buffer for each processor (total logical four buffers) that maps to two physical one DWORD buffers. The two logical buffers for each processor are mapped to their respective processor register space. The host and MIPS use two logical buffers to send and receive messages. A single control register that contains the port enable bit, interrupt enable bits and buffer flags are also mapped into both the host and MIPS register space. The buffer flags can also be used to generate interrupts for interrupt driven data communication.

A processor can initiate communication by writing a DWORD to its sending buffer. For example, assume that the external HOST processor wishes to send a message to the Xilleon 220 internal MIPS processor. The host writes a DWORD to its sending buffer HOST_WRITE_BUFFER. The HM_COM port hardware raises the MIPS receiving buffer valid flag (HM_WB_VALID =1) to tell the MIPS processor that there is a message in its receiving buffer. The MIPS can subsequently read its receiving buffer, which is the same as the HOST_WRITE_BUFFER, to get the message. After MIPS reads the message, the HM_COM port clears the receiving buffer valid flag (HM_WB_VALID =0) and sets the sending buffer empty flag to one (HM_WB_EMPTY =1). One round of communication is completed.

The sending buffer and receiving buffer for each processor in the HM_COM port are mapped into the respective processor's register space. Each processor's reading and writing of its sending and receiving buffers is like reading and writing normal 32 bit registers. The sending buffer and receiving buffer are mapped into the same address so that writing to the register is the same as writing to the sending buffer and reading from the register is the same as reading from the receiving buffer. The HOST side register is called HM_COM_DATA, which is described in *Table 4-1 on page 10,* and the MIPS side register is called MH_COM_DATA, which is described in *Table 4-2 on page 10*.

The buffer control flags are mapped into a 32-bit control register. The control register also contains the HM_PORT enable bit for enabling and disabling of the HM_COM port as well as the interrupt enable bit for interrupt driven communication. The single control register is mapped into the address space of both processors. On the host side, it is called HM_COM_CNTL, and on the MIPS side, it is called MH_COM_CNTL. *Table 4-3 on page 10* describes the HM_COM_CNTL register and *Table 4-4 on page 11* describes the MH_COM_CNTL register. The host side register HM_COM_CNTL is the master. The host side is responsible for enabling and disabling the HM_COM port as well as setting up the mode of communication, either polling or interrupt driven. The MIPS side control register is a slave and an exact mirror of the host side register. It is read only. The MIPS can use the buffer flag to determine when to send and receive messages. The MIPS is not responsible for setting up the communication port.

Table 4-1 Host Side HM COM Port Data Register

HM_COM_DATA - 32 bits			
Field Name	**Bits**	**Default**	**Description**
HM_COM_DATA_W	31:0	No	Host CPU sending buffer
HM_COM_DATA_R	31:0	No	Host CPU receiving buffer

Table 4-2 MIPS Side HM COM Port Data Register

MH_COM_DATA - 32 bits			
Field Name	**Bits**	**Default**	**Description**
MH_COM_DATA_W	31:0	No	Host CPU sending buffer
MH_COM_DATA_R	31:0	No	Host CPU receiving buffer

Table 4-3 Host Side HM COM Port Control Register

HM_COM_CNTL- 32 bits			
Field Name	**Bits**	**Default**	**Description**
HM_COM_PORT_EN	0	0x0	Enable the HM com port. Default is disabling.

Table 4-3 Host Side HM COM Port Control Register (Continued)

HM_COM_CNTL- 32 bits			
Field Name	**Bits**	**Default**	**Description**
HM_WB_EMPTY	8	0x1R	Host write buffer empty flag
HM_RB_VALID	9	0x0R	Host read buffer valid flag
HM_WB_VALID	12	0x0R	MIPS read buffer valid flag
HM_RD_EMPTY	13	0x1R	MIPS write buffer empty flag
HM_WB_EMPTY_INT_EN	16	0	Enable HOST write buffer empty flag to generate an interrupt to HOST CPU
HM_RB_VALID_INT_EN	17	0	Enable HOST read buffer valid flag to generate an interrupt to HOST CPU
HM_WB_VALID_INT_EN	20	0	Enable MIPS read buffer valid flag to generate an interrupt to MIPS
HM_RB_EMPTY_INT_EN	21	0	Enable MIPS read buffer empty flag to generate an interrupt to MIPS

Table 4-4 MIPS Side HM COM Port Control Register

MM_COM_CNTL_R- 32 bits			
Field Name	**Bits**	**Default**	**Description**
HM_COM_PORT_EN	0	0x0	Enable the HM com port. Default is disabling.
HM_WB_EMPTY	8	0x1R	Host write buffer empty flag
HM_RB_VALID	9	0x0R	Host read buffer valid flag
HM_WB_VALID	12	0x0R	MIPS read buffer valid flag
HM_RD_EMPTY	13	0x1R	MIPS write buffer empty flag
HM_WB_EMPTY_INT_EN	16	0	Enable HOST write buffer empty flag to generate an interrupt to HOST CPU
HM_RB_VALID_INT_EN	17	0	Enable HOST read buffer valid flag to generate an interrupt to HOST CPU
HM_WB_VALID_INT_EN	20	0	Enable MIPS read buffer valid flag to generate an interrupt to MIPS
HM_RB_EMPTY_INT_EN	21	0	Enable MIPS read buffer empty flag to generate an interrupt to MIPS

4.3.2.4 MIPS Performance Counter

The MIPS performance counter block in the Xilleon 220 MIPS top level implements six independent 32 bit counters that hook up to the MIPS 4Kc performance monitor (PM) interface. The purpose of the performance counter is for the software developer to monitor performance using factors such as cache hits, cache misses and/or execution time of the software code under development. Sometimes, there may be more than six PM signals that need to be monitored. For example, if there are ten PM signals, then the performance counter blocks will separate the ten PM signals into two groups. One group of performance monitors can be counted at a time. The group selection bit, MIPS_PM_CNT_SEL, is in register MIPS_CNTL. The six performance counters can be enabled by writing one to the MIPS_PM_CNT_EN bit in the MIPS_CNTL register. The six performance counters can be stopped by writing zero to the MIPS_PM_CNT_EN bit in the MIPS_CNTL register. To clear the six performance counters, write one to the MIPS_PM_CNT_CLEAR bit in the MIPS_CNTL register. The first group concentrates on the cache performance monitor. The six counters and counter increment conditions for this first group are shown below:

1. Instruction cache hit. The counter increments by one when there is an instruction cache hit.
2. Instruction cache miss. The counter increments by one when there is an instruction cache miss.
3. Data cache hit. The counter increments by one when there is a data cache hit.
4. Data cache miss. The counter increments by one when there is data cache miss.
5. Write merging. The counter increments by one when the write operation is a merge operation in the 4Kc internal write buffer.
6. Write not merge. The counter increments by one when the write to the 4Kc internal write buffer is not a merge operation.

The second group concentrates on monitoring the table look aside buffer (TLB) performance. The six counters and counter increment conditions are shown below:

1. TLB hit. The counter increments by one when there is a TLB hit.
2. TLB miss. The counter increments by one when there is a TLB miss.
3. JTLB hit. The counter increments by one when there is a JTLB hit.
4. JTLB miss. The counter increments by one when there is a JTLB miss.
5. Write merging. The counter increments by one when the write operation is a merge operation in the 4Kc internal write buffer.
6. Write not merge. The counter increments by one when the write to the 4Kc internal write buffer is not a merge operation.

Chapter 5
PCI Interface

5.1 PCI Controller (Host Bridge)

5.1.1 Overview

The PCI Controller (PCIC) functions as the "Host Bridge" for the MIPS processor. Host Bridge is a term from the PCI Bus Specification, and is used to describe an agent of the PCI bus that connects a processor (usually with some locally-addressable RAM) to a PCI bus segment. A Host Bridge does not implement the standard PCI configuration registers; it is typically the agent that generates the PCI configuration cycles during system initialization.

For Xilleon 220 in SOLO Mode, the Host Bridge acts as an Initiator on the external PCI bus, to allow the MIPS processor to actively access devices on the PCI bus. It also acts as a Target on the external PCI bus, allowing external PCI devices (as well as PCI devices that are internal to the Xilleon 220) to gain access to the SDRAM managed by the Xilleon 220 chip.

For Xilleon 220 in PEER-Plus Mode, the Host Bridge acts as an Initiator on the internal PCI bus of the Xilleon 220, allowing the MIPS processor to actively access the internal peripheral devices of the Xilleon 220 (namely, IDE, USB, LPC, and DAIO Controllers). It also acts as a Target on the internal PCI bus, allowing those same peripheral devices to gain access to the SDRAM managed by the Xilleon 220 chip.

In PEER Mode, the Host Bridge is inactive; the internal MIPS processor cannot directly access the external PCI bus, under program control. That is, the PCI bus does not appear in the MIPS' address space.

The PCI Controller also supplies the "Central Resource" functions for PCI. The term "Central Resource" is also from the PCI Bus Specification. The specific functions that the PCIC provides are:

- Central arbitration
- Generation of PCI configuration transactions
- Generation of IDSEL signals to each PCI device

The Xilleon 220 can be viewed as supporting either one or two PCI buses, depending on the operating mode. In SOLO Mode and PEER Mode, there is a single PCI bus that is shared between the external devices and the Xilleon 220-internal PCI devices. In PEER-Plus Mode, there are two separate PCI buses — one that goes off-chip (connecting the external devices with the Multimedia subsystem), and one that stays entirely on-chip (connecting the internal PCI devices with the PCIC/Host Bridge).

When in PEER-Plus Mode, the PCIC provides the Central Resource functions for the internal PCI bus; whereas in SOLO Mode, the PCIC provides Central Resource functions for the external PCI bus. Since the PCIC is inactive in PEER Mode, it does not supply any

Central Resource functions when in that mode. See the topic on PCI Arbitration *on page 5-5* for more details about the PCI bus topology in these modes.

5.1.2 Feature List

- Compliance with PCI Local Bus Specification, Revision 2.2
- 32-bit bus
- Synchronous PCI bus operation up to 66 MHz
- 3.3 V PCI only
- Master/Slave capability on PCI bus
- PCI arbitration
- Three PCI interrupt inputs
- Endian Swap for Master transactions

5.2 Functional Description

5.2.1 Block Diagram

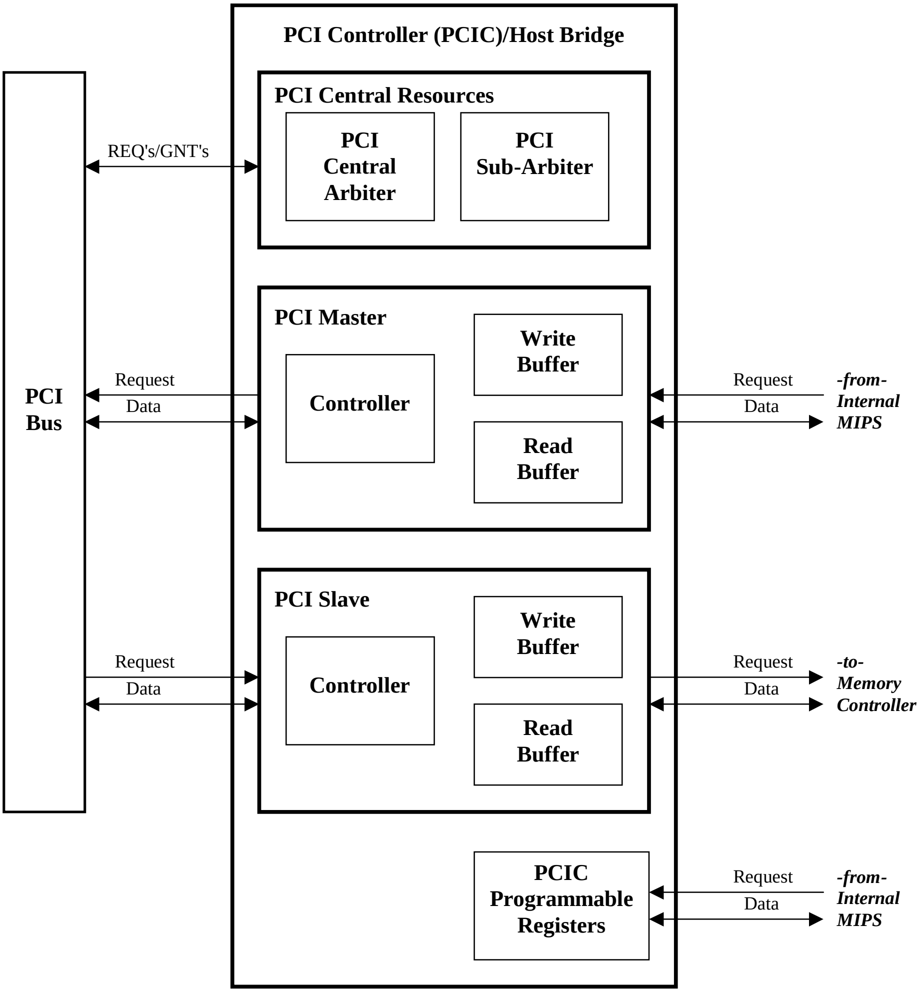


Figure 5-1. Block Diagram of PCI Controller

5.2.2 Theory of Operation

The PCI Controller is composed of four primary sub-blocks: Master, Slave, Central Resources, and Programmable Registers.

The table below provides a quick reference, indicating how the various programmable registers of the PCIC affect each of the sub-blocks in the Block Diagram.

Table 5-1 Effect of PCIC Registers on other PCIC Sub-Blocks

Registers	PCI Master	PCI Slave	Central Resources	General
PCIC_BM_STATUS	X			
PCIC_BUSMASTER_CNTL	X			
PCIC_BUSSLAVE_CNTL		X		
PCIC_BUS_CNTL	X	X	X	
PCIC_COMMAND_STATUS	X	X	X	X
PCIC_DEBUG_CNTL	X	X	X	X
PCIC_DISCARD_CNTL		X		
PCIC_INT_ENABLE				X
PCIC_LINE_LATENCY	X	X		
PCIC_MEM_BASE0		X		

PCI Slave Module

The PCI Slave module monitors transactions on the PCI bus. When a memory-space transaction is a "hit" in the aperture defined by the PCIC_MEM_BASE0 register, the PCIC responds to the PCI transaction, and forwards the request to the memory controller, for access to the SDRAM. The PCIC_MEM_BASE0 register defines a base address and a size for an aperture in the PCI memory-space address map.

Note that before the Slave module becomes active, it first must be set up, or configured by the internal MIPS processor. That is, software running on the MIPS "opens up" an aperture to the SDRAM memory-space (by programming the PCIC_MEM_BASE0, and PCIC_COMMAND_STATUS registers), thus allowing initiator devices on the PCI bus to access the SDRAM memory.

The PCIC Slave supports all sixteen PCI command types. However, it only responds to the memory-type of commands. Memory Read Multiple and Memory Read Line are aliased to Memory Read, Memory Write, and Invalidate is aliased to Memory Write. All the read and write requests are sent through a common request queue, which does not allow re-ordering. The read and write data are sent through FIFO buffers.

Version 2.2 of the PCI Bus Specification requires that Host Bridge targets respond to memory accesses within 32-clocks for the first data item, and within 8-clocks for each subsequent data item of a burst. (Note that Host Bridges are a special-case target; all other types of targets must respond within 16 clocks initially). The Xilleon 220 has a programmable field (PCIC_BUSSLAVE_CNTL . BUS_RETRY_WS) that controls how many clocks the PCIC should wait, before replying with the RETRY response on the PCI bus.

The PCI Slave module implements an intelligent pre-fetch mechanism for memory reads. It contains a pre-fetch cache of 128 bytes, organized as 8 lines of 16-bytes each. If a read transaction is not a hit in the pre-fetch cache, then the PCIC will terminate the PCI cycle with a RETRY. This is called a "delayed read" transaction, in PCI parlance. The PCIC can track up to four outstanding delayed read requests at any given time. The PCIC_DISCARD_CNTL register is used to control what the PCIC does with outstanding read requests that are never retried by the initiator. This mechanism provides optimum performance for systems that have multiple PCI initiators that desire access to the SDRAM.

The pre-fetch cache also has a read-ahead capability that enables it to support long burst reads with few or no wait states once the burst has begun. Certain parameters of the prefetch size and algorithm are programmable via fields in the PCIC_BUSSLAVE_CNTL register. See the register reference for details.

PCI Master Module

The PCI Master module receives read/write requests from the internal MIPS processor, and generates the appropriate PCI transaction to fulfill the request. Since the MIPS can only generate "read" and "write" requests, there is a programmable register field (APER_CP_PCIC[12]_CNTL . PCI_ATYPE) that translates the read and write requests into one of the 16 PCI command types. The PCIC_BUSMASTER_CNTL register has fields that control read/write combining and other aspects of the master operation.

Follow these rules to allow the MIPS to generate PCI configuration cycles:

If PCI_ATYPE is 3, then the output address of the CP_PCIC[12] aperture is interpreted as:

> [10:0] get placed directly onto bits 10:0 of the PCI`s AD-bus during the address-phase.
>
> [15:11] get decoded into a one-hot vector that is placed onto bits 31:11 of the PCI`s AD-bus during the address-phase. (The DEVSEL inputs of the PCI targets in the system are usually wired to one of these upper address bits; The `internal PCI modules of the Xilleon 220 (ie, HBIU, IDE, USB, LPC, DAIO) use AD[31] as their DEVSEL. So, to select the `internal PCI` modules of Xilleon 220 for PCI config cycles, set address bits 15:11 to 0x14, which will assert AD[31].)

A typical setting for the APERSIZE field would be 0xB (64KB), which would allow the MIPS to access the Configuration Space of all of the PCI devices on a single PCI bus segment, without having to change the APER_CP_PCIC[12]_ADDR . BASE_OUT register field.

Central Resource Module

The primary purpose of the Central Resource module is to provide bus-arbitration support for the initiators present on the PCI bus. To support the various operating modes of the Xilleon 220 (SOLO, PEER, PEER-Plus), there are two distinct arbiters inside this module: the Central

Arbiter, and the Sub-Arbiter. Refer to figures *Figure 5-2.* through *Figure 5-4.* for a view of how the arbiters operate in each of the modes.

The Central Arbiter takes in requests from up to nine initiators (some internal to Xilleon 220, and some external), performs arbitration, and generates the "grant" signal back to the winning initiator.

The Sub-Arbiter takes in requests from up to seven initiators (some internal to Xilleon 220, and some external), "collects them together" into a single request to an external Central Arbiter (elsewhere in the system). Then, when the "grant" comes back from the system's central arbiter, the Sub-Arbiter performs arbitration to determine a winner, and then routes the "grant" signal back to that initiator.

Both arbiters use a simple, single-level round-robin arbitration algorithm, giving fair and equal access to all requesters.

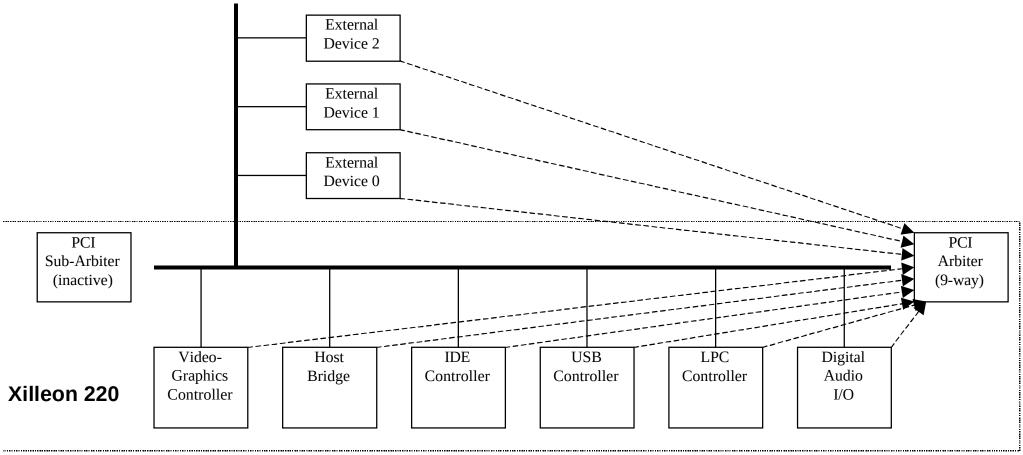


Figure 5-2. PCI Arbitration: Solo Mode

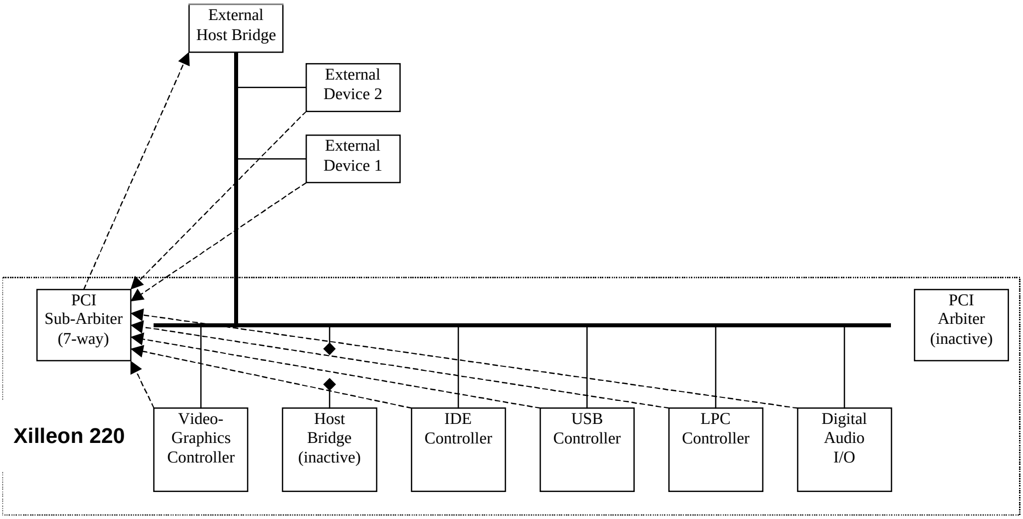


Figure 5-3. PCI Arbitration: Peer Mode

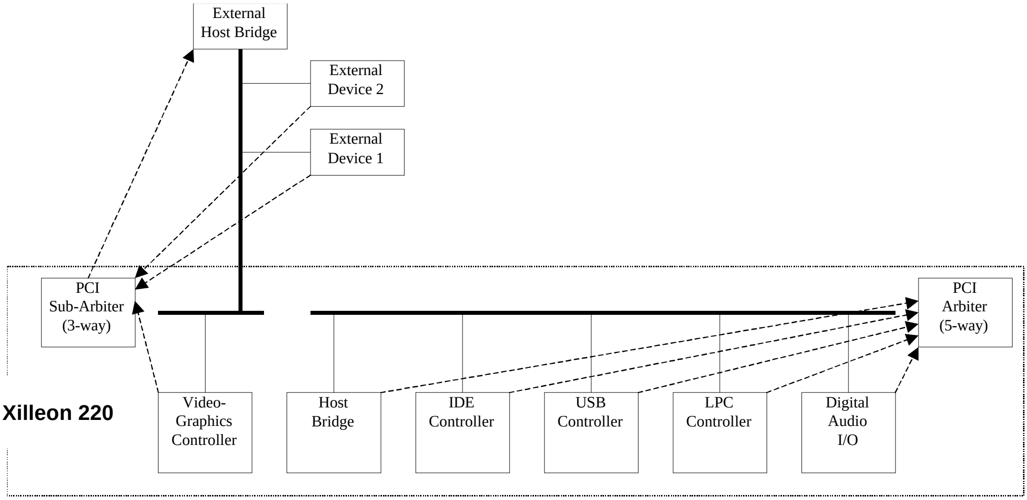


Figure 5-4. PCI Arbitration: Peer Plus Mode

Registers Module

The Registers module contains the programmable registers of the PCIC.

Interrupts related to the PCIC are enabled by setting the appropriate bit(s) in the PCIC_INT_ENABLE register, and the status of an interrupting condition can be read from the upper half of the PCIC_COMMAND_STATUS register. Writing a '1' to the associated bit in the PCIC_COMMAND_STATUS register clears the interrupting condition.

5.3 External Chip Interfaces

5.3.1 PCI Bus Interface

See *Chapter 9* for pin-out information, and *Chapter 11* for timing and protocol information, and .

Chapter 6
PCI Peripherals

6.1 Introduction

This chapter covers the following PCI peripheral interfaces:
(See *Chapter 7* for other peripheral interfaces.)

- *"Enhanced Integrated Drive Electronics (EIDE) Controller" on page 6-2*
- *"Universal Serial Bus (USB) Controller" on page 6-10*
- *"Low-Pin Count (LPC) Interface Controller" on page 6-17*
- *"AC-Link Interface" on page 6-21*

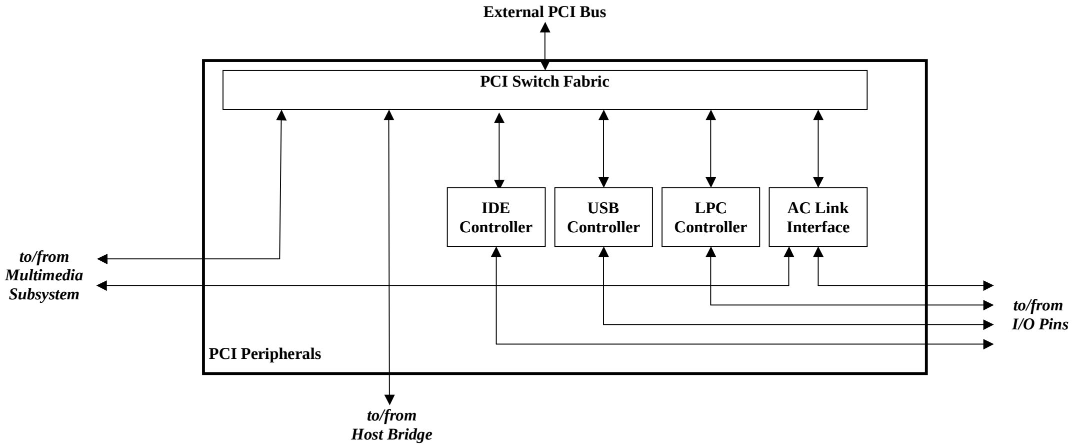


Figure 6-1 Block Diagram of the Peripheral Controller Unit

6.2 Enhanced Integrated Drive Electronics (EIDE) Controller

The integrated EIDE controller is compliant with the ANSI-NCITS ATA/ATAPI-5 Rev a1 (May 26, 1999) specification, and implements a single primary EIDE bus. It supports all PIO and Ultra DMA modes, with operating speed up to 100MHz. First-party scatter/gather DMA is built into the EIDE controller.

6.2.1 Feature List

- Compliance with the ANSI-NCITS ATA/ATAPI-5 Rev a1 (May 26, 1999)
- Support for both Native and Compatibility Mode register accesses
- PCI Interface that supports up to 2 IDE devices
- 100 MByte/sec maximum IDE transfer rate
- Ultra DMA mode with scatter/gather capability
- PIO modes 0-4 support
- Multiword DMA modes 0-2 support
- Ultra DMA modes 0-5 support
- IORDY handshaking for PIO modes

The maximum transfer rate of 100 MBytes/sec is achieved using Ultra DMA mode 5.

6.2.2 Functional Description

6.2.2.1 Block Diagram

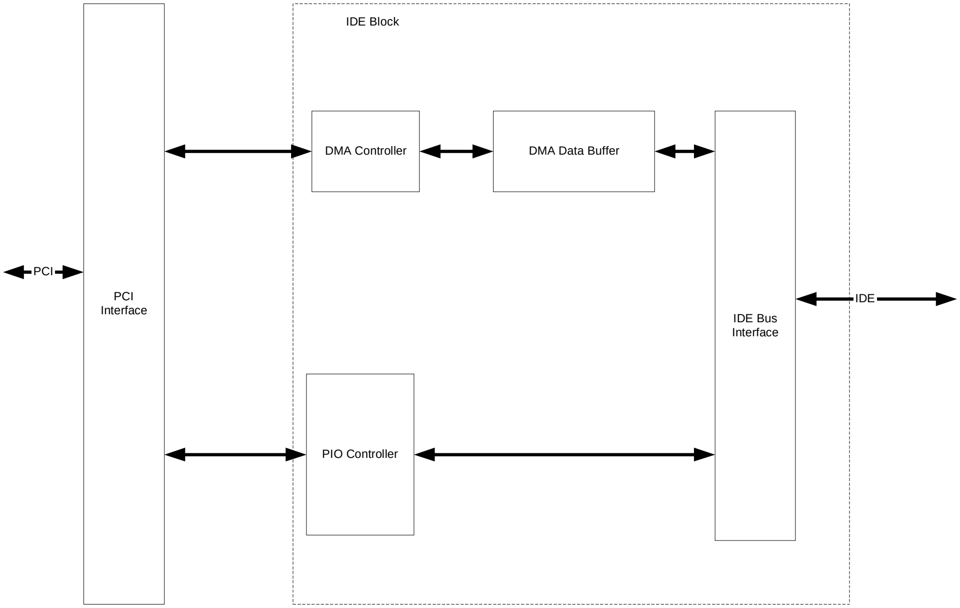


Figure 6-2 EIDE Controller Block Diagram

6.2.2.2 Theory of Operation

The integrated EIDE controller has two main components, the PCI interface, and the IDE related circuitry. The PCI interface supports two IDE devices on a single IDE channel. All of the required PCI registers are implemented to support PIO modes 0-4, Multiword DMA modes 0-2, and Ultra DMA modes 0-5.

The IDE related circuitry is comprised of four sections. Namely, the PIO controller, the DMA controller, the DMA data buffer, and the IDE bus interface. Together these sections handle the signal generation for the IDE interface signals in all modes as well as the transferring of data in all modes. This includes the buffering of data in DMA modes.

The timing of the IDE interface control signals and the determination of which controls are used is accomplished by the PIO and DMA controllers. The controllers require setup information which is passed from registers in the PCI interface block to the IDE interface block. Each controller uses the register information in the PCI interface pertaining to its type of transaction to correctly time the control signals on the IDE Interface.

In order to support DMA operations across the PCI bus, some data buffering is required. This buffering is provided by the DMA data buffer for both read data and write data.

The IDE bus interface provides input signal synchronization and final selection of some of the output signals on the IDE bus.

6.2.2.3 Primary Chip Interface

The physical interface consists of receivers and drivers communicating through a set of conductors using an asynchronous interface protocol. *Table 6-1* defines the signal names. All signals are either high active or low active signals. The "#" symbol at the end of a signal name indicates that the active, or asserted state occurs when the signal is at a low voltage level. When "#" is not present after the signal name, the signal is asserted when at the high voltage level.

Asserted means that the signal is driven by an active circuit to its true state. Negated means that the signal is driven by an active circuit to its false state. Released means that the signal is not actively driven to any state.

Control signals that may be used for more than one mutually exclusive functions are identified with their function names separated by a colon (e.g. DIOW#:STOP).

Table 6-1 EIDE Interface Signal Name Assignments

Acronym	Description	I/O
CSEL	Cable select	O
CS0#	Chip select 0	O
CS1#	Chip select 1	I
DD[15:0]	Data bus bits 15 to 0	I/O
DASP#	Device active or slave (Device 1) present	*
DA0	Device address bit 0	O
DA1	Device address bit 1	O
DA2	Device address bit 2	O
DMACK#	DMA acknowledge	O
DMARQ	DMA request	I
INTRQ	Interrupt request	I
DIOR#	I/O read	O
HDMARDY#	DMA ready during Ultra DMA data-in bursts	O
HSTROBE	Data strobe during Ultra DMA data-out bursts	O
IORDY	I/O ready	I
DDMARDY#	DMA ready during Ultra DMA data-out bursts	I
DSTROBE	Data strobe during Ultra DMA data-in bursts	I
DIOW#	I/O write	O
STOP	Stop during Ultra DMA data bursts	O
PDIAG#	Passed diagnostics	*
CBLID#	Cable assembly type identifier	*
RESET#	Reset	O

* See section *6.2.2.4* below for details

6.2.2.4 Detailed IDE Signal Description

CS [1:0]# (Chip select)
These are the chip select signals from the host used to select the Command Block registers. When DMACK- is asserted, CS0- and CS1- shall be negated and transfers shall be 16-bits wide.

DA [2:0] (Device address)
This is the 3-bit binary coded address asserted by the host to access a register or data port in the device.

DASP# (Device active, device 1 present)
This is a time-multiplexed signal that indicates that a device is active, or that Device 1 is present.

NOTE: The indication that the device is active may be unsynchronized with the execution of the command.

DD [15:0] (Device data)
This is an 8- or 16-bit bi-directional data interface between the host and the device. The lower 8 bits are used for 8-bit register transfers. Data transfers are 16-bits wide.

DIOR#:HDMARDY#:HSTROBE (Device I/O read:Ultra DMA ready:Ultra DMA data strobe)
DIOR# is the strobe signal asserted by the host to read device registers or the data port.

HDMARDY# is a flow control signal for Ultra DMA data-in bursts. This signal is asserted by the host to indicate to the device that the host is ready to receive Ultra DMA data-in bursts. The host may negate HDMARDY# to pause an Ultra DMA data-in burst.

HSTROBE is the data-out strobe signal from the host for an Ultra DMA data-out burst. Both the rising and falling edge of HSTROBE latch the data from DD(15:0) into the device. The host may stop generating HSTROBE edges to pause an Ultra DMA data-out burst.

DIOW#:STOP (Device I/O write:Stop Ultra DMA burst)
DIOW# is the strobe signal asserted by the host to write device registers or the data port.

DIOW# shall be negated by the host prior to initiation of an Ultra DMA burst. STOP shall be negated by the host before data is transferred in an Ultra DMA burst. Assertion of STOP by the host during an Ultra DMA burst signals the termination of the Ultra DMA burst.

DMACK# (DMA acknowledge)
This signal shall be used by the host in response to DMARQ to initiate DMA transfers.

DMARQ (DMA request)
This signal, used for DMA data transfers between host and device, shall be asserted by the device when the device is ready to transfer data to or from the host. For Multiword DMA transfers, the direction of data transfer is controlled by DIOR# and DIOW#. This signal is used in a handshake manner with DMACK#, i.e., the device shall wait until the host asserts DMACK# before negating DMARQ, and re-asserting DMARQ if there is more data to transfer.

When a DMA operation is enabled, CS0# and CS1# shall not be asserted and transfers shall be 16-bits wide.

INTRQ (Device interrupt)
This signal is used by the selected device to interrupt the host system when interrupt pending is set. When the nIEN bit is cleared to zero and the device is selected, INTRQ shall be enabled through a tri-state buffer. When the nIEN bit is set to one or the device is not selected, the INTRQ signal shall be released.

When asserted, this signal shall be negated by the device within 400 ns of the negation of DIOR# that reads the Status register to clear interrupt pending. When asserted, this signal shall be negated by the device within 400 ns of the negation of DIOW# that writes the Command register to clear interrupt pending.

When the device is selected by writing to the Device/Head register while interrupt pending is set, INTRQ shall be asserted within 400 ns of the negation of DIOW# that writes the Device/Head register. When the device is deselected by writing to the Device/Head register while interrupt pending is set, INTRQ shall be released within 400 ns of the negation of DIOW# that writes the Device/Head register.

For devices implementing the Overlapped feature set, if INTRQ assertion is being disabled using nIEN at the same instant that the device asserts INTRQ, the minimum pulse width shall be at least 40 ns.

IORDY:DDMARDY#:DSTROBE (I/O channel ready:Ultra DMA ready:Ultra DMA data strobe)
This signal is negated to extend the host transfer cycle of any host register access (Read or Write) when the device is not ready to respond to a data transfer request.

If the device requires to extend the host transfer cycle time at PIO modes 3 and above, the device shall utilize IORDY. Hosts that use PIO modes 3 and above shall support IORDY.

DDMARDY# is a flow control signal for Ultra DMA data-out bursts. This signal is asserted by the device to indicate to the host that the device is ready to receive Ultra DMA data-out bursts. The device may negate DDMARDY# to pause an Ultra DMA data-out burst.

DSTROBE is the data-in strobe signal from the device for an Ultra DMA data-in burst. Both the rising and falling edge of DSTROBE latch the data from DD(15:0) into the host. The device may stop generating DSTROBE edges to pause an Ultra DMA data-in burst.

PDIAG#:CBLID# (Passed diagnostics:Cable assembly type identifier)

PDIAG# shall be asserted by Device 1 to indicate to Device 0 that Device 1 has completed diagnostics. The host may sample CBLID- after a power-on or hardware reset in order to detect the presence or absence of an 80-conductor cable assembly by performing the following steps:

a) The host shall wait until the power on or hardware reset sequence is complete for all devices on the cable;

b) If Device 1 is present, the host should issue IDENTIFY DEVICE or IDENTIFY PACKET DEVICE and use the returned data to determine that Device 1 is compliant with ATA-3 or subsequent standards. Any device compliant with ATA-3 or subsequent standards releases PDIAG# no later than after the first command following a power on or hardware reset sequence.

NOTE: Older devices not in compliance with this standard or ATA-3 may continue to assert this signal providing a false indication of the cable type. Issuing IDENTIFY DEVICE or IDENTIFY PACKET DEVICE not only provides the host with the information required to verify that the devices are compliant with these standards, but also provides a command resulting in the release of this signal.

If the host detects that CBLID# is connected to ground, an 80-conductor cable assembly is installed in the system. If the host detects that this signal is not connected to ground, an 80-conductor cable assembly is not installed in the system.

Hosts may also use IDENTIFY DEVICE or IDENTIFY PACKET DEVICE information to determine the cable type. Hosts which use this device reporting method shall add a 0.047 microfarad capacitor from PDIAG#:CBLID# to ground. The tolerance of this capacitor should be 20% or less.

Devices supporting Ultra DMA modes greater that two shall support this cable determination method. Devices shall support this method by using CBLID# to test for a connection to this host capacitor.

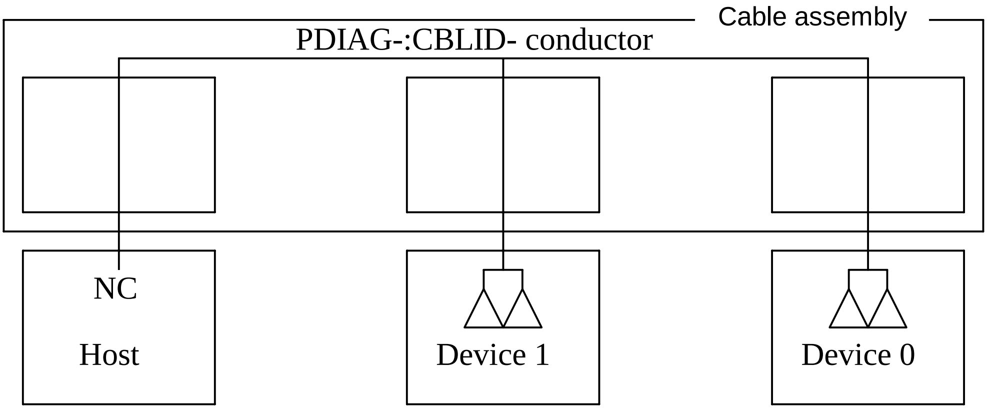


Figure 6-3 Example of legacy host with 40-conductor cable

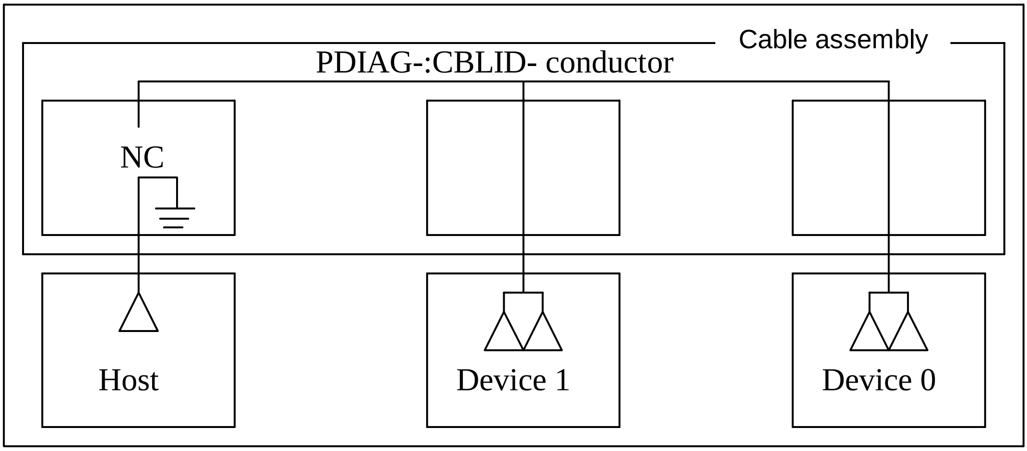


Figure 6-4 Example of host detecting 80-conductor cable

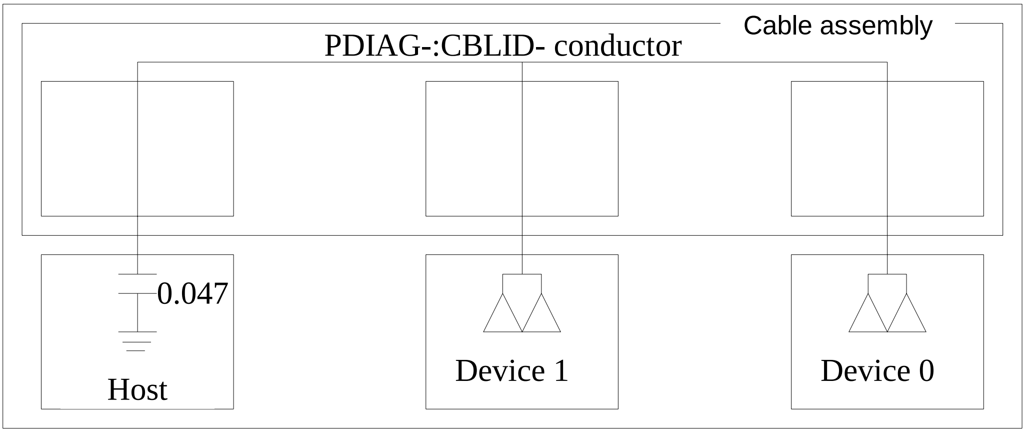


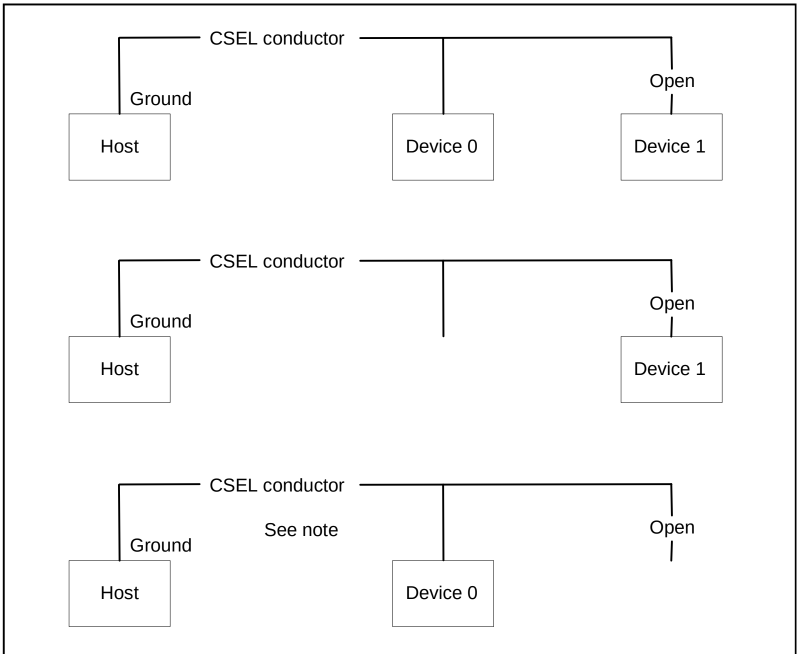
Figure 6-5 Example of device detecting 80-conductor cable

RESET- (Hardware reset)
This signal, referred to as hardware reset, shall be used by the host to reset the device.

CSEL (Cable select)
The device is configured as either Device 0 or Device 1 depending upon the value of CSEL:

- If CSEL is negated, the device address is 0;
- If CSEL is asserted, the device address is 1.

NOTE: Special cabling may be used to selectively ground CSEL, e.g., CSEL of Device 0 is connected to the CSEL conductor in the cable, and is grounded, thus allowing the device to recognize itself as Device 0. CSEL of Device 1 is not connected to CSEL because the conductor is removed, thus the device recognizes itself as Device 1. It should be recognized that if a single device is configured at the end of the cable using CSEL, a device 1 only configuration results.



NOTE – If a single device configuration is implemented with the device in the middle, a cable stub results that may cause degradation of signals.

Figure 6-6 Cable select example

NOTE: For designated cable assemblies (including all 80-conductor cable assemblies): these assemblies are constructed so that CSEL is connected from the host connector to the connector at the opposite end of the cable from the host. Therefore, Device 0 shall be at the opposite end of the cable from the host. It should be recognized that if a single device is configured at the connector that is not at the end of the cable, a Device 1 only configuration results.

6.3 Universal Serial Bus (USB) Controller

The USB controller is designed to be a USB host (or root hub) able to support two standard USB ports at speeds of 12Mb/sec and 1.5Mb/sec in compliance with the Universal Serial Bus (USB) 1.1 specification. The USB controller also has a built-in DMA controller which helps to off-load work from the central processor. The USB physical layer control is integrated on-chip. External support components include the USB5V power supply and a clamping circuit to provide 5V input tolerance.

Both USB ports share a single USB controller for a combined bandwidth of 12 Mb/sec.

Common USB devices include mouse, keyboard, joystick, scanner, printer, digital camera, and audio (speakers).

6.3.1 Feature List

- Compliance with the Universal Serial Bus (USB) 1.1 specification
- PCI Interface that supports USB control
- 12Mbits/sec maximum data rate through the USB controller
- Supports the following types of USB transfers:
    - Control
    - Isochronous
    - Interrupt
    - Bulk

6.3.2 Functional Description

6.3.2.1 Block Diagram

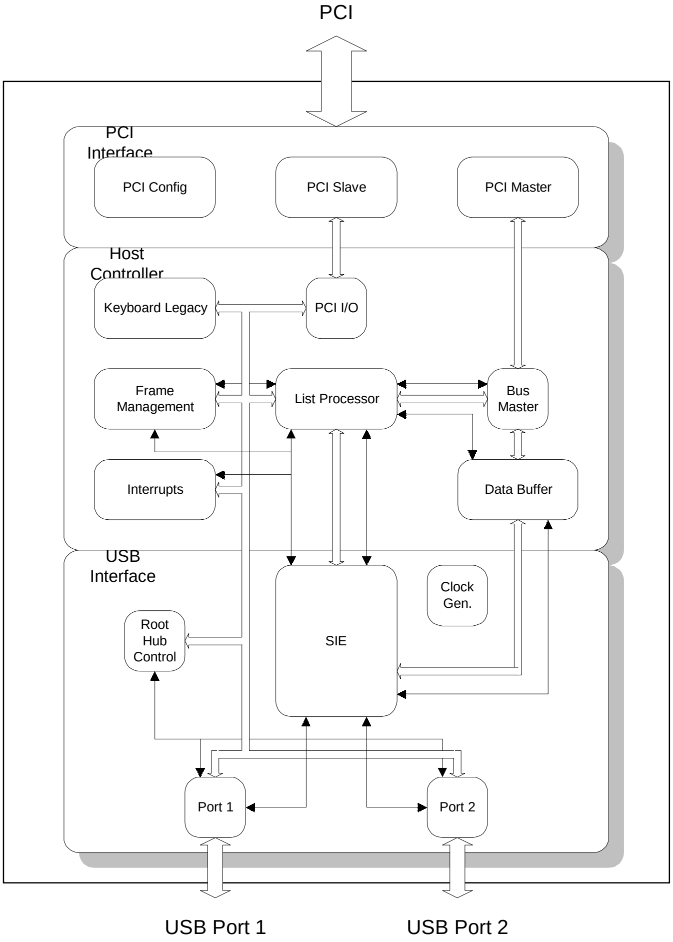


Figure 6-7 USB Interface Block Diagram

6.3.2.2 Theory of Operation

The Host Controller has four USB states visible to the Host Controller Driver via the Operational Registers: USBOPERATIONAL, USBRESET, USBSUSPEND, and USBRESUME. These states define the Host Controller responsibilities relating to USB signaling and bus states.

Start-of-frame Tokens are used to synchronize all USB devices to a nominal 1 ms bus interval.

The USB states are reflected in the **HostControllerFunctionalState** field of the *HcControl* register. The Host Controller Driver is permitted to perform only the USB state transitions shown in *Figure 6-8*. The Host Controller may only perform a single state transition. During a remote wakeup event, the Host Controller may transition from USBSUSPEND to USBRESUME.

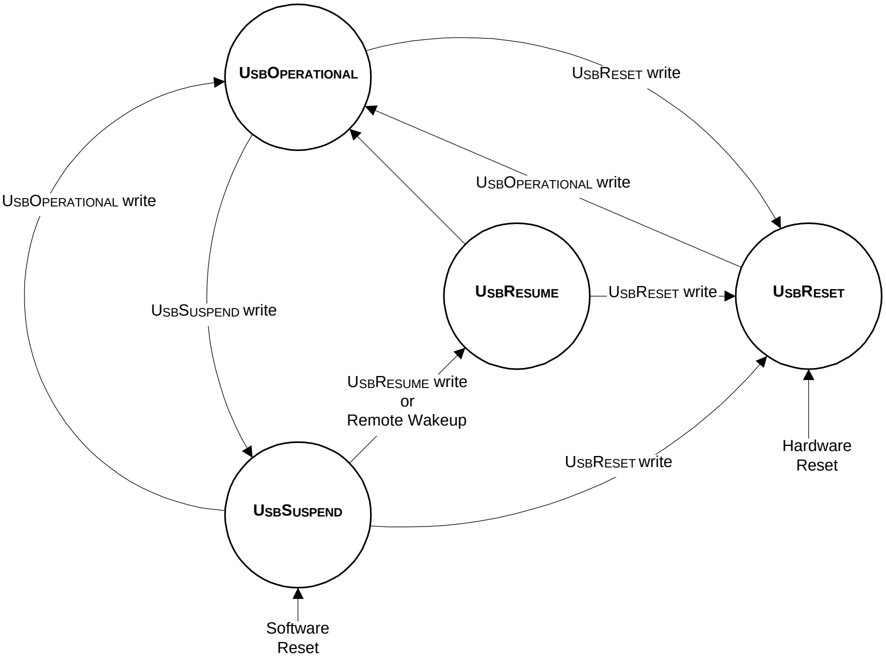


Figure 6-8 USB States

USBOPERATIONAL

When in the USBOPERATIONAL state, the Host Controller may process lists and will generate Start-of-frame Tokens. The USBOPERATIONAL state may be entered from the USBRESUME or USBRESET states. It may be exited to the USBRESET or USBSUSPEND states.

When transitioning from USBRESET or USBRESUME to USBOPERATIONAL, the Host Controller is responsible for terminating the USB reset or resume signaling as defined in the USB Specification prior to sending a token.

A transition to the USBOPERATIONAL state affects the frame management registers of the Host Controller. Simultaneously with the Host Controller's state transition to USBOPERATIONAL, the **FrameRemaining** field of the *HcFmRemaining* register is loaded with the value of the **FrameInterval** field in the *HcFmInterval* register. There is no Start-of-frame Token sent at this initial load of the **FrameRemaining** field. The first Start-of-frame Token sent after entering the USBOPERATIONAL state is sent following next frame boundary when **FrameRemaining** transitions from 0 to **FrameInterval**. The **FrameNumber** field of the *HcFmNumber* register is incremented on a state transition to USBOPERATIONAL.

USBRESET

When in the USBRESET state, the Host Controller forces reset signaling on the bus. The Host Controller's list processing and Start-of-frame Token generation are disabled while in USBRESET. In addition, the **FrameNumber** field of the *HcFmNumber* register does not increment while the Host Controller is in the USBRESET state. The USBRESET state can be entered from any state at any time. The Host Controller defaults to the USBRESET state following a hardware reset. The Host Controller Driver is responsible for satisfying USB Reset signaling timing defined by the USB Specification.

USBSUSPEND

The USBSUSPEND state defines the USB Suspend state. The Host Controller's list processing and Start-of-frame Token generation are disabled. However, the Host Controller's remote wakeup logic must monitor USB wakeup activity. The **FrameNumber** field of the *HcFmNumber* register does not increment while the Host Controller is in the USBSUSPEND state.

USBSUSPEND is entered following a software reset or from the USBOPERATIONAL state on command from the Host Controller Driver. While in USBSUSPEND, the Host Controller may force a transition to the USBRESUME state due to a remote wakeup condition. This transition may conflict with the Host Controller Driver initiating a transition to the USBRESET state. If this situation occurs, the Host Controller Driver initiated transition to USBRESET has priority. The Host Controller Driver must wait 5 ms after transitioning to USBSUSPEND before transitioning to the USBRESUME state. Likewise, the Root Hub must wait 5 ms after the Host Controller enters USBSUSPEND before generating a local wakeup event and forcing a transition to USBRESUME. Following a software reset, the Host Controller Driver may cause a transition to USBOPERATIONAL if the transition occurs no more than 1 ms from the transition into USBSUSPEND. If the 1 ms period is violated, it is possible that devices on the bus will go into Suspend.

USBRESUME

When in the USBRESUME state, the Host Controller forces resume signaling on the bus. While in USBRESUME, the Root Hub is responsible for propagating the USB Resume signal to downstream ports as specified in the USB Specification. The Host Controller's list processing and Start-of-frame Token generation are disabled while in USBRESUME. In addition, the **FrameNumber** field of the *HcFmNumber* register does not increment while the Host Controller is in the USBRESUME state.

USBRESUME is only entered from USBSUSPEND. The transition to USBRESUME can be initiated by the Host Controller Driver or by a USB remote wakeup signaled by the Root Hub. The Host Controller is responsible for resolving state transition conflicts between the hardware wakeup and Host Controller Driver initiated state transitions. Legal state transitions from USBRESUME are to USBRESET and to USBOPERATIONAL.

The Host Controller Driver is responsible for USB Resume signal timing as defined by the USB Specification.

6.3.2.3 Packet Control

The sections below describe the packet types and formats generated by the Serial Interface Engine (SIE).

Sync Pattern
The first byte of every packet is a synchronization pattern that allows the data receiver to synchronize to the transmitters data rate prior to any meaningful data. The pattern 80h translates into an NRZI encoding of 54h on the bus.

End of Packet (EOP)
Every packet is terminated by an end of packet (EOP) consisting of 2 transmitted single-ended zeros (SE0) followed by a driven idle state for another bit time.

Packet Identifier (PID)
Each packet contains a packet identifier (PID) declaring the packet function. The PID is transmitted as a 4-bit encoding followed by the bit-wise inversion of the PID as an error check. If the PID check fails, the packet is ignored since without a valid PID the packet cannot be identified. Table *Table 6-2* below shows the PID encoding and definitions for each packet type.

Table 6-2 PID Coding

Packet Type	Function	PID[3:0]	Description
Token	OUT	0001	Host to Device transaction request
	IN	1001	Device to Host transaction request
	SOF	0101	Start of Frame timing mark
	SETUP	1101	Host to Device Control transaction setup
Data	DATA0	0011	Even data packet
	DATA1	1011	Odd data packet
Handshake	ACK	0010	Receiver accepts the data packet
	NAK	1010	Receiver rejects the data packet or transmitter cannot send data packet
	STALL	1110	Endpoint is stalled.
Special	PRE	1100	Preamble for low speed driver enable in hubs

Token Packet
There are four types of token packets described below: IN, OUT, SETUP, Start of Frame (SOF). IN and OUT specify data packet direction in the data phase of the transaction. SETUP has the same meaning as an OUT token to the SIE but has special meaning to the device decoding it. An SOF is used to synchronize all USB devices to a nominal 1 ms bus interval.

An IN, OUT, or SETUP token packet contains an 11-bit data field containing the device and endpoint address. For a SOF token, this same 11-bit field contains the lower 11 bits of the **FrameNumber** field in the *HcFmNumber* register. The packet organization is shown in *Table 6-3* and *Table 6-4*. The data field is followed by a 5-bit CRC. The root hub is the only device that can send a token packet so it is not necessary to decode a token packet. All fields are serially transmitted LSB first (except the CRC).

Table 6-3 Standard Token Packet Format

Packet Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Sync Pattern[7:0]							
1	~PID[3:0]				PID[3:0]			
2	Endpoint[0]	Address[7:0]						
3	~CRC[0:4]					Endpoint[3:1]		
4	High-Z					Idle	SE0	

Table 6-4 SOF Token Packet Format

Packet Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Sync Pattern[7:0]							
1	~PID[3:0]				PID[3:0]			
2	**FrameNumber**[7:0]							
3	~CRC[0:4]					**FrameNumber**[10:8]		
4	High-Z					Idle	SE0	

Data Packet

The data packet contains the actual data transfer between the host and endpoint specified in the token packet. DATA0 and DATA1 identify the packet as a data packet. The 0/1 tag on the PID provides a mechanism for data synchronization from one data packet to the next to the same endpoint.

The data packet includes the 4 bit PID, 4 bit inverted PID, 0-1023 data bytes, and a 16 bit CRC. All fields are serially transmitted LSB first (except CRC). The number of data bytes for an OUT is determined by the Host Controller. The IN data byte count is tracked by the Data Buffer Engine. The number of data bytes is irrelevant to the operation of the SIE.

Table 6-5 Data Packet Format

Packet Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Sync Pattern[7:0]							
1	~PID[3:0]				PID[3:0]			
2	Data0[7:0]							
...	...							
N+2	DataN[7:0]							
N+3	~CRC[8:15]							
N+4	~CRC[0:7]							
N+5	High-Z					Idle	SE0	

Handshake Packet

The handshake packet is used to close the bus transaction and report completion status. An ACK handshake concludes a successful transaction. The STALL handshake is returned when the device is unable to handle data due to an internal error. NAK is used when the endpoint has no data to send or does not want any data. The host can only send an ACK handshake to the device, but the device can return any handshake.

The handshake packet includes only the 4 bit PID and 4 bit inverted PID.

Table 6-6 Handshake Packet Format

Packet Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Sync Pattern[7:0]							
1	~PID[3:0]				PID[3:0]			
2	High-Z					Idle	SE0	

Preamble Packet

The preamble packet is used to inform downstream hubs that a low speed packet is coming. This allows the low speed drivers to be enabled by all ports connected to low speed devices. This is repeated for all packets within the transaction and not just preceding the token packet. This procedure applies only to packets from the host and not packets received from the device. The preamble packet, like the handshake packet, only includes the 4 bit PID and 4 bit inverted PID, except that it is not terminated by an EOP. Instead, the bus is driven idle for 4 Full Speed clocks, then the data rate changes to 1.5MHz and the sync pattern of the Low Speed packet is begun immediately. The Low Speed packets are otherwise as described in the previous sections.

Table 6-7 Preamble Packet Format

Packet Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Sync Pattern[7:0]							
1	~PID[3:0]				PID[3:0]			
2	Change to LS data Rate and begin LS token, data, or handshake packet as defined in the above sections..				Idle			

6.3.3 Performance

The USB controller can operate at 12Mbits/sec maximum data rate. This value is based on a 48MHz USB reference clock. This data rate must be shared between the two USB ports, so the data rate one each USB port will be reduced if both ports are operating at the same time.

6.4 Low-Pin Count (LPC) Interface Controller

6.4.1 Overview

The LPC interface controller implements the Low Pin Count (LPC) interface, as described in *Intel Low Pin Count (LPC) Interface Specification, Rev. 1.0*, and the serial interrupt interface, described in *Serialized IRQ Support for PCI Systems, Rev. 6.0*. The LPC bus is primarily used to support so-called “Super I/O” chips which commonly include a number of standard interfaces, such as: floppy disk controller, parallel port, keyboard/mouse controller, additional serial ports, and additional general purpose I/Os. The Xilleon 220 LPC controller does not support DMA or bus mastering.

6.4.2 Feature List

- Read and write LPC PCI configuration registers in response to PCI configuration cycles addressed to the LPC function.
- Responds to regular PCI data transfer cycles addressed to certain legacy IO spaces and relays requests in these ranges to the LPC bus.
- Responds to regular PCI data transfer cycles addressed to certain memory spaces and relays requests in these ranges to the LPC bus.
- Acts as a host controller for a serial interrupt request bus.

6.4.3 Functional Description

6.4.3.1 Block Diagram

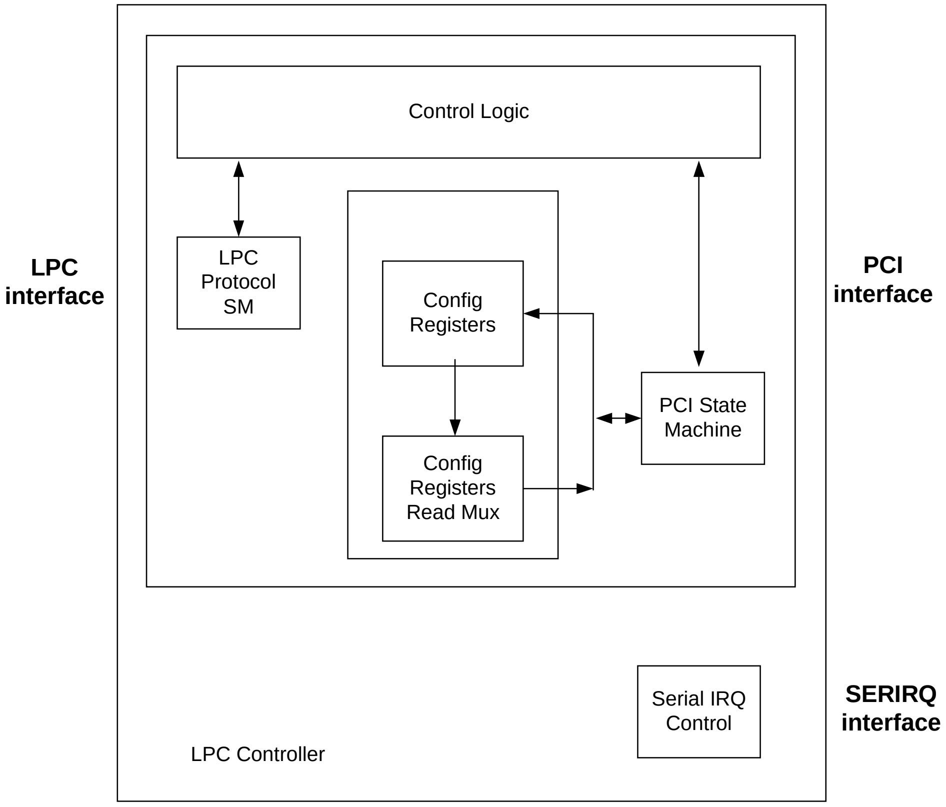


Figure 6-9 LPC Controller Block Diagram

6.4.3.2 Serial Interrupt Control Detail

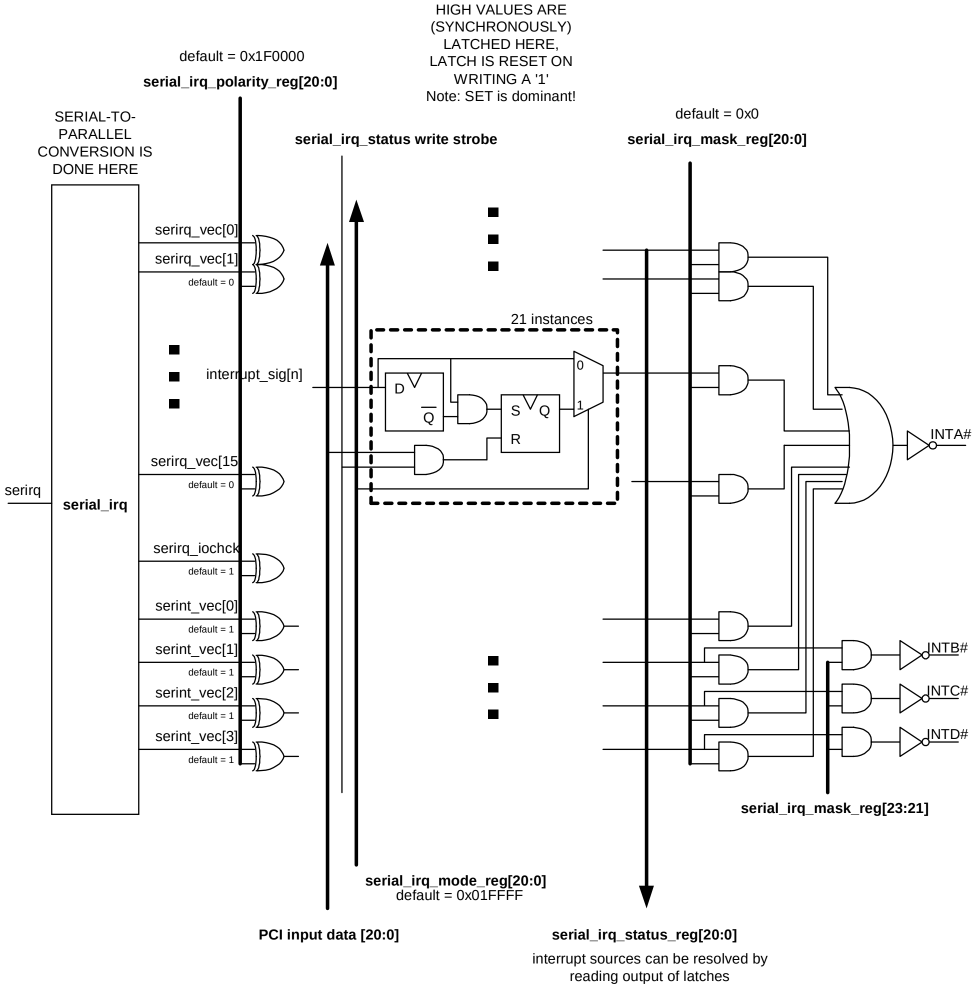


Figure 6-10 Serial Interrupt Control Detail

6.4.3.3 Theory of Operation

The I/O ranges listed below are enabled by setting the appropriate configuration registers (e.g. IO_PORT_DECODE_EN_n, WIDE_GENERIC_BASE_ADDR).

Table 6-8 Configuration I/O Ranges

Device	# Ranges	I/O Ranges
Parallel Port	1 of 3 ranges	378h-37Fh (+ 778h-77Fh for ECP) 278h-27Fh (+ 678h-67Fh for ECP) See note 1. 3BCh-3BFh (+ 7BCh-7BFh for ECP)
Serial Ports	2 of 8 ranges	3F8h-3FFh, 2F8h-2FFh, 220h-227h, 228h-22Fh, 238h-23Fh, 2E8h-2EFh, 338h-33Fh, 3E8h-3EFh
Audio	1 of 4 ranges	SoundBlaster™ compatible: 220h-233h, 240-253h, 260h-273h, 280h-293h
MIDI	1 of 4 ranges	300h-301h, 310h-311h, 320h-321h, 330h-331h
MSS	1 of 4 ranges	530h-537h, 604h-60Bh, E80h-E87, F40h-F47h
FDC	1 of 2 ranges	3F0h-3F7h or 370h-377h
Game Ports	2 1 -byte ranges	Each mapped to any one byte in the 200h-20Fh range.
Wide Generic	16-bit base address register. 512 bytes wide	Can be mapped anywhere in lower 64kB. AC'97 and other configuration registers are expected to be mapped to this range. It is wide enough to allow many unforeseen devices to be supported.
KBC	60h and 64h	
ACPI Micro-Controller	62h and 66h	
Ad-Lib	388h - 389h	
SuperI/O Configuration	2Eh - 2Fh	
Alternate SuperI/O Configuration	4E - 4Fh	

Note [1]: The LPC controller includes configuration registers that allow the selective enable/disable of each of the PCI decode ranges listed above, the enable/disable of the entire LPC controller block, and the control of serial IRQ interface features.

The LPC controller block is mapped at PCI function number 3.

6.4.3.4 Data Flow

The LPC controller block is part of the Peripheral Control Unit (PCU). It resides on an internal PCI bus, and all LPC signals (LFRAME, LAD) are routed through an I/O pin mux before connecting to the external chip pins.

6.5 AC-Link Interface

6.5.1 Overview

Xilleon 220 contains an AC-Link Interface which is used for digital audio and modem data transfer with up to two AC'97 CODECs, as shown in *Figure 6-11,"Xilleon 220 Connection to AC'97 CODEC(s) for Analog Audio Input/Output," on page 21*. Note that when only one AC'97 CODEC is used in a system, the unused ACDATA_IN pin of Xilleon 220 must be tied low. The AC-Link Interface is configured as a PCI function, but the actual audio and modem data streams are internally transferred to the AC-Link Interface via the Stream Interface or via direct connection to the Audio Subsystem.

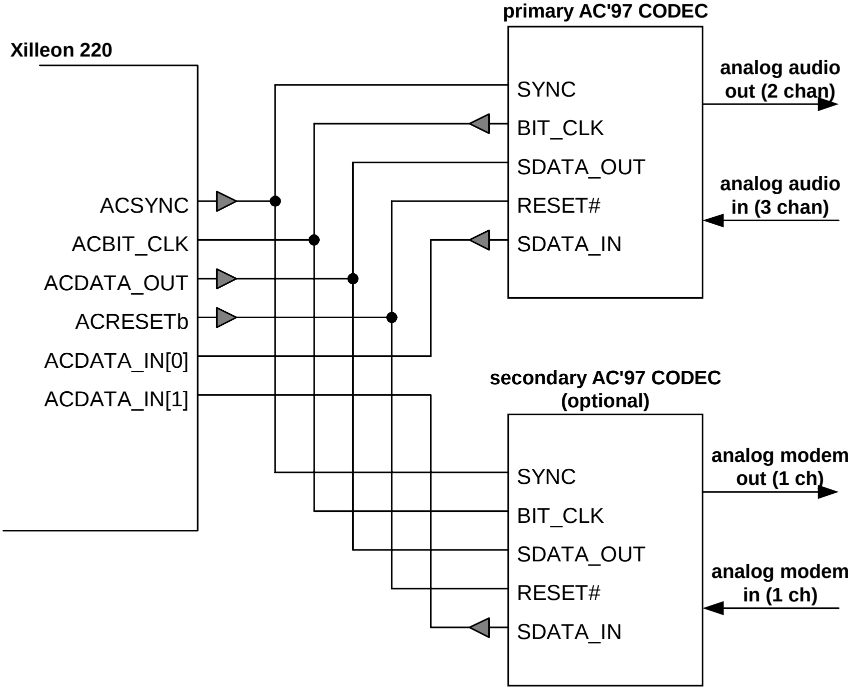


Figure 6-11 Xilleon 220 Connection to AC'97 CODEC(s) for Analog Audio Input/Output

Audio data to be output via the AC-Link Interface is passed directly to the unit from the Audio Subsystem via a direct internal connection. Audio data input to the AC-Link Interface is transferred to local SDRAM memory or PCI space via the Stream Interface. Modem data input and output through the AC-Link Interface is transferred to and from local SDRAM memory or PCI space via the Stream Interface.

NOTE: the AC-Link Interface and Secondary SP-DIF output are conceptually grouped within HDTV as the Digital Audio Input/Output (DAIO) feature set. The DAIO acronym appears frequently in the register specifications relating to these interfaces.

6.5.2 Feature List

The Xilleon 220 AC-Link Interface follows the specifications contained in Intel Audio Codec '97 Specification, Revision 2.1, May 22, 1998.

- Receive 2 channels of 20-bit PCM audio data from the Audio Subsystem at sample rates up to 96 KHz, and output on AC-Link using Slots3 and 4, and optionally using Slots 10 and 11.
- Transfer of digitized 20-bit modem signals from memory via the Stream Interface on AC-Link using Slot 5, at sample rates up to 19.2 KHz.
- Transfer of digitized 20-bit modem signals received on AC-LINK Slot 5 to memory via the Stream Interface at sample rates up to 19.2 KHz.
- Transfer of 20-bit audio data received on AC-LINK Slots 3, 4 and 6 (and optionally Slots 7, 8 and 9) to memory via the Stream Interface at rates up to 48 KHz per slot. An interleaved storage option, whereby two 16-bit audio samples are packed in each 32-bit word, is supported when data is received on only Slots 3 and 4.
- Up to 2 external AC'97 CODECs are supported, with arbitrary mapping of input and output channels to CODECs.

6.5.3 Functional Description

6.5.3.1 Block Diagram

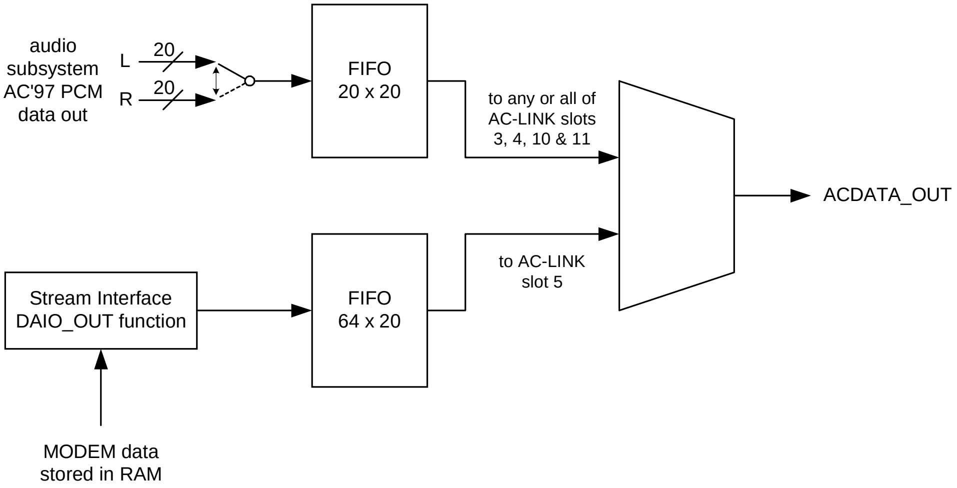

audio subsystem AC'97 PCM data out
L
20
R
20
FIFO 20 x 20
to any or all of AC-LINK slots 3, 4, 10 & 11
ACDATA_OUT
Stream Interface DAIO_OUT function
FIFO 64 x 20
to AC-LINK slot 5
MODEM data stored in RAM

Figure 6-12 Audio and Modem Data Out Through AC-Link Interface

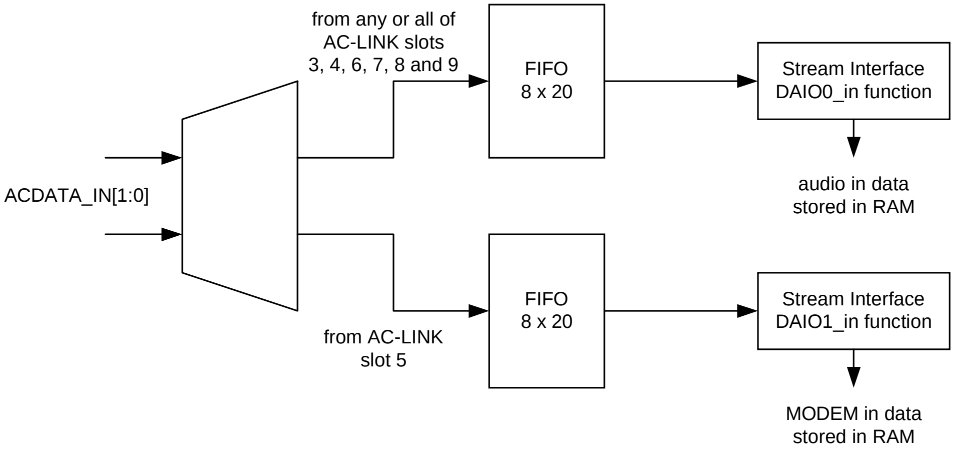


Figure 6-13   Audio and Modem Data In Through AC-Link Interface

6.5.3.2 Theory of Operation

The following basic types of operations can be accomplished via the AC-Link Interface:

- *"PCI Function Configuration" on page 6-23*
- *"AC'97 CODEC Cold Reset and Warm Reset" on page 6-24*
- *"Detection of One or Two AC'97 CODECs" on page 6-26*
- *"Reading and Writing AC'97 CODEC Registers" on page 6-27*
- *"Sending Stereo Audio to an AC'97 CODEC" on page 6-27*
- *"Capturing Input Audio Data From AC'97 CODEC(s)" on page 6-27*
- *"Sending and Receiving Digital Modem Data with an AC'97 CODEC" on page 6-27*

PCI Function Configuration

The Xilleon 220 AC-Link Interface is implemented as two PCI functions, Function 5 and Function 6, as follows:

AC-Link audio input and output and secondary SP-DIF output	PCI Function 5
AC-Link modem data input and output	PCI Function 6

All AC-Link audio input and output is managed at PCI Function 5. All AC-Link modem input and output is managed at PCI Function 6. Note that PCI Function 5 is also used to manage the secondary SP-DIF output.

During the PCI elaboration phase of the software initialization process, each of these two functions must be assigned a 256 byte region in PCI memory space. This is done by setting the AUD_MEM_BASE and MOD_MEM_BASE registers in the PCI configuration spaces for

Functions 5 and 6, respectively. The 256-byte region within PCI memory space defined by the AUD_MEM_BASE register will be referred to as the DAIO_AUDIO space and the 256-byte region within PCI memory space defined by the MOD_MEM_BASE register will be referred to as the DAIO_MODEM space.

During normal operation, all software interaction with the AC-Link Interface is accomplished by reading and writing registers within the DAIO_AUDIO and DAIO_MODEM spaces. The names of all registers within the DAIO_AUDIO space begin with the prefix "AUD", and those within the DAIO_MODEM space begin with the prefix "MOD".

NOTE: the programming interface for the AC'97 Interface audio functions and modem functions are very similar. Wherever possible, generic programming information is presented, with registers represented as:

$_REGISTER_NAME.FIELD_NAME or
$_$_REGISTER_NAME.FIELD_NAME

to indicate

AUD_REGISTER_NAME.FIELD_NAME or
AUD_AUDIO_REGISTER_NAME.FIELD_NAME

in DAIO_AUDIO space for processing audio data, and

MOD_REGISTER_NAME.FIELD_NAME or
MOD_MODEM_REGISTER_NAME.FIELD_NAME

in DAIO_MODEM space for processing modem data.

AC'97 CODEC Cold Reset and Warm Reset

Xilleon 220 includes an AC-Link State Machine (ACLSM) to manage AC-Link Cold Reset and Warm Reset operations.

Cold Reset following PCI Reset

Assertion of the PCI RESET# signal (which occurs at power-on) causes the ACLSM to assert the AC-Link RESET# signal to the AC'97 CODECs, and causes the ACLSM to enter the POWER_DOWN state. After PCI RESET# has been deasserted the following steps would occur:

1. The programmer writes a '1' bit to XXX_AUDIO_COMMAND.AC97_CODEC_RESET_L.
2. The ACLSM enters the COLD_RST state, and remains in this state for 75 PCI clock cycles (at least 1.125 µs).
3. The ACLSM deasserts ACS-Link RESET#, and enters the PCI_WAIT state.
4. The primary AC'97 CODEC starts generating the BIT_CLK clock to Xilleon 220.
5. Xilleon 220 starts generating normal AC-Link SYNC and SDATA_OUT signals.
6. The AC'97 CODECs assert the CODEC Ready bit (Slot 0, Bit 15).

NOTE: All AC'97 CODECs are brought out of the reset state after either function (audio or modem) has had the AC97_CODEC_RESET_L bit deasserted.

Cold Reset or Warm Reset following AC-Link Power Down

Once the ACLSM has reached the PCI_WAIT state, the AC'97 CODECs can only be reset after an AC'97 CODEC AC-Link power down. AC'97 CODEC AC-Link power down is accomplished by setting bit PR4 of the Powerdown Control/Status Register (Index 26h) in the primary AC'97 CODEC (see *"Reading and Writing AC'97 CODEC Registers" on page 6-27*). This will result in cessation of the BIT_CLK clock from the primary CODEC, which in turn will cause the ACLSM to enter the POWER_DOWN state and the clearing of the AUD_AUDIO_COMMAND.AC_LINK_ACTIVE and MOD_MODEM_COMMAND. AC_LINK_ACTIVE bits.

AC'97 Cold Reset (i.e. assertion and subsequent deassertion of the AC-Link RESET# signal) can be accomplished by the programmer writing '0' to bit AUD_AUDIO_COMMAND.AC97_CODEC_RESET_L and to bit MOD_MODEM_COMMAND.AC97_CODEC_RESET_L prior to setting bit PR4 of the Primary AC'97 CODEC, and subsequently writing a '1' to $_$_COMMAND.AC97_CODEC_RESET_L bit after polling $_$_COMMAND.ACLINK_ACTIVE for the value '0'.

Similarly, AC'97 Warm Reset (i.e. assertion and subsequent deassertion of the AC-Link SYNC signal) can be accomplished by the programmer writing '1' to bit $_$_COMMAND.AC97_CODEC_RESET_L prior to setting bit PR4 of the Primary AC'97 CODEC, and subsequently writing a '1' to $_$_COMMAND.AC97_SYNC after polling $_$_COMMAND.ACLINK_ACTIVE for the value '0'. The $_$_COMMAND.AC97_SYNC register bits are implemented as one-time pulses, and do not need to be written with the value '0'.

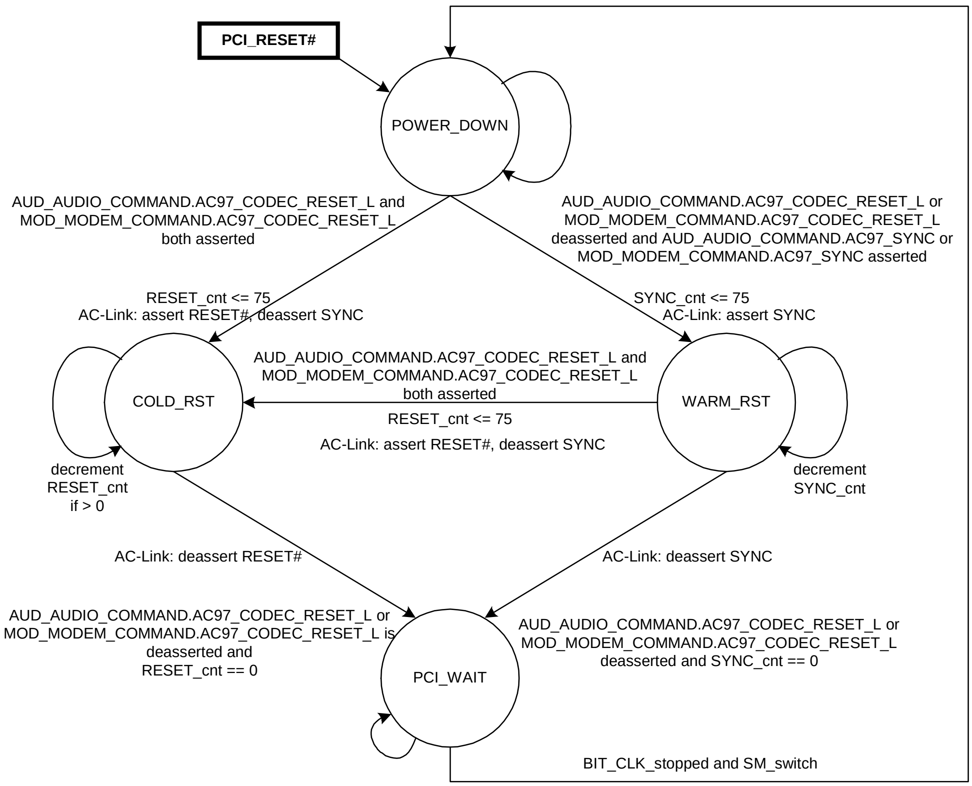

PCI_RESET#
POWER_DOWN
AUD_AUDIO_COMMAND.AC97_CODEC_RESET_L and MOD_MODEM_COMMAND.AC97_CODEC_RESET_L both asserted
AUD_AUDIO_COMMAND.AC97_CODEC_RESET_L or MOD_MODEM_COMMAND.AC97_CODEC_RESET_L deasserted and AUD_AUDIO_COMMAND.AC97_SYNC or MOD_MODEM_COMMAND.AC97_SYNC asserted
RESET_cnt <= 75
AC-Link: assert RESET#, deassert SYNC
SYNC_cnt <= 75
AC-Link: assert SYNC
COLD_RST
AUD_AUDIO_COMMAND.AC97_CODEC_RESET_L and MOD_MODEM_COMMAND.AC97_CODEC_RESET_L both asserted
RESET_cnt <= 75
AC-Link: assert RESET#, deassert SYNC
WARM_RST
decrement RESET_cnt if > 0
decrement SYNC_cnt
AC-Link: deassert RESET#
AC-Link: deassert SYNC
AUD_AUDIO_COMMAND.AC97_CODEC_RESET_L or MOD_MODEM_COMMAND.AC97_CODEC_RESET_L is deasserted and RESET_cnt == 0
AUD_AUDIO_COMMAND.AC97_CODEC_RESET_L or MOD_MODEM_COMMAND.AC97_CODEC_RESET_L deasserted and SYNC_cnt == 0
PCI_WAIT
BIT_CLK_stopped and SM_switch

Figure 6-14 AC-Link State Machine (ACLSM) State Transition Diagram

Detection of One or Two AC'97 CODECs

After completion of either the AC'97 CODEC Cold Reset or Warm Reset operation, properly operating AC'97 CODEC(s) will eventually assert the Codec Ready bit (i.e. (Slot 0, Bit 15). The activity of this bit can be detected by the programmer by reading the bits:

$_INTERRUPT.CODEC0_NOT_READY

$_INTERRUPT.CODEC1_NOT_READY

One potential difficulty in accomplishing automatic detection of AC'97 CODECs is determining the maximum amount of time required for the AC'97 CODEC(s) to become ready after being Reset. The Crystal CS4205 CODEC specifies this time as 62.5 µs Typical, not Maximum. The TI TLV320AIC27 CODEC does not specify any bound on this time. Thus it may be infeasible to automatically detect the presence of installed CODECs.

Reading and Writing AC'97 CODEC Registers

The AC'97 Specification supports up to 64 16-bit registers within the AC'97 CODEC. The AC'97 CODEC registers are read or written via AC-Link. The Xilleon 220 AC-Link Interface provides the programmer access to the AC'97 CODEC registers indirectly via Xilleon 220 registers.

Sending Stereo Audio to an AC'97 CODEC

The Xilleon 220 Audio Subsystem passes digital stereo PCM data to the AC-Link Interface for output to an AC'97 CODEC. This internal data path is 20-bits per audio channel, and can support audio sampling rates up to 96 KHz. Selection of the audio data source is done via the Audio Subsystem programming interface.

The audio data from the Audio Subsystem can be directed to AC-Link Slots 3 & 4 for single rate audio up to 48 KHz, or to AC-Link Slots 3, 4, 10 & 11 for double rate audio up to 96 KHz.

Capturing Input Audio Data From AC'97 CODEC(s)

Up to 6 channels of input audio data can be captured to memory from the AC'97 CODEC(s). There are 3 modes of audio capture, as described in *Table 6-15, “AC-Link Audio Input Capture Modes,” on page 27*. Only data from AC-Link input Slots on ACDATA_IN[1:0] marked with the slot valid tag are captured to memory. If two AC'97 CODECs are installed, they must be configured so that no more than one CODEC sends audio data on a given Slot.

Table 6-15 AC-Link Audio Input Capture Modes

Mode	Audio Format in Memory	Slots Captured to Memory
2 channel	packed 16-bit audio data	3, 4
3 channel	20-bit audio data plus 4-bit Slot ID packed in 32-bit format	3, 4, 6
6 channel	20-bit audio data plus 4-bit Slot ID packed in 32-bit format	3, 4, 6, 7, 8, 9

Sending and Receiving Digital Modem Data with an AC'97 CODEC

A software-based telephone MODEM can be accomplished through the use of an AC'97 MODEM Analog Front End CODEC. Modulated data for transmission to the telephone network is stored in memory buffers, passed to the AC-Link Interface via the Xilleon 220 Stream Interface and delivered to the AC'97 CODEC on AC-Link Slot 5. MODEM data from AC-Link Interface Slot 5 is captured to a memory buffer for software demodulation by the microprocessor.

6.5.3.3 Data Flow

Digital audio and modem data output flow is shown in *Figure 6-12,"Audio and Modem Data Out Through AC-Link Interface," on page 22* above. Digital audio output data is provided to the AC-Link Interface by the audio subsystem, and sequenced on the serial ACDATA_OUT pin. Digital MODEM output data is generated by modulation software, loaded into memory buffers, transferred to the AC-Link Interface by the Stream Interface, and sequenced on the serial ACDATA_OUT pin.

Digital audio and modem data input flow is shown in *Figure 6-13,"Audio and Modem Data In Through AC-Link Interface," on page 23* above. Digital audio input data is extracted from the ACDATA_IN[1:0] pins at Slots 3, 4, 6, 7, 8 and 9 and transferred to memory through the Stream Interface. Digital MODEM input data is extracted from the ACDATA_IN[1:0] pins at Slot 5 and transferred to memory through the Stream Interface for software demodulation by the microprocessor.

6.5.4 External Chip Interfaces

6.5.4.1 AC-Link

AC-Link is a synchronous, time-slotted, serial data protocol that runs at a clock rate of 12.288 MHz. It handles multiple inputs, outputs audio streams, and controls register accesses employing a time division multiplexed (TDM) scheme. The AC-Link architecture divides each audio frame into 12 outgoing and 12 incoming data streams, each with 20-bit sample resolution.

The control and data slots defined by AC '97 2.1 include:

- SDATA_OUT TAG 1 output slot (0)
- SDATA_IN TAG 1 input slot (0)
- Control (CMD ADDR & DATA) write port 2 output slots (1,2)
- Status (STATUS ADDR & DATA) read port 2 input slots (1,2)
- PCM L & R DAC Playback 2 output slots (3,4)
- PCM L & R ADC Record 2 input slots (3,4)
- Optional Modem Line 1 DAC 1 output slot (5)
- Optional Modem Line 1 ADC 1 input slot (5)
- Optional Dedicated Microphone ADC 1 input slot (6)

Xilleon 220 also supports audio input on AC-Link slots 7, 8 & 9.

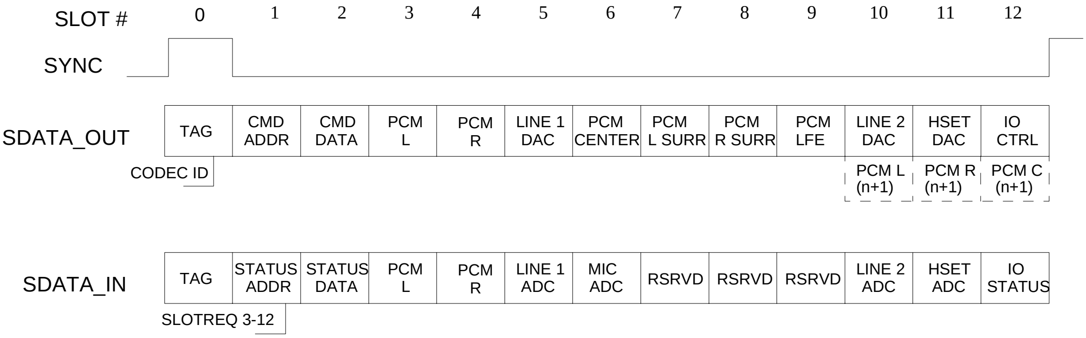


Figure 6-16 AC’97 Standard Bi-directional Audio Frame

The primary AC '97 CODEC drives the BIT_CLK clock to Xilleon 220, which Xilleon 220 then uses to construct audio frames. SYNC, fixed at 48 kHz, is derived by dividing down the serial bit clock (BIT_CLK). BIT_CLK, fixed at 12.288 MHz, provides the necessary clocking granularity to support 12, 20-bit outgoing and incoming time slots. AC-Link serial data is driven on the rising edge of BIT_CLK and sampled on the falling edge of BIT_CLK.

The AC-Link protocol provides for a special 16-bit time slot (Slot 0) wherein each bit conveys a valid tag for its corresponding time slot within the current audio frame. A 1 in a given bit position of slot 0 indicates that the corresponding time slot within the current audio frame has been assigned to a data stream, and contains valid data. If a slot is tagged as invalid, it is the responsibility of the source of the data to stuff all bit positions with 0's during that slot's active time. SYNC remains high for a total duration of 16 BIT_CLKs at the beginning of each audio frame. The portion of the audio frame where SYNC is high is defined as the Tag Phase. The remainder of the audio frame where SYNC is low is defined as the Data Phase.

6.5.4.2 Multiple CODECs and Unused Inputs

The Xilleon 220 AC-Link Interface supports up to two AC'97 CODECs. A single AC-Link serial data output pin, ACDATA_OUT, is provided for connection to all AC'97 CODEC SDATA_OUT input pins. Two AC-Link serial input pins, ACDATA_IN[0] and ACDATA_IN[1], are provided for connection to the SDATA_IN output pins of up to two AC'97 CODECs. Any unused ACDATA_IN inputs must be tied to a logical '0'.

6.5.5 Performance

6.5.5.1 Sample Resolution

The AC-Link Interface supports 20-bit digital audio and MODEM data resolution. Some operating modes discard the 4 least significant bits, and only retain 16 bits of resolution in order to make more efficient use of memory.

6.5.5.2 Sample Rates and Operating Frequency

The AC-Link Interface is designed to operate with an input BIT_CLK clock rate of 12.288 MHz, as required by the AC'97 Specification. AC'97 Variable Rate Audio (VRA) is supported for both input and output. The AC-Link Interface supports any audio output sample rate up to 96 KHz, and any audio input sample rate up to 48 KHz, though output audio sample rates may be restricted by the audio subsystem.

6.5.5.3 Formats

The AC-Link Interface transfers data to and from memory via the Stream Interface. Data is formatted in memory according to the mode (interleaved or not) and whether the flow is in or out. All memory formats are little endian, i.e. where the least significant byte occupies the lowest byte address of a multi-byte word.

Non-interleaved Input Data Format (Audio or Modem)
Mapping of data in the non-interleaved input data format is shown in *Figure 6-17,“Non-interleaved Input Data Format in Memory,” on page 30*.

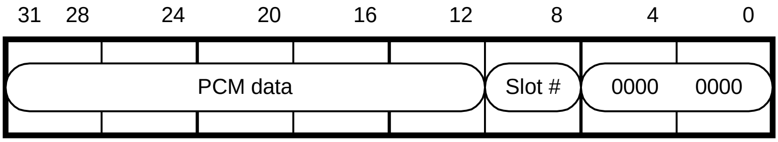


Figure 6-17 Non-interleaved Input Data Format in Memory

Interleaved Input Data Format (Audio only)
Mapping of data in the interleaved input data format is shown in *Figure 6-18,“Interleaved Input Data Format in Memory,” on page 30*.

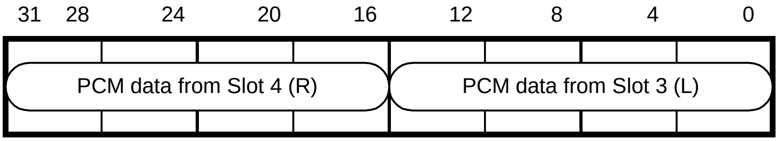


Figure 6-18 Interleaved Input Data Format in Memory

Non-interleaved Output Data Format (Modem only)

Mapping of data in the non-interleaved output data format is shown in *Figure 6-19,"Non-interleaved Output Data Format in Memory," on page 31*.

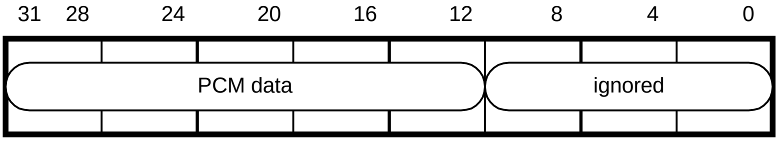


Figure 6-19 Non-interleaved Output Data Format in Memory

6.5.5.4 Assumptions and Restrictions

- When sending digital audio data over AC-Link, slots 3 & 4 must both be enabled. If either slot 10 or slot 11 is enabled, then both must be enabled.
- It is assumed that, when sending digital audio over AC-Link with Variable Rate Audio (VRA), the AC'97 CODEC will control the SLOTREQ bits for all of the enabled audio slots in tandem.
- In order to send GPIO data, the MODEM_PRESENT bit must be set, but this precludes using slots 10 & 11 for audio. Thus, if AC'97 CODEC GPIO is supported, then the audio output rate is restricted to 48 KHz.
- AC'97 CODEC GPIO data can only be sent and received through the register interface. There is no direct path to or from AC'97 GPIO and memory.

6.5.5.5 Dependencies

- When sending output audio data from the Audio Subsystem, both the Audio Subsystem and the AC'97 CODEC must be configured for the same audio sample rate.
- When using two AC'97 CODECs, they must be configured so that no more than one CODEC passes input audio data on a given AC-Link Slot.

This page intentionally left blank.

Chapter 7
Other Peripherals

7.1 Introduction

This chapter describes the following peripheral functions:

- *"General Purpose Timers and Real-Time Clock" on page 7-3*
- *"FlexBus (Flexible Peripheral Bus)" on page 7-7*
- *"Universal Asynchronous Receiver/Transmitters (UARTs)" on page 7-22*
- *"Universal Infrared Receiver/Transmitter (UIRT)" on page 7-28*
- *"Peripheral DMA Unit" on page 7-43*
- *"Hardware Controlled I2C Serial Ports" on page 7-51*
- *"VIP Host Port" on page 7-57*

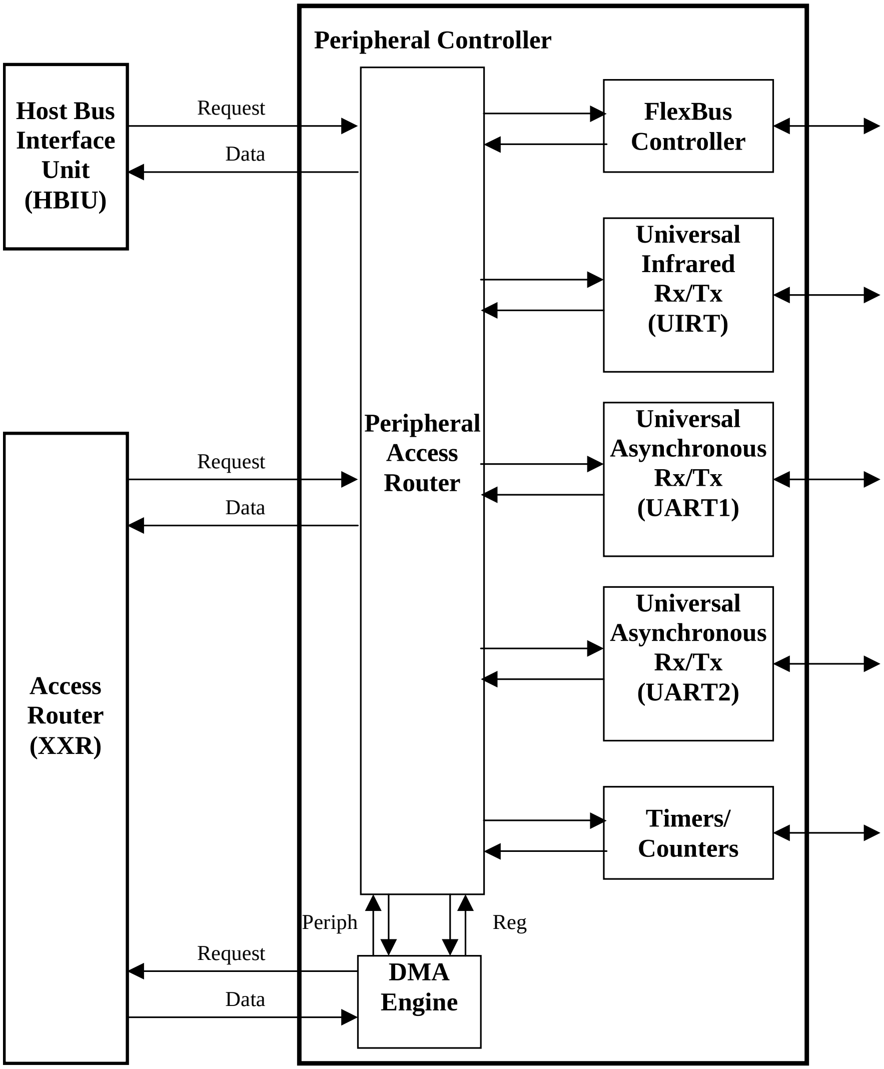


Figure 7-1 Block Diagram of the Peripheral Controller Unit

7.2 General Purpose Timers and Real-Time Clock

The Xilleon 220 timer support consists of two discrete sections: the General Purpose Timers and the Real-Time Clock (RTC). The General Purpose Timers section consists of six 32-bit timers that are clocked with the 27 MHz. system clock. Timers 0-3 each have an output signal and a count enable signal that are routed off-chip through the GP I/O pins. These output signals can be programmed via a register bit to output a single pulse when the counter reaches zero, or a square wave output that may be externally filtered and used for inexpensive audio applications. All six timers are capable of generating an individually maskable interrupt to the internal processor for event timing purposes. All six timers are loadable to any 32-bit value by the processor, and may be read back at any time for status purposes. Besides the count enable function, one-shot or constant count output functions are supported, as well as retriggering capability. The six interrupt signals are OR'd together to produce a single Timer Interrupt signal to the processor. When this signal is received, the processor should read the appropriate Timer Control Register to see which individual interrupt or interrupts are actually valid. After the interrupt is serviced, it is individually cleared with a unique interrupt clear signal, also located in a control register.

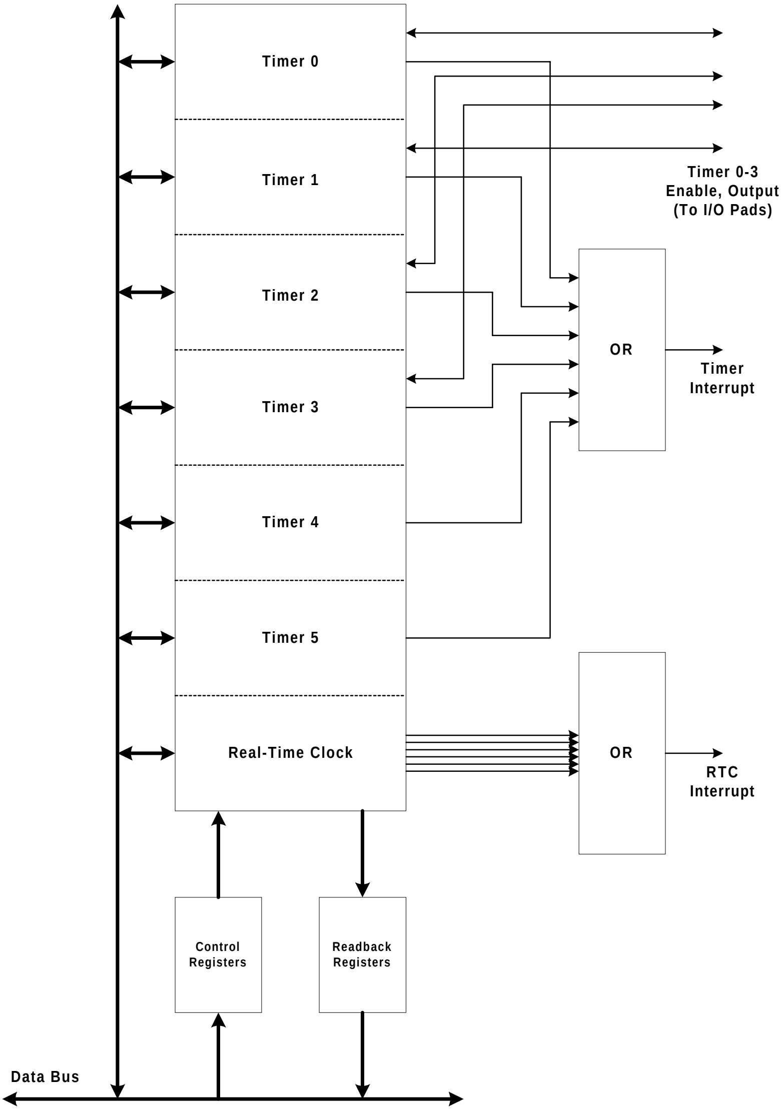


Figure 7-2 General Purpose Timers

The Real-Time Clock (RTC) is a modified timer that is designed expressly for keeping a time-of-day clock. For maximum accuracy, the RTC is clocked via the 27 MHz. system clock. Further accuracy may be maintained by having the processor make periodic adjustments to the RTC based upon the reception of network time base information.

The RTC consists of hardware counters to count seconds, minutes, hours, days, and months. These counters may be set via an RTC register, and may be read back by the processor at any time during normal operation.

There are six programmable event registers that may be used to generate interrupts at desired times in the future. They may store events up to one year in the future. As with the General Purpose Timers, the six interrupt signals are OR’d together to produce a single RTC Interrupt signal to the processor. When this signal is received, the processor should read the appropriate RTC control register to see which individual interrupt or interrupts are actually valid. After the interrupt is serviced, it is individually cleared with a unique interrupt clear signal, also located in a control register.

The RTC does not store any time information during a system power-down. It therefore must be fully reprogrammed during a power interruption.

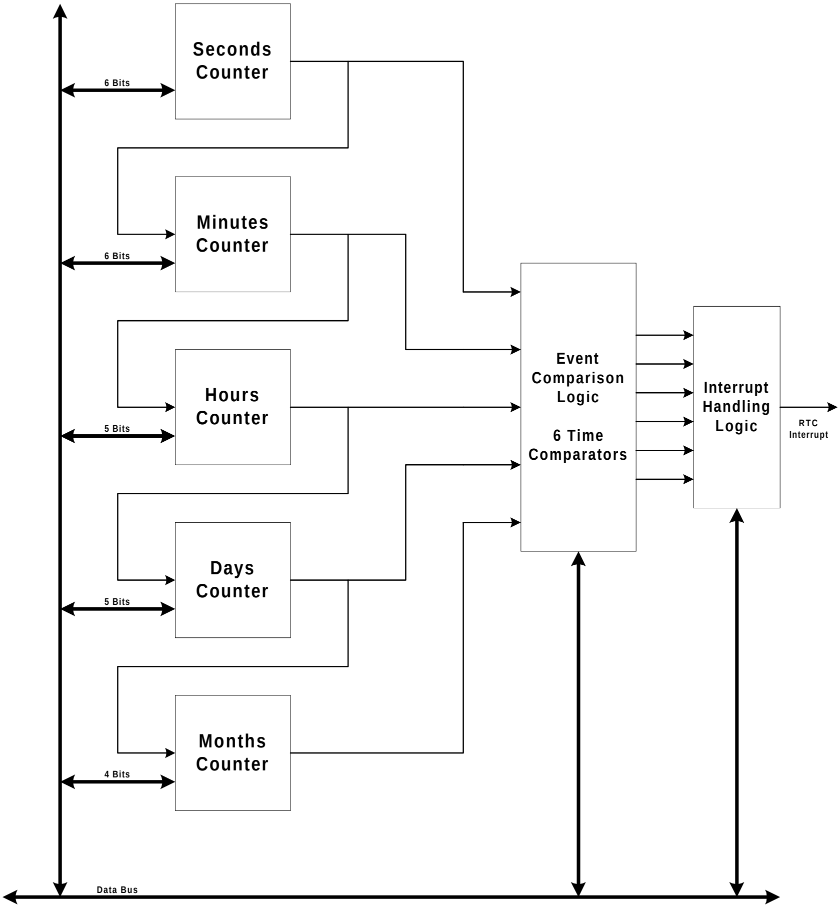


Figure 7-3 Real-Time Clock (RTC)

7.3 FlexBus (Flexible Peripheral Bus)

7.3.1 Overview

The FlexBus operates in one of several configurations, and the configuration may change with each transaction on the bus. Some of the characteristics of the bus that can be varied are:

- Data Bus Width (8- or 16-bit)
- Address Bus Width (up to 32-bit)
- Variable number of Chip Enables, from 0 to 6.
- Timing and protocol of the bus; using the Strobe, Ready, Read, Write Enable, and Chip Enable signals.

The Xilleon 220 device acts as the **Host** of the FlexBus. The duties of the host include:

- Generating the clock for the bus.
- Implementing the arbiter for the ownership of the bus.

The FlexBus can support multiple **agents** on the bus. An agent is a device that can act as a master or a slave on the bus. The FlexBus directly supports one external master through a simple Request/Grant protocol to gain ownership of the bus. Additional external masters may be implemented by having an external PAL perform arbitration among them.

Note that the Xilleon 220 can only act as a master on the FlexBus, never as a slave.

7.3.2 Feature List

- Programmable internal sequencer can emulate the "68000-like" bus as well as other protocols used by popular peripheral devices.
- Supports up to six different protocols/timings to accommodate a variety of devices residing on the common bus.
- Powers-up to a state that allows the host CPU to boot from a device residing on the FlexBus.
- Supports 8-bit or 16-bit data devices.
- Supports up to 32-bits of address.
- Bus operates up to 66 MHz clock frequency.
- Xilleon 220 can relinquish ownership of the bus in order to allow another external master device to perform transactions to another external target device.

7.3.3 Bus Overview

In order to minimize pin usage, the FlexBus employs a multiplexed address and data bus. This bus is 16-bits wide, and is multiplexed among three sources: Address[31:16], Address[15:0], and Data[15:0]. For the external latching of the address bus, it is assumed that the board designer will use a ‘573 type of device, a transparent latch that is transparent while the enable signal is a ‘1’. Specifically, it is expected that there is no free-running clock to the latch; the loading of the latch is controlled solely by the FALEn signal.

The FlexBus Controller manages when the external address latches need to be updated. The address is managed in two different pieces: bits [31:16] and bits [15:0], and each can be updated independently by the FlexBus Controller. Via the ABUSLATCH field of the FBC_APERn_CNTL register, the FlexBus Controller knows the range of address bits required by each aperture, and it also keeps a shadow copy of the contents of the external latches so that it can decide whether to update an external latch or not. For example, if the aperture uses address bits from M to N, and the current transaction request indicate these bits to be different from the shadow copy, then that address latch must be updated. This management of address latches helps to reduce the time spent using the pins to send out address information. One bus clock cycle is needed to update an address latch.

A bus transaction begins when the address is stable in the latches. The updating of address latches is not viewed as part of a bus transaction. It is a task that may or may not occur **between** bus transactions, depending on how the address is changing from one transaction to the next. See *Figure 7-4,“Overview of FlexBus Operation,” on page 8*.

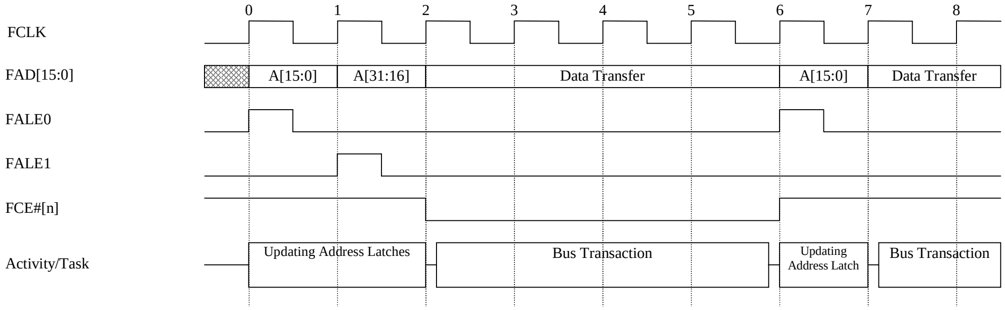


Figure 7-4 Overview of FlexBus Operation

Generally, the Chip Enable outputs from the Xilleon 220 will control individual devices on the FlexBus such that each device is attached to a specific Chip Enable pin. This enables up to six external devices to be attached to the FlexBus.

The FlexBus Controller manages the driving of the bidirectional FAD[] bus to ensure that there is always one clock of no driving when turning around the direction of the bus. For example, if the first bus transaction shown in *Figure 7-4,“Overview of FlexBus Operation,” on page 8* were a read cycle, the FlexBus Controller would insert an additional cycle after clock edge 6, which would have the bus tri-state in order to turn around the FAD bus from an input to an output. Subsequently, the address-latch updating would occur during the following clock period.

The FRDY signal of the FlexBus is driven by the external device, and can be sampled by the FlexBus Controller. The FlexBus Controller generally would use this input to insert wait-states into a bus transaction. Some transactions on the FlexBus do not use the FRDY signal to control the length of the transaction on the bus, i.e., they take a fixed number of clocks, and the slave device has no mechanism to throttle the action of the master.

If there are several devices on the FlexBus which support the use of FRDY, then the FRDY signal must be an active-low open-collector (or open-drain) signal with an external pull-up. When an agent on the FlexBus determines that it is the slave device for a given transaction, it can drive the FRDY signal, otherwise it is not driven by any device, and it is pulled-high by the pull-up resistor.

If only one FlexBus device has a Ready output, then it can be either active-high or active-low.

7.3.3.1 Data Bus Steering

The FlexBus can directly support 8-bit devices because data is always transferred on **bits 7:0** of the data bus. The FlexBus Controller steers the data to/from the FlexBus to accommodate this case.

7.3.3.2 Bus Request/Grant Protocol

The FlexBus Controller supports external masters on the FlexBus using a simple request/grant protocol, similar to the HOLD/HLDA protocol that Intel defined for their microprocessor buses. The FlexBus Controller provides an arbiter that receives FREQ# as an input and asserts FGNT# as an output. The FlexBus Controller tri-states the FlexBus signals before asserting FGNT#. While FGNT# and FREQ# are asserted, an external master may drive the FlexBus for transactions.

Figure 7-5,“Master-initiated Surrender of Bus Ownership,” on page 10 shows an example of the arbitration protocol. An external device asserts FREQ# when it wishes to gain ownership of the bus, and the arbiter responds by asserting FGNT#. The arbiter may take an indefinite (but finite) time to respond, depending upon the activity demands of the internal devices. As long as the external master holds FREQ# asserted, it owns the bus. It is the responsibility of the external master to eventually surrender ownership by deasserting FREQ# and tri-stating the bus at the same time. In response to FREQ# being deasserted, the arbiter will deassert FGNT#, and the host will again own the bus and drive the tri-state signals.

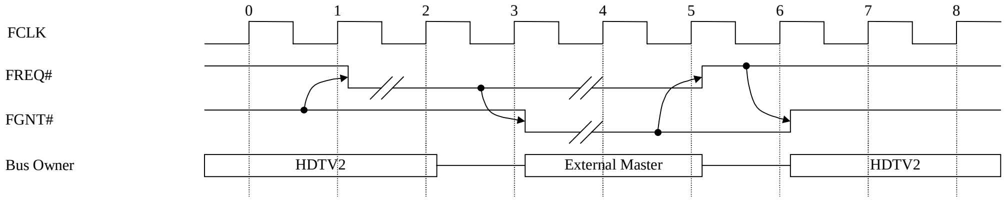


Figure 7-5 Master-initiated Surrender of Bus Ownership

The signals that must be tri-stated by the relinquishing master are:
FAD[15:0], FBE[1:0], FSTB#, FRD#, FWE#, and FCE#[5:0].

It may be necessary to add an external pull-up resistor on certain signals to keep them from floating to an intermediate state while no device is actively driving them, especially FSTB#, FRD#, FWE#, and FCE#[].

7.3.4 Functional Description

7.3.4.1 Block Diagram

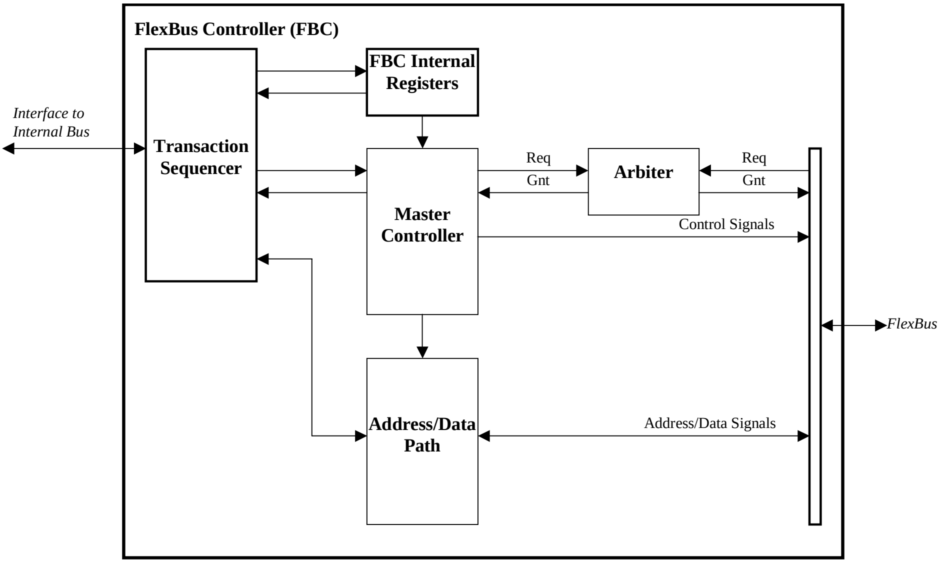


Figure 7-6 Block Diagram of the FlexBus Controller

When a request arrives on the internal bus interface, it enters into the *Transaction Sequencer*. This block analyzes the Byte Enables for the request, and generates the required number of transactions to the FlexBus Master Controller, based upon the programmed size of the FlexBus's data bus (DBUSWIDTH field of the FBC_APERn_CNTL register). For example, a request with all four Byte Enables asserted will cause four (4) transactions to an 8-bit FlexBus device, and two (2) transactions to a 16-bit device.

The *Master Controller* is a microprogrammed engine that can generate a wide range of protocols on the FlexBus. The Master Controller requests ownership of the FlexBus by communicating with the *Arbiter*. Upon gaining ownership of the bus, the Master Controller initiates a master transaction to the FlexBus, using the protocol for the specific aperture that the request "hits" in. The Master Controller performs the address latch management as well as the timing and protocol generation for each aperture.

The *Address/Data Path* logic contains the multiplexing logic to drive the correct portion of the address bus onto the output interface pins.

When a write request to an internal register arrives at the internal bus interface, the FlexBus Controller blocks that request until there are no active memory requests in the entire FlexBus Controller pipeline. This is to ensure that a register change does not affect memory transactions that are in-flight in the FlexBus Controller pipeline.

When a request to the FlexBus Controller memory space arrives at the internal bus interface, the FlexBus Controller blocks that request until there are no active register write requests in the entire FlexBus Controller pipeline. This is to ensure that all pending register changes have been "retired" before the next memory transaction is initiated.

7.3.4.2 Theory of Operation

The FlexBus Controller implements six device apertures. Each aperture provides an address range to control an external chip select signal. When an FlexBus Controller request is received from the internal bus, the requested address is compared against each FlexBus Controller aperture. The matching aperture determines the control parameters applied to the requested peripheral bus transaction. Each aperture provides programmable selection of these control parameters:

- Data bus width of this device.
- Address bus bit usage.
- Transaction Protocol type and parameters.
- Wait state timing.

Data bus width for FlexBus devices can be 8 or 16 bits. The FlexBus Controller will automatically generate multiple cycles to transfer 16 bits on smaller buses. The transaction sequencer follows a **little-endian** rule to generate the sub-cycles, namely:

If the data arriving from the host is in the form of AABBCCDD (bits 31 to 0), then:

- If the device data bus is 16-bits, then
  the first cycle is to address 0, with data CCDD
  the second cycle is to address 2, with data AABB
- If the device data bus is 8-bits, then
  the first cycle is to address 0, with data DD
  the second cycle is to address 1, with data CC
  the third cycle to address 2, with data BB
  the fourth cycle to address 3, with data AA

Each aperture specifies a transaction protocol type for all devices addressed through that aperture. The transaction protocol type characterizes the details of interacting with a particular device. For example, Read Strobe timing, Write Strobe timing, and the number of wait states can be programmed for each aperture in the FlexBus Controller. The transaction protocol is programmed by the host into a "program store" which holds the protocol programs. The program store is written through the register space, and occupies the registers with names like: FBC_MSTOREn. Multiple apertures may use the same protocol, and thus may point to the same routine in the program store. The READVEC and WRITEVEC fields of the FBC_APERn_CNTL registers contain the starting program store address for the read and writes routines, respectively. See *"Programming Considerations" on page 7-13* for details on programming the protocols. The protocol microstore has room for 64 instructions.

If the system designer wishes to support more than six devices on the FlexBus, he/she can have two devices share the same aperture (and hence, Chip Enable), but use upper address bits to decode which device is actually selected for any given cycle. Of course, devices grouped in this way must also share a common transaction protocol.

The Master Controller module of the FlexBus Controller ensures that there is no driving conflict on the tri-state FAD bus. If necessary, it will add an idle cycle on the FlexBus to do this. During an idle cycle, all Chip Enables are deasserted, and the Xilleon 220 is not driving the FAD bus. *Table 7-1 on page 7-12* shows the conditions under which the FlexBus Controller will insert an idle cycle between tasks on the FlexBus.

Table 7-1 Conditions Under which an Idle Cycle is Added

Next Task	Current Task		
	Read Aperture	**Write Aperture**	**Update Address Latch**
Read Same Aperture	-	yes	yes
Read Different Aperture	yes	yes	yes
Write Same Aperture	yes	-	-
Write Different Aperture	yes	-	-
Update Address Latch	yes	-	-

7.3.5 Programming Considerations

7.3.5.1 MicroController Programming

Transaction protocols for the FlexBus are created by placing short programs in the microcontroller "program store" memory. Each aperture has a start address associated with it which indicates to the microcontroller where the protocol routine for the current aperture begins. The microcontroller executes the code starting at the specified address until it reaches a halt instruction or an invalid command. Each protocol routine is constructed using the following commands:

Table 7-2 Protocol Routine Commands

Command	Code	Description
jump(nnnnn)	1nnnnn	Jump to (PC + nnnnn)
wait(nnn)	001nnn	Wait at this instruction for (nnn+1) clocks, then continue
wait_frdy	000010	Wait for FRDY to be '1', then continue
wait_nfrdy	000011	Wait for FRDY to be '0', then continue
wait_dfrdy	000110	Wait for synchronized (delayed) FRDY to be '1', then continue
wait_ndfrdy	000111	Wait for synchronized (delayed) FRDY to be '0', then continue
waith(nnn)	011nnn	Wait at this instruction for (nnn+1) clocks, then halt
wait_frdyh	010010	Wait for FRDY to be '1', then halt
wait_nfrdyh	010011	Wait for FRDY to be '0', then halt
wait_dfrdyh	010110	Wait for synchronized (delayed) FRDY to be '1', then halt
wait_ndfrdyh	010111	Wait for synchronized (delayed) FRDY to be '0', then halt
halt	000000	Halt at this instruction.

For each command code, the programmer supplies a data code which defines the state of the FlexBus signals for as long as the program remains on that command. The data code bits are assigned to the following control signals:

Table 7-3 Data Code Bits

Data Bit	Mnemonic	Control Signal
0	RD	Read -- to external pins
1	WE	Write Enable -- to external pins
2	STB	Strobe -- to external pins
3	CE	Chip Enable -- through decoder, then to external pins
4	RDS	Readback Data Strobe

Therefore, a program entry is a 11-bit word. The 6 MSbs are the command, and the 5 LSbs are the data.

Note that the RD, WE and STB signals connect directly to the output pads, so in effect the programmer has flexible control over defining how these pins are used or connected to an external device. For example, if a target device has a simple 2-pin interface (Chip Enable and Read/Write), then the system designer/programmer can use either the RD or WE signal to connect to the Read/Write pin of the device. The "Chip Enable" output pins are always active low; a '0' in the program store asserts the chip enable, and a '1' deasserts it.

When reading from an external FlexBus device, the RDS bit (ReadBack Data Strobe) enables the clocking of the data from a FlexBus device into a flip-flop internal to the FlexBus Controller. Data is clocked into a synchronous-loading D flip-flip.

If the RDS bit is a ‘1’ and the Command is one of: wait, wait_frdy, wait_nfrdy, wait_dfrdy, wait_ndfrdy, waith, wait_frdyh, wait_nfrdyh, wait_dfrdyh, or wait_ndfrdyh, the internal readback data register is loaded on the **last** clock of the wait, just before the program counter advances to the next instruction. The FRDY input signal can be sampled directly from the output pin, or it can be sampled after being ranked through one or two consecutive flip-flops to avoid metastability for external devices that use an asynchronous handshake protocol. The option of one flip-flop vs. two is controlled by a register-programmable control bit.

If the RDS bit is a ‘1’ and the Command is halt, the internal readback data register is loaded on the **first** clock of the halt only.

One rule the programmer must follow when writing the microroutine for a read transaction is that he/she must ensure that the readback data strobe is asserted for one and only one clock for any given read transaction. Otherwise, the FlexBus Controller will hang.

The programmer needs to understand the state of the FlexBus when the protocol microroutine is first called. For reads, the programmer can assume that the address is stable (in the latches). For writes, the address is stable, and the write data has just begun to be driven on the FAD[] bus.

A block diagram of the microcontroller circuit that runs the protocol programs is shown in *Figure 7-7*:

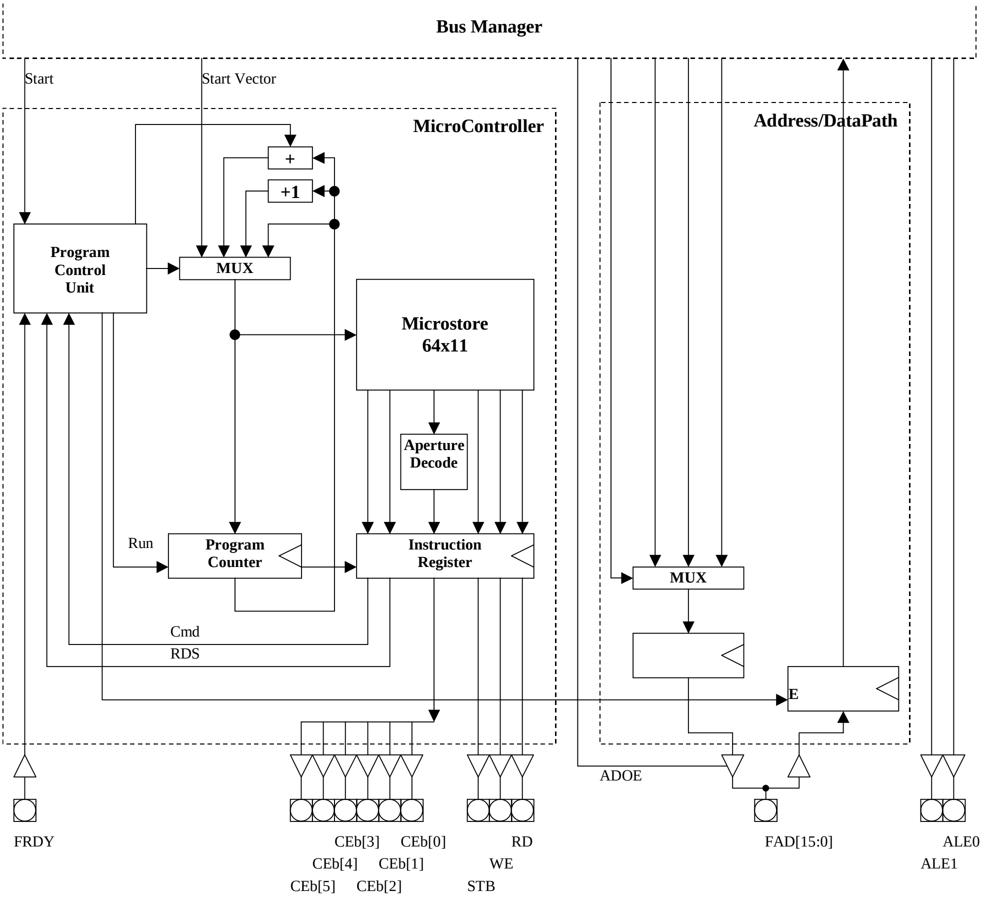


Figure 7-7 Block Diagram of the MicroController and Surrounding Logic

7.3.5.2 Modifying the Microcode

The microstore can be written at any time. If necessary, the hardware will stall a write to the microstore until any pending FlexBus transaction has been completed, and the microsequencer has become idle.

7.3.5.3 Endian Support

All transactions on the FlexBus follow the little-endian rule.

7.3.5.4 Interrupt Issues

If a device on the FlexBus needs to generate an interrupt to the on-chip processor of the Xilleon 220 or to the PCI bus, then the interrupt output from such a device should be wired to a GPIO input pin of the Xilleon 220. That pin is then interpreted as an interrupt input. This logic is **not** part of the FlexBus Controller.

7.3.5.5 Booting Issues

The FlexBus Controller powers-up in a state that allows access to a typical Boot ROM, using the Chip Enable 0 (FCE#[0]) pin.

There is a hard-strap input to the FlexBus Controller that tells it whether the Boot ROM aperture has an 8-bit device or a 16-bit device. The BOOTROM_DBUSWID strap initializes the DBUSWIDTH field of the FBC_APER0_CNTL register. The ABUSLATCH field is initialized to '11', which means that both of the external FlexBus address latches carry valid address bits to the boot device. The READVEC and WRITEVEC fields are initialized to "0", so the routine at microstore location zero is used for all reads and writes immediately after reset.

The arbiter resets such that no device owns the FlexBus so that the output pins are tri-stated. Upon the very first request to the FlexBus Controller, the Master Controller will request ownership of the bus, and the arbiter will begin to operate.

A default microroutine for ROM read is placed at location 0 of the microstore. The program is as follows:

```
location 0:   001111_00110;   // wait(8),    !RDS,  !CE,   STB,   WE,   !RD
location 1:   001111_10110;   // wait(8),    RDS,   !CE,   STB,   WE,   !RD
location 2:   011001_01111;   // waith(2),   !RDS,  CE,    STB,   WE,   RD
```

This sequence asserts RD (low) for 16 clocks, strobing in the read data on the last clock. It has a 2-clock period when the Chip Enable is unasserted between successive read cycles.

In SOLO mode, the start-up FlexBus clock has a 40.7 ns cycle time (24.576 MHz), which results in about a 650 ns pulse on the FRD signal.

In PEER or PEERPLUS mode, the start-up FlexBus clock is sourced from the PCI bus clock, which is either 30 ns (33 MHz) or 15 ns (66 MHz). These clocks result in 480 ns or 240 ns read pulses, respectively.

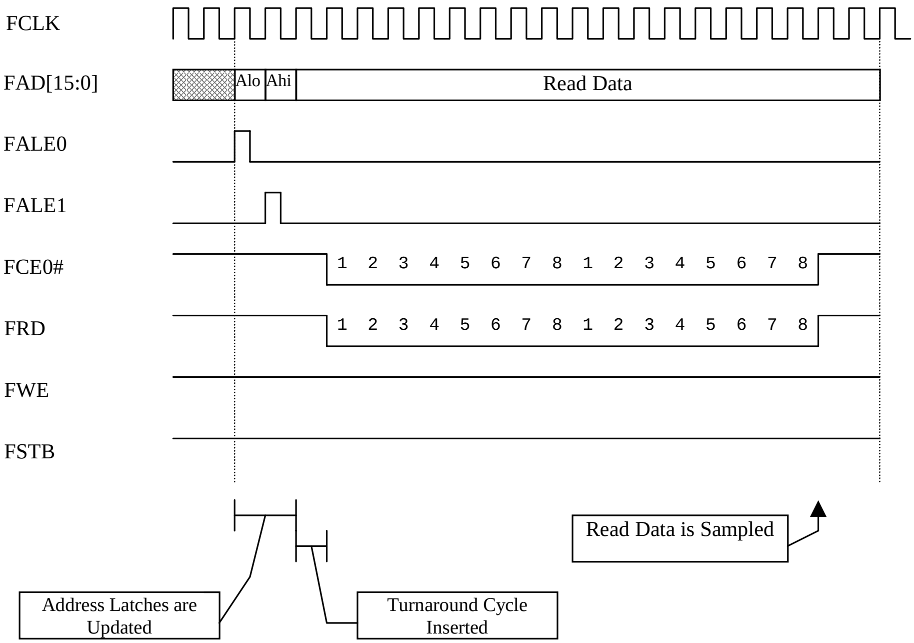


Figure 7-8 FlexBus Cycles

7.3.6 External Chip Interfaces

7.3.6.1 FlexBus Pins

The FlexBus signals are mapped to 33 pins on the Xilleon 220 device, according to *Table 7-4, “FlexBus Pin Usage,” on page 18*

Any interrupt inputs are **not** part of the FlexBus Controller; they are brought in from general purpose I/O pins.

Table 7-4 FlexBus Pin Usage

Ordinal	Pin Name	Description	Type	Notes
1	FAD[0]	A16_A0_D0	In/Out	Three-way multiplexed address/data
2	FAD[1]	A17_A1_D1		
3	FAD[2]	A18_A2_D2		
4	FAD[3]	A19_A3_D3		
5	FAD[4]	A20_A4_D4		
6	FAD[5]	A21_A5_D5		
7	FAD[6]	A22_A6_D6		
8	FAD[7]	A23_A7_D7		
9	FAD[8]	A24_A8_D8		
10	FAD[9]	A25_A9_D9		
11	FAD[10]	A26_A10_D10		
12	FAD[11]	A27_A11_D11		
13	FAD[12]	A28_A12_D12		
14	FAD[13]	A29_A13_D13		
15	FAD[14]	A30_A14_D14		
16	FAD[15]	A31_A15_D15		
17	FBE[0]	Byte Enable for data[7:0]	Out	If transaction is 16-bits, is BE[0], else A[0]
18	FBE[1]	Byte Enable for data[15:8]	Out	If transaction is 16-bits, is BE[1], else A[1]
19	FCE#[0]	Chip Enables	Out	Timing is programmable
20	FCE#[1]			Timing is programmable
21	FCE#[2]			Timing is programmable
22	FCE#[3]			Timing is programmable
23	FCE#[4]			Timing is programmable
24	FCE#[5]			Timing is programmable
25	FALE0	Address Latch Enable, on A[15:0]	Out	
26	FALE1	Address Latch Enable, on A[31:16]	Out	
27	FSTB	Strobe, aka Transfer Start	Out	Timing is programmable
28	FRDY	Ready, aka Transfer Ack	In	Timing is programmable
29	FWE	Write Enable	Out	Timing is programmable
30	FRD	Read	Out	Timing is programmable
31	FGNT#	Grant of Bus Ownership	Out	
32	FREQ#	Request for Bus Ownership	In	
33	FCLK	FlexBus Clock	Out	

7.3.6.2 Address Bus Connection

This section describes how 8-bit and 16-bit devices must be wired to the address bus of the FlexBus.

The rules can be simply stated:

- 8-bit devices must have their two least-significant address pins wired to the FBE[1:0] pins of Xilleon 220. Note that they should **not** be wired to the latch output corresponding to bit 0 of the FAD bus. i.e.,
  A[0] of the device connects to FBE[0], and
  A[1] of the device connects to FBE[1].
- 16-bit devices must have their byte enable pins connected to FBE[1:0], and their A[1] input pin connected to the output of the latch on the FAD[1] signal. i.e.,
  A[1] of the device connects to AQ[1], and
  BE[0] of the device connects to FBE[0], and
  BE[1] of the device connects to FBE[1].

The reason that 8-bit devices cannot be connected to the output of an address latch is because for 8-bit devices, the FlexBus Controller will **not** update the address latch (for higher performance) if only bits 1 or 0 of the address bus have changed.

Figure 7-9,“Address Bus Connection for 8-bit and 16-bit devices,” on page 20 shows this wiring. Note that the FlexBus supports connection to both 8-bit and 16-bit devices in the same system.

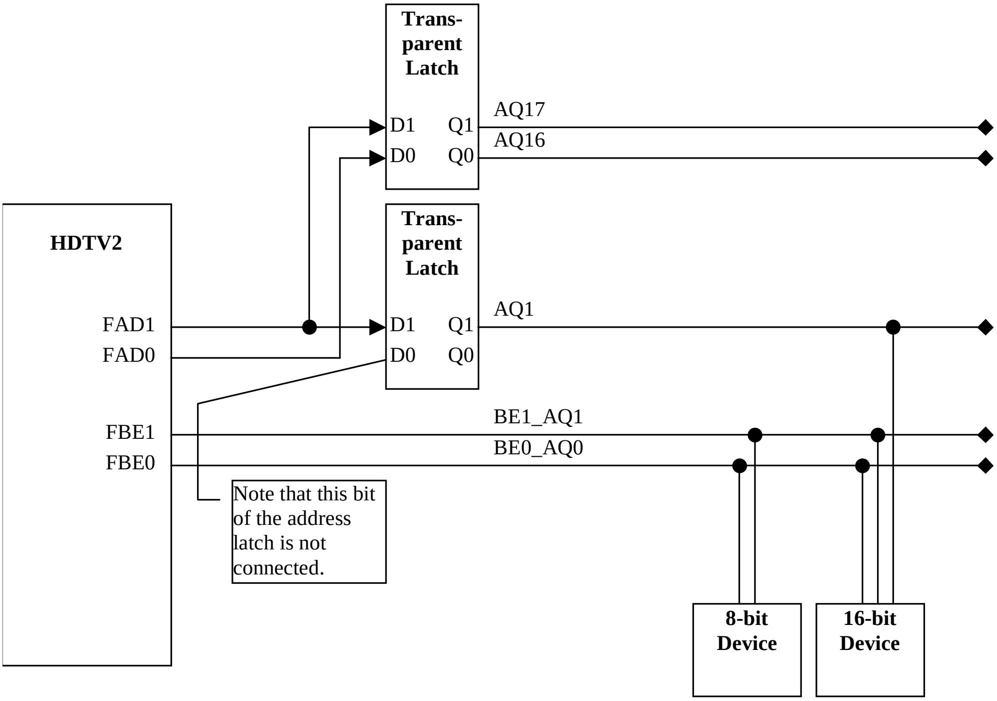


Figure 7-9 Address Bus Connection for 8-bit and 16-bit devices

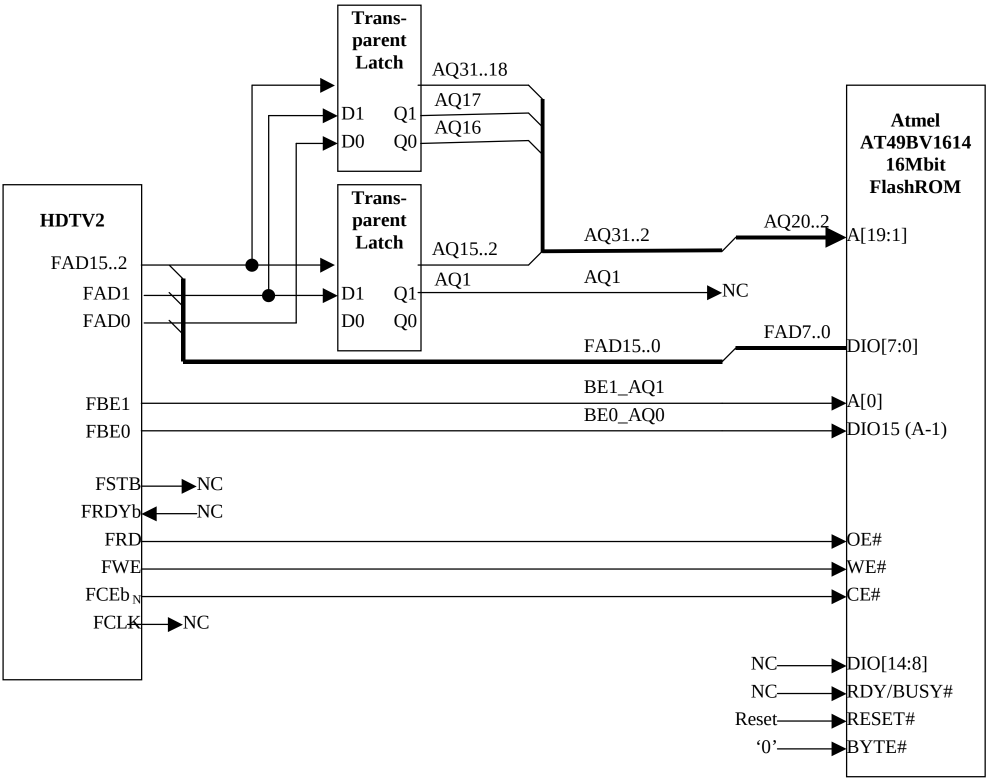


Figure 7-10 Example Connection to 8-bit Flash ROM Device

7.3.7 Performance

The FlexBus Controller operates up to a clock frequency of 66 MHz. As an example, assuming a 16-bit data bus, and, say, 4-clocks per-transaction, the maximum throughput would be 33 MB/s.

7.4 Universal Asynchronous Receiver/Transmitters (UARTs)

7.4.1 Overview

The Xilleon 220 contains two PC-compatible UARTs.

A UART supports the serial interchange of data between computing equipment, and typically operates in the range of 300 bits/sec to 115,000 bits/sec. This type of communication typically follows the EIA RS-232 standard.

For transmission, the CPU writes a byte to the UART, and the UART assembles a "Serial Data Unit" by concatenating the necessary START, STOP, and PARITY bits. It then performs parallel-to-serial conversion to transmit the data over a single wire.

For reception, the UART samples the incoming serial signal, performs serial-to-parallel conversion, and based on the programmed data format, extracts the character, and places it into the FIFO for the CPU to read. The receiver also does error-checking and reports this status back to the CPU.

The UART contains a programmable baud-rate generator which controls the speed of both the transmitter and the receiver.

7.4.2 Feature List

Both of the UARTs are identical, and possess the following features:

- The transmitter and receiver each have a 16-character FIFO, with a programmable threshold. (FIFOs can be disabled for legacy compatibility.)
- Programmable data format:
  - Character size: 5, 6, 7, or 8-bit
  - Parity generation/detection: None, Even, Odd, MARK, SPACE.
  - Number of stop bits: 1, 1.5, 2.
- Fully programmable baud rates from DC to 1.5M baud.
- Control and Status registers for RS-232 MODEM signals.
- Line BREAK generation/detection.
- Full interrupt support.
- Error detection and reporting.
- Loopback mode for testing.

Both UARTs implement the full 8-pin serial interface, including the MODEM control signals.

The Peripheral DMA Unit can be used to service either or both of the UARTs, moving data between the UART and the SDRAM memory. The use of DMA reduces the frequency of interrupts to the CPU, and decouples the CPU from the data-rate of the UART itself. See *“Peripheral DMA Unit” on page 7-43* for details on the DMA Unit.

7.4.3 Functional Description

7.4.3.1 Block Diagram

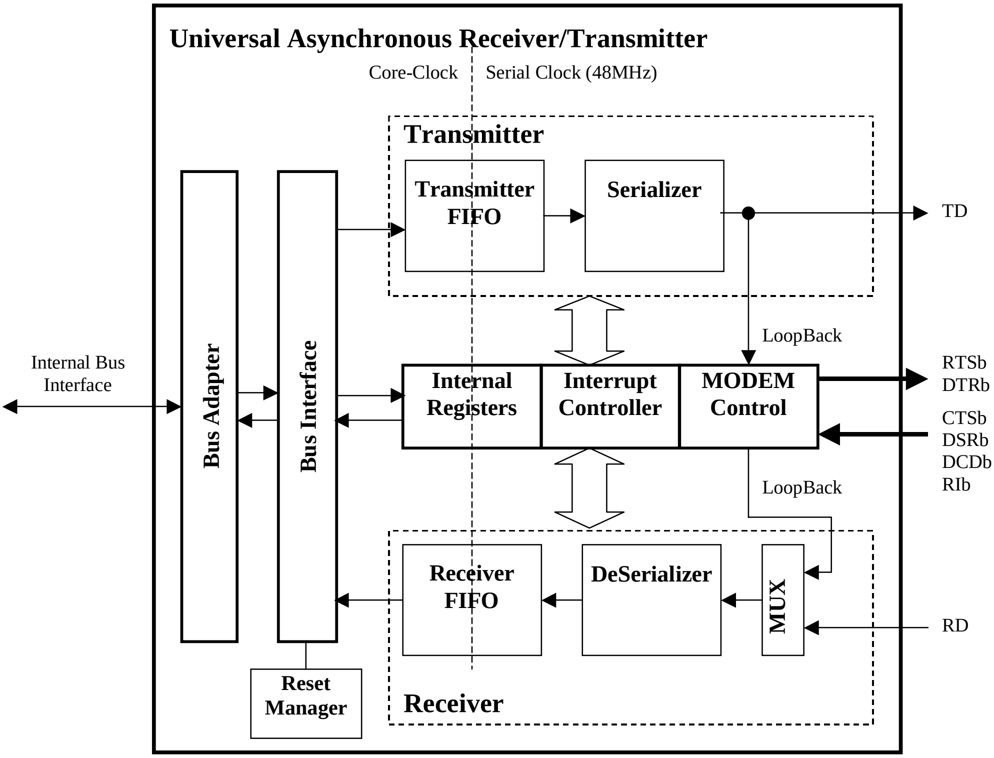


Figure 7-11 Block Diagram of the UART

7.4.3.2 Theory of Operation

The CPU programs the UART via the Internal Bus Interface shown in the block diagram. In addition to register reads/writes to control the UART itself, the reads and writes of the receive and transmit FIFOs also occur through this interface.

Since the internal bus is 32-bits wide, and the core UART is an 8-bit device, the Bus Adapter unit converts transactions on the internal bus interface into (possibly multiple) 8-bit transactions to the Bus Interface unit.

The Bus Interface unit manages reads and writes to the registers and transmit/receive FIFO's of the UART. It decodes the incoming opcode and address, and routes the transaction to one of the following: Transmit FIFO, Receive FIFO, or Internal registers.

Note that writes to the TxFIFO are not blocked if the TxFIFO is full, so it is the host's responsibility to make sure that there is an available entry in the FIFO before he writes to it. Similarly for the RxFIFO; a read from an empty RxFIFO is not blocked, so the host must first check the status of the FIFO before reading from it, or else it will read indeterminate data.

The receiver deserializer has detection logic for the following types of conditions: Parity Error, Framing Error, Break, and RxFIFO Overrun. The Parity Error, Framing Error, and Break signals get loaded into the Receive FIFO along with the data word which has the error, so the status register (UARTn_LSR) always shows the status of the byte which is the next to be read from the Receiver FIFO.

A Framing Error means that the Deserializer has detected a "0" bit when it was a expecting a STOP bit (STOP bits are always "1"s). When a framing error is detected, the receiver then tries to get back into synchronization by assuming that the "0" bit is a START bit for the next frame.

When the receiver is idle and it detects a high-to-low edge, it assumes this edge is the beginning of the START bit. Then, it samples the START bit again in the middle of the bit-period to make sure the line is still a "0". If it is not a "0", the receiver assumes that the transition was a noise pulse, and goes back to the idle state, waiting for another START bit.

When a Break is detected, the receiver places a single "all 0" data character into the RxFIFO, and then does not place any more characters into the RxFIFO until a "non-Break" character is received.

7.4.3.3 Registers

The table below provides a quick reference, indicating how the various programmable registers of the UART affect each of the sub-blocks in the Block Diagram.

Table 7-5 Registers vs. UART Sub-blocks

Register	Transmitter	Receiver	Interrupt Controller	MODEM Control	General
UARTn_FCR	X	X			
UARTn_IER			X		
UARTn_IIR			X		
UARTn_LCR	X	X			
UARTn_LSR	X	X			
UARTn_MCR				X	
UARTn_MSR				X	
UARTn_RBR		X	X		
UARTn_SCR					X
UARTn_THR	X				

The table below shows how the UART registers are arranged in the address space of the CPU. Note that address locations 0 and 1 have unique behavior — namely, the register which is accessed at this address is actually selected by another programmable bit in the LCR register (bit 7, "DLAB"). For example, a byte-write to address 0 when the DLAB is "1" will result in a write to the DLL register; and a write to address 0 when the DLAB bit is “0” will result in a write to the THR register (or TxFIFO).

Table 7-6

DLAB	A[2:0]	Write	Read
0	0	THR (Transmitter Holding Register)	RBR (Receive Buffer Register)
0	1	IER (Interrupt Enable Register)	IER (Interrupt Enable Register)
x	2	FCR (FIFO Control Register)	IIR (Interrupt Identification Register)
x	3	LCR (Line Control Register)	LCR (Line Control Register)
x	4	MCR (MODEM Control Register)	MCR (MODEM Control Register)
x	5	LSR (Line Status Register)	--
x	6	MSR (MODEM Status Register)	MSR (MODEM Status Register)
x	7	SCR (ScratchPad Register)	SCR (ScratchPad Register)
1	0	DLL (Divisor Latch LSB)	DLL (Divisor Latch LSB)
1	1	DLM (Divisor Latch MSB)	DLM (Divisor Latch MSB)

7.4.3.4 Interrupts

The UART has five types of interrupting conditions that it checks. Rather than having a "one-bit-per-interrupt" mechanism for reporting the status, the UART hardware has a built-in priority encoder, and reports to the software just the highest-priority interrupt that is currently active. The bottom four bits of the UARTn_IIR register tell the user the highest priority of interrupt that has been encountered. The table below shows the priorities of the various interrupt types as well as the mechanism that should be employed to clear the cause of the interrupt.

Table 7-7 Interrupt Types and Priorities

IIR[3:0]	Priority	Interrupt Type	Interrupt Source	Interrupt Reset Control
0001	--	None	None	--
0110	Highest	Receiver Line Status	Overrun Error, or Parity Error, or Framing Error, or Break Interrupt	Read the LSR register
0100	Second	Received Data Available	Receiver Data Available, or Trigger Level Reached	Reading the RBR Register, or the FIFO drops below the trigger level
1100	Second	Character Timeout Indication	No characters have been removed from, or input to, the RxFIFO during the last four character times, and there is at least one character in it during that time.	Read the RBR register

Table 7-7 Interrupt Types and Priorities (Continued)

IIR[3:0]	Priority	Interrupt Type	Interrupt Source	Interrupt Reset Control
0010	Third	TxFIFO is empty	TxFIFO is empty	Read the IIR register (if IIR[3:0]==0010), or Writing into the THR register
0000	Fourth	MODEM Status	CTS or DSR or RI or DCD has changed since the last read of the MSR register	Read the MSR register

7.4.3.5 Baud Rate Generator

The baud rate generator of the UART is based on a 48 MHz reference clock. For software compatibility with PC-based UARTs, however, the baud rate divisor registers have been implemented as if the reference clock were 24 MHz, since this is a popular frequency used in PC-based systems. The second column of the table below indicates the decimal value that must be programmed into the Divisor Latch Registers, in order to obtain the actual baud rate.

Table 7-8 Baud Rate Divisor Programming

Desired Baud Rate	Decimal Divisor for "16x baud" clock	Actual Baud Rate	Percent Error
50	30000	50.0000	0.0000 %
75	20000	75.0000	0.0000 %
110	13636	110.0029	0.0027 %
134.5	11152	134.5050	0.0037 %
150	10000	150.0000	0.0000 %
300	5000	300.0000	0.0000 %
600	2500	600.0000	0.0000 %
1200	1250	1200.0000	0.0000 %
1800	833	1800.7202	0.0400 %
2000	750	2000.0000	0.0000 %
2250	667	2248.8755	0.0500 %
2400	625	2400.0000	0.0000 %
2500	600	2500.0000	0.0000 %
3600	417	3597.1223	0.0799 %
4800	313	4792.3322	0.1597 %
7200	208	7211.5384	0.1603 %
9600	156	9615.3846	0.1603 %
14400	104	14423.0769	0.1603 %
19200	78	19230.7692	0.1603 %
28800	52	28846.1538	0.1603 %
38400	39	38461.5384	0.1603 %
56000	27	55555.5555	0.7937 %
57600	26	57692.3076	0.1603 %

Table 7-8 Baud Rate Divisor Programming

Desired Baud Rate	Decimal Divisor for "16x baud" clock	Actual Baud Rate	Percent Error
115200	13	115384.6153	0.1603 %

7.4.4 External Chip Interfaces

7.4.4.1 Serial Port Data Interface

The figure below shows how the UART transmits an 8-bit character (with parity and one stop bit) on a serial data line. The START bit is always a '0', and the STOP bit is always a '1'.

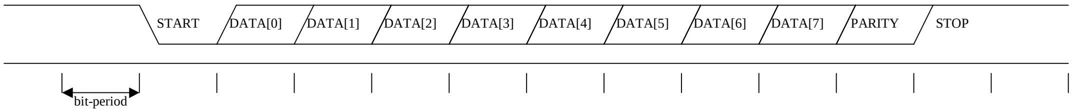


Figure 7-12 UART Transmission

7.4.5 Performance

The UART can operate at speeds up to 1.5M baud. This value is based on a 24 MHz reference clock, divided by 16, as the UART performs 16x oversampling of the input signal.

7.5 Universal Infrared Receiver/Transmitter (UIRT)

7.5.1 Overview

The UIRT supports “Universal” consumer IR transmit/receive, including, but not limited to, the following IR protocols: RC-5, RC-5(extended), RECS-80, NEC, Sony, and RCA. It may also be able to support Sharp-IR and some of the IrDA modes by writing the appropriate software. The UIRT is able to transmit at the same time as receive (full duplex). It is also able to use different IR protocols on the transmit-side and the receive-side simultaneously.

The IR receiver is primarily intended to accept commands from a remote control device, for channel-changing and navigation of menus in set-top boxes.

The IR transmitter can be used to transmit commands to other consumer electronic devices, such as analog VCRs, DVD players, and TV’s (among other devices). For example, an analog VCR may be powered up and automatically commanded to record a program the scheduling information of which is looked up in an Electronic Program Guide by the CPU.

The primary goal of the UIRT, on the transmission-side, is to generate a specific waveform on its output signal. This is the level at which the software interacts with the hardware, i.e., the software instructs the hardware how long the signal should be '1', how long it should be '0', and a series of these "commands" is used to compose a waveform that conveys the data which is to be transmitted.

On the receiver-side, the UIRT hardware essentially does the opposite of what the transmitter does. At the hardware/software interface, the hardware conveys to the software how long the input waveform was '1' and how long it was '0', and provides a list of these status words for the software to decode and interpret. The driver software for the receiver must be written in a very robust fashion, as IR transmission is often very noisy, and packets of data are often fragmented in some way due to the "human-element" at the other end of the remote control. The driver software must be able to "throw away" such fragmented packets.

7.5.2 Feature List

- The transmitter and receiver each have a 32-deep FIFO, with a programmable threshold.
- Flexible transmitter/receiver command interface, using RLE encoding, or just sampled data.
- Fully programmable transmitter carrier frequency and duty cycle.
- Built-in demodulator in the receiver, with fully programmable carrier frequency.
- Programmable noise filter in the receiver.
- Full interrupt support.
- Error detection and reporting.
- Special features to prevent transmitter underrun and receiver overrun.
- Loopback mode for testing.

The Peripheral DMA Unit can be used to service the UIRT, moving data between the UIRT and the SDRAM memory. The use of DMA reduces the frequency of interrupts to the CPU, and decouples the CPU from the data-rate of the UIRT itself. See *“Peripheral DMA Unit” on page 7-43* for details on the DMA Unit.

7.5.3 Functional Description

7.5.3.1 Block Diagram

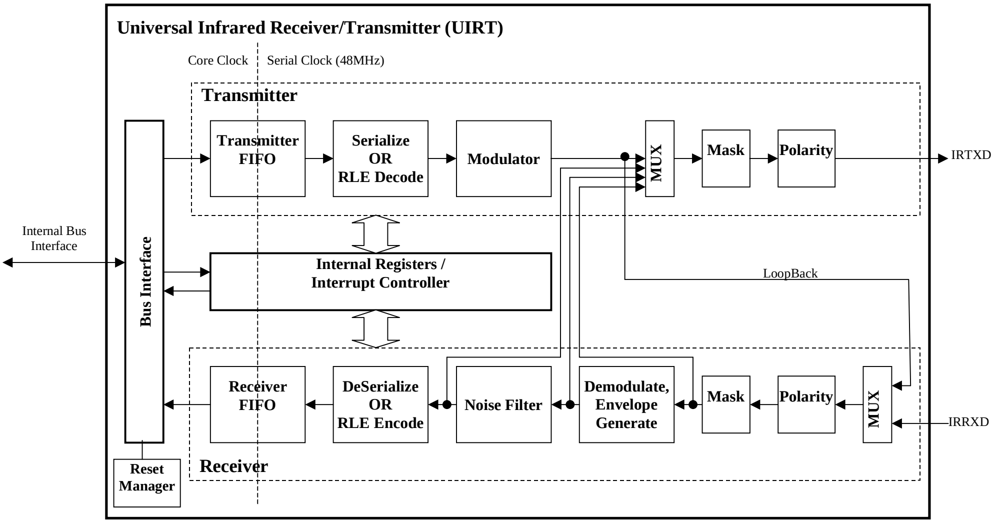


Figure 7-13 Block Diagram of the UIRT

7.5.3.2 Theory of Operation

The *Bus Interface* unit manages reads and writes to the registers and transmit/receive FIFO’s of the UIRT. It decodes the incoming opcode and address, and routes the transaction to one of the: Transmit FIFO, Receive FIFO, or Internal registers. It also manages the handshake with the internal bus.

On the "output-side" of the transmitter FIFO, the *Serializer* block looks at the command from the FIFO and decides whether to serialize the data in the command or to decode the Run-Length Encoded data in the command.

The *Modulator* inserts the carrier signal into the output waveform whenever the input waveform has a ‘1’ value. With regard to the phase, the Modulator always begins the carrier signal with a full high pulse, and continues to oscillate the output as long as the input is a ‘1’. In other words, the rising edge of the input is detected and this initializes the counter that generates the carrier signal. See *Figure 7-14,“Input and Output Signals of the Transmitter of the UIRT,” on page 30*.

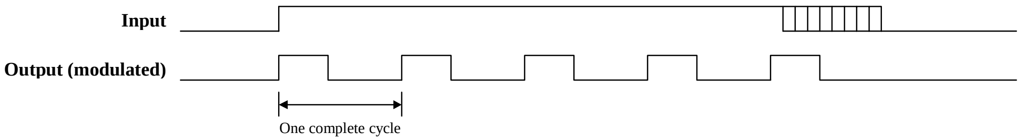


Figure 7-14 Input and Output Signals of the Transmitter of the UIRT

The *Receiver* effectively does the reverse of the Transmitter. The *Demodulator* extracts the original signal from the carrier, and presents it to the Noise Filter. If the signal coming into the Xilleon 220 pin is not modulated, then the demodulator can be bypassed.

The *Noise Filter* can be programmed to filter-out spikes on the signal that are shorter in duration than a certain programmable threshold. The noise filter is implemented with a programmable-tap median filter. A median filter examines the N most recent consecutive samples (N is an odd number). If there is a majority of 1's, then the output is 1; otherwise the output is 0.

The noise filter operates in one of 2 modes: Always On, or Automatic Turn-On/Off. In the second mode, the filter starts-up being ON, and when a pulse passes through the filter, it turns itself OFF. Then, when all of the N most-recent samples are '0', it turns itself back ON. The UIRT_RCVR_NOISE_CNTL register contains the parameters that control the noise filter.

The *Deserializer/RLE Encoder* module samples the input signal, optionally computes the run-length, and places the formatted value into the RxFIFO for eventual consumption by the host processor.

7.5.3.3 Registers

Table 7-9 on page 7-30 provides a quick reference, indicating how the various programmable registers of the UIRT affect each of the sub-blocks in the Block Diagram.

Table 7-9 Registers vs. UIRT Sub-blocks

Block Name Reg. Name	Transmitter	Receiver	Interrupt Controller	General
UIRT_XMIT_FIFO	X			
UIRT_XMIT_CNTL	X			
UIRT_XMIT_MOD_CNTL	X			
UIRT_RCVR_FIFO	X			
UIRT_RCVR_CNTL		X		
UIRT_RCVR_DEMOD_CNTL		X		
UIRT_RCVR_NOISE_CNTL		X		
UIRT_SAMPLE_CLKDIV	X	X		
UIRT_STATUS	X	X		

Table 7-9 Registers vs. UIRT Sub-blocks (Continued)

Block Name Reg. Name	Transmitter	Receiver	Interrupt Controller	General
UIRT_CNTL	X	X		X
UIRT_INTR_ENABLE			X	
UIRT_INTR_STATUS			X	
UIRT_INTR_CNTL			X	

7.5.3.4 Interrupts

The UIRT can generate an interrupt under any of the following circumstances. Each of the interrupts is independently controlled in terms of enable/disable, and clearing.

XMIT_FIFO_STAT	The Transmitter FIFO has less than or equal to XMIT_FIFO_THRESHOLD number of occupied elements.
XMIT_UNDERRUN_STAT	The Transmitter FIFO has gone from non-empty to empty.
XMIT_EOT_STAT	A Transmit command that had its ENABLE_EOT_INTR bit set has completed transmission.
RCVR_FIFO_STAT	The Receiver FIFO has greater than or equal to RCVR_FIFO_THRESHOLD number of occupied elements.
RCVR_OVERRUN_STAT	The Receiver FIFO has been overrun, meaning that receive data has come in at a time when the FIFO was full. This helps to signal an error or warning to the software, so that it becomes aware that an unusual condition has occurred.
RCVR_TIMEOUT_STAT	The Receiver FIFO has timed-out, that is, there have been one or more items sitting in the receive FIFO for a while since the last read of the FIFO.

7.5.3.5 Transmitter Programming

The task that the programmer faces is to program the Transmitter in order to send a specific IR protocol. As an example, assume that a protocol with the characteristics shown in *Table 7-10 on page 7-32* is used. See *“IR Background” on page 7-38* for background information on IR transmission in general.

Table 7-10 Example - Protocol Characteristics

Packet Duration	Repeat Interval	Encoding Style	Header Format	‘0’ Format	‘1’ Format	No. of code bits	“Special” Repeat Code
Variable	45 ms	Pulse	P 2600 S 400	P 800 S 400	P 1400 S 400	12	None

First set-up the transmitter. The following three registers are written in order to set it up:

UIRT_SAMPLE_CLKDIV . XMIT_SAMPLE_DIV = 9600 clocks.

UIRT_XMIT_MOD_CNTL . XMIT_CARRIER_DIV = 1200 clocks.
UIRT_XMIT_MOD_CNTL . XMIT_CARRIER_HI = 400 clocks.
UIRT_XMIT_MOD_CNTL . XMIT_MOD_EN = 1

UIRT_XMIT_CNTL . UNDERRUN_CNTL = 0
UIRT_XMIT_CNTL . XMIT_FIFO_THRESHOLD = 0
UIRT_XMIT_CNTL . TXD_SELECT = 0
UIRT_XMIT_CNTL . TXD_MASK = 1
UIRT_XMIT_CNTL . TXD_POLARITY = 1

In considering the sample clock for this protocol, it is observed that the smallest divisible time unit for all of the possible pulses and spaces is 200µs. It makes sense to set the sample clock to this interval. So, the sample clock divider is computed as:

XMIT_SAMPLE_DIV = 200us*48MHz = 9600 clocks.

The carrier frequency is 40kHz with a 33% duty cycle, so make the following computations:

XMIT_CARRIER_DIV = 48MHz/40kHz = 1200 clocks.
XMIT_CARRIER_HI = 48MHz/40kHz * .33 = 400 clocks.

The XMIT_UNDERRUN_CNTL field is set to ‘0’ so that whenever transmission is not occurring, the output of the transmitter is a ‘0’, thus sending a Space (i.e., no IR pulse).

The XMIT_FIFO_THRESHOLD field really only needs to be programmed if interrupts or DMA are being used. This threshold sets a point in the transmit FIFO at which the host or DMA controller wishes to be told that the transmit FIFO has enough available elements and that it is time for the host or DMA controller to provide more data to the UIRT.

The TXD_SELECT field is set to ‘0’ for normal transmit operation; all other settings are for debug or testing.

The TXD_MASK field is set to ‘1’ to allow the output of the transmitter to switch. Turning this bit to ‘0’ is a convenient way to simply put a constant Space out on the transmitter output.

The setting of the TXD_POLARITY field depends upon the physical transmission circuit that one connects to on the outside of one’s device. A value of ‘1’ means that a transmitter output of ‘1’ is a Pulse, and a ‘0’ is a Space.

After set-up, one is ready to start sending IR pulses out through the transmitter.

Pulses are transmitted by writing to the Transmit FIFO with the ENABLE_XMISSION bit set to ‘1’. If the programmer writes to the FIFO with that bit set to ‘0’, then the command will be placed into the FIFO and will wait there until a subsequent command with that bit set to ‘1’ is written into the FIFO. This allows the programmer to fill-up the FIFO with a sequence of commands and then trigger them to transmit once all of them are queued-up. This helps to alleviate Transmit FIFO of underrun situations if, for some reason, the Transmit FIFO cannot be filled quickly enough to keep up with the transmitter.

Commands for the Transmitter have the following format:

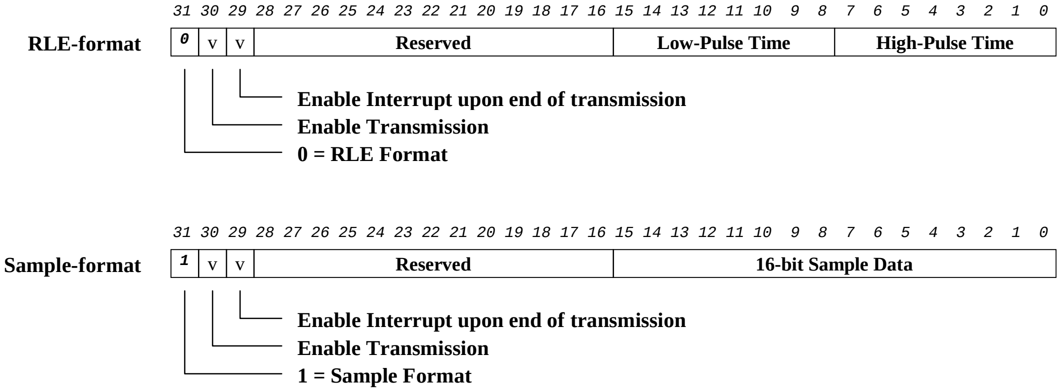


Figure 7-15 Format of Commands for the Transmitter

Now, assume that one wishes to send the following 12-bit code word: 0011_0101_1010. The example protocol has a header, and it uses Pulse-coding to represent 1’s and 0’s for transmission. The first decision to make is, based on the chosen sample frequency, whether it is more efficient (in terms of number of commands) to use the RLE commands or the raw Sample commands. With the 200µs sample clock, and a repeat interval of 45ms, there are 45ms/200us = 225 bit-periods in a packet.

If one uses all Sample commands, it will take ROUNDUP(225/16) = 15 commands to transmit a single packet.

If one uses RLE commands, it will take 13 commands to send the header plus the twelve data bits.

So far, RLE appears to be more efficient. But, another possibility is to mix sample commands with RLE commands, using from 6 to 8 commands (depending on the number of 1’s and 0’s in the code word) to send the sample commands for the header and 12 data bits, plus one more RLE command to “pad-out” to the end of the packet. This gives a total of 7 to 9 commands per-packet, thus averaging out to about 8 commands per-packet.

When making a decision about what type of format to use, the programmer may also want to weigh the difference between the processing time needed to generate the sample commands and that needed for the RLE commands. In addition, the programmer may also wish to consider the difference, between the two formats, in the number of commands per-packet.

Generally, RLE is simpler, so it will be used in the example. Therefore, to transmit 0011_0101_1010, the following commands would be fed to the transmit FIFO:

```
UIRT_XMIT_FIFO    <-   0x0000020D    -- Header   : 13 Hi,   2 Lo
UIRT_XMIT_FIFO    <-   0x00000204    -- ‘0’ Bit  : 4  Hi,   2 Lo
UIRT_XMIT_FIFO    <-   0x00000204    -- ‘0’ Bit  : 4  Hi,   2 Lo
UIRT_XMIT_FIFO    <-   0x00000207    -- ‘1’ Bit  : 7  Hi,   2 Lo
UIRT_XMIT_FIFO    <-   0x00000207    -- ‘1’ Bit  : 7  Hi,   2 Lo
UIRT_XMIT_FIFO    <-   0x00000204    -- ‘0’ Bit  : 4  Hi,   2 Lo
UIRT_XMIT_FIFO    <-   0x00000207    -- ‘1’ Bit  : 7  Hi,   2 Lo
UIRT_XMIT_FIFO    <-   0x00000204    -- ‘0’ Bit  : 4  Hi,   2 Lo
UIRT_XMIT_FIFO    <-   0x00000207    -- ‘1’ Bit  : 7  Hi,   2 Lo
UIRT_XMIT_FIFO    <-   0x00000207    -- ‘1’ Bit  : 7  Hi,   2 Lo
UIRT_XMIT_FIFO    <-   0x00000204    -- ‘0’ Bit  : 4  Hi,   2 Lo
UIRT_XMIT_FIFO    <-   0x00000207    -- ‘1’ Bit  : 7  Hi,   2 Lo
UIRT_XMIT_FIFO    <-   0x4000027A    -- ‘0’ Bit  : 4  Hi,   122 Lo (Enable Transmission)
```

The last command pads a Space out to satisfy the repeat interval of 45ms.

7.5.3.6 Receiver Programming

Since the receiver has more logical functions, it is more complicated to set up. The registers are grouped by the function that they control, so there is a register to setup each of the demodulator, the noise filter, and the receiver in general. Therefore, if the demodulator is not being used, its register can be set up to disable the demodulator, and left alone from then on. In fact, the default setting of the registers is such that the Demodulator and Noise Filter are disabled.

The following five registers are written in order to set up the receiver:

UIRT_SAMPLE_CLKDIV . RCVR_SAMPLE_DIV

UIRT_CNTL . RCVR_FIFO_TIMEOUT
UIRT_CNTL . EN_FULL_DUPLEX
UIRT_CNTL . LOOPBACK_EN

UIRT_RCVR_DEMOD_CNTL . RCVR_DEMOD_DIV
UIRT_RCVR_DEMOD_CNTL . DEMOD_QFACTOR
UIRT_RCVR_DEMOD_CNTL . DEMOD_THRESHOLD
UIRT_RCVR_DEMOD_CNTL . DEMOD_EN

UIRT_RCVR_NOISE_CNTL . RCVR_NOISE_DIV
UIRT_RCVR_NOISE_CNTL . NOISE_FILTER_MODE

UIRT_RCVR_NOISE_CNTL . NOISE_FILTER_CNTL
UIRT_RCVR_NOISE_CNTL . NOISE_START_THRESHOLD
UIRT_RCVR_NOISE_CNTL . NOISE_STOP_THRESHOLD

UIRT_RCVR_CNTL . RCVR_STOP_SAMPLES
UIRT_RCVR_CNTL . RCVR_START_STOP
UIRT_RCVR_CNTL . RCVR_SAMPLE_SYNC
UIRT_RCVR_CNTL . RCVR_MODE
UIRT_RCVR_CNTL . RCVR_OVERRUN_CNTL
UIRT_RCVR_CNTL . RCVR_FIFO_THRESHOLD
UIRT_RCVR_CNTL . RXD_POLARITY

Considerations when programming the receiver

As with the transmitter, one of the most fundamental software decisions is whether to take the received data in RLE format or Sample format. This is controlled by the RCVR_MODE register field. Sample format can be useful if the programmer wishes to gather certain data and run further digital filtering upon it, or if the sample clock frequency is not significantly higher than the highest frequency content of the input, thus causing “high-frequency” switching of the samples. In the high-frequency case, the Sample format can fit more information into the Receive FIFO than the RLE format can.

In most cases, for real IR protocol reception, the RLE format is probably easier to program in the software packet decoder. For the RLE format, the sample clock (RCVR_SAMPLE_DIV) can be set to 10 to 100 times the pulse interval, and allow fairly accurate discrimination of IR pulses and spaces.

The fields that control the fundamental setup of the receiver (independent of the IR protocol in use) are: EN_FULL_DUPLEX, LOOPBACK_EN, and RXD_POLARITY. The EN_FULL_DUPLEX field, when ‘0’, can dynamically disable the receiver while transmitting, thus avoiding the receiver from picking-up stray IR transmissions from one’s own device. The LOOPBACK_EN bit is mainly intended for testing the UIRT. The setting of the RXD_POLARITY field depends upon the physical IR detection circuit that one is connecting to on the outside of one’s device. A value of ‘1’ means that a receiver input of ‘1’ is a Pulse, and a ‘0’ is a Space. Many real-world detector devices have an active-low output.

Another important decision when programming the receiver involves **when** the receiver should actually be putting data into the Receive FIFO. For example, if the user is not pressing any remote-control keys, it is preferable not to take up CPU time by having the CPU read-out data from the receive FIFO that basically says that it is not detecting any IR light. The UIRT has three register fields that give the programmer very flexible control over when the receiver is placing results into the FIFO. These three fields are: RCVR_START_STOP, RCVR_SAMPLE_SYNC, and RCVR_STOP_SAMPLES.

The RCVR_START_STOP field dictates the basic operation of the receiver. The basic modes of operation are:

1. The host controls turning the receiver on and off.
2. The host turns on the receiver, and it runs until a programmed number of samples have been received.
3. The host turns on the receiver, and it runs until a programmed number of samples **of the same** value have been received.
4. The receiver turns on automatically when an edge is detected, and it runs until a programmed number of samples have been received.
5. The receiver turns on automatically when an edge is detected, and it runs until a programmed number of samples **of the same** value have been received.

The RCVR_STOP_SAMPLES field is used in conjunction with the above modes 2,3,4, and 5 to specify the programmed number of samples.

The RCVR_SAMPLE_SYNC field is a subtle control that specifies when the sample clock period should begin. The *Figure 7-16,“Effect of the RCVR_SAMPLE_SYNC field,” on page 36* gives a picture of the two different possibilities.

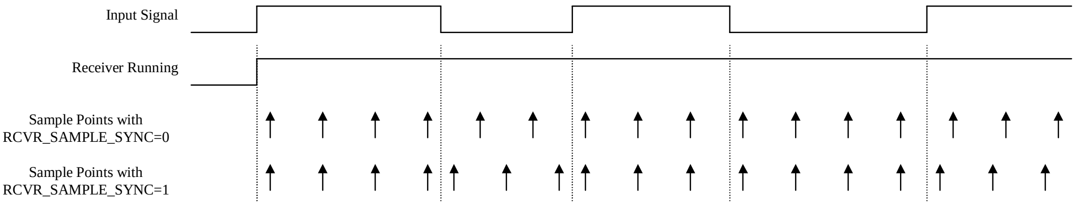

Input Signal
Receiver Running
Sample Points with RCVR_SAMPLE_SYNC=0
Sample Points with RCVR_SAMPLE_SYNC=1

Figure 7-16 Effect of the RCVR_SAMPLE_SYNC field

If the programmer needs to enable Demodulation, the RCVR_DEMOD_DIV field must be programmed to approximately 4-times the expected carrier frequency because the digital filter used for the bandpass filtering is designed for this exact rate of oversampling. The DEMOD_QFACTOR gives the programmer some flexibility in the “width” of the bandpass filter. This value varies from 1 to 6, with higher numbers meaning a “narrower” pass band, centered around the carrier frequency. It is advisable to set this value as high as possible while still permitting reliable operation. The narrower the filter, the better the receiver will be at rejecting IR noise. The DEMOD_THRESHOLD field controls the last stage of the demodulator, which is a Clipper. This is an unsigned 7.4 value, which is generally set to 2^DEMOD_QFACTOR. It also may be tweaked around that range slightly to increase reliability.

The Noise Filter is effective at removing spikes on the input signal. In order to program the noise filter, one must first determine the narrowest valid Pulse or Space used by the particular IR protocol of interest. One must ensure that the narrowest valid “feature” can consistently pass through the noise filter; otherwise good IR commands will be filtered out! The window of the median filter should be **slightly less than twice the width of the narrowest valid feature**. Once this *time* is determined, one must decide how many taps to make the filter. At the time of writing, it seems that the best approach is to use the maximum number of taps in order to achieve the best accuracy. Some further experimentation is warranted to see if this is a sound approach. The number of taps is programmed into the NOISE_FILTER_CNTL field. Once the number of taps is determined, then the absolute time is divided by the number of taps in order to determine the sample rate to be programmed into the RCVR_NOISE_DIV field.

The UIRT also has a feature that allows the noise filter to be automatically turned on and off during reception of an IR packet. This feature is intended to be used for IR protocols that have a long header at the beginning of each packet. (“Long” compared to the Pulses and Spaces for the code word bits). This feature is controlled by the NOISE_FILTER_MODE field. The goal is to have the noise filter ON during the time in-between packets, but OFF during the time when a packet is being received. The independent NOISE_START_THRESHOLD and NOISE_STOP_THRESHOLD fields allow the programmer to build some hysteresis into whatever circumstances that might turn the filter ON and OFF.

For the RCVR_OVERRUN_CNTL field, it is generally desirable to set this field to a ‘1’, which simply indicates to the Receiver FIFO to keep only the 32 most-recent entries. Therefore, in case the host CPU has gotten hopelessly behind in responding to IR input, it at least responds to the user’s most recent actions, which should give a better “feel” to the response of our device to the user’s pressing of keys on his remote control. However, some IR protocols send the command packet for an initial key press, and while the user holds the key depressed, the transmitter sends a “special” code that simply indicates “repeat the last command”. Since the repeat packet does not actually convey the command information, it is important that the original command stay in the receive FIFO, and not be overwritten by any succeeding commands. For this type of protocol, you may want to set the RCVR_OVERRUN_CNTL field to a ‘0’.

7.5.4 External Chip Interfaces

7.5.4.1 IR Port

The UIRT is connected to 2 external pins of the Xilleon 220; one for the transmission of data (IRTXD), and one for reception (IRRXD).

The protocol used on these pins is highly programmable; see the following section for a description of the signalling protocols used on these pins.

7.5.5 IR Background

7.5.5.1 Terminology

There does not seem to be consistent terminology used among the literature on infrared devices, so it is important to adopt a specific terminology in this document. This document will use the following terms to describe infrared transmissions:

Pulse	A time period during which there is IR light being emitted or detected.
Space	A time period during which there is **no** IR light being emitted or detected.
Packet	A logical group of IR pulses and spaces that comprise the transfer of one command from the transmitter to the receiver.
Header	The beginning portion of a packet, which is used to give the receiving circuit time to prepare for the stream of IR pulses which are to follow.
Code	The portion of a packet that contains the actual “logical” data that is being sent from the transmitter to the receiver. Logical 1’s and 0’s are conveyed as a combination of pulses and spaces of specific durations. The length of the code word can vary from, say 5, up to about 50 binary bits. The code word usually consists of several sub-fields, for example, Start bits, Stop bits, Device ID, Command, etc.

7.5.5.2 Physical Layer

IR remote control units communicate using light waves in the infrared spectrum, typically in the 940nm wavelength region.

Figure 7-17,“Typical IR System Diagram,” on page 39 shows a typical application of consumer remote control devices. The digital transmitter, modulator, and IR-emitting diode usually reside in a handheld remote control device, or, in the case for Xilleon 220, the IR-emitting diode and associated circuitry can be tethered to a set-top box and directed at the remote-control sensor located in the consumer’s VCR or TV.

On the receiving side of the system is the IR-detecting circuit which is composed of several analog functions, the task of which is to reconstruct the demodulated digital signal waveform as shown in *Figure 7-17*. It is then the task of the digital receiver to extract the command that is being conveyed by the IR pulses.

The light waves emitted by the transmitter are modulated using a carrier frequency typically in the range of 33 to 40 kHz (although Sharp-IR uses 500 kHz). Modulation is used to increase immunity to other sources of IR “noise”, such as fluorescent lamps, sunlight, etc. On the receiver side, the bandpass filter limits the receiver’s sensitivity to +/- 2KHz, close to the center frequency of the carrier wave, thus producing the final digital waveform. An AGC (automatic gain control) circuit adjusts the incoming level to the demodulator, which explains the presence of a long leading pulse in many of the IR protocols. This time allows the receiver to stabilize its AGC circuit prior to the reception of the code word.

There are some IR-detecting diodes that do not have the demodulator built-in. For such devices, we have also included an optional demodulator stage in the Xilleon 220's UIRT Receiver.

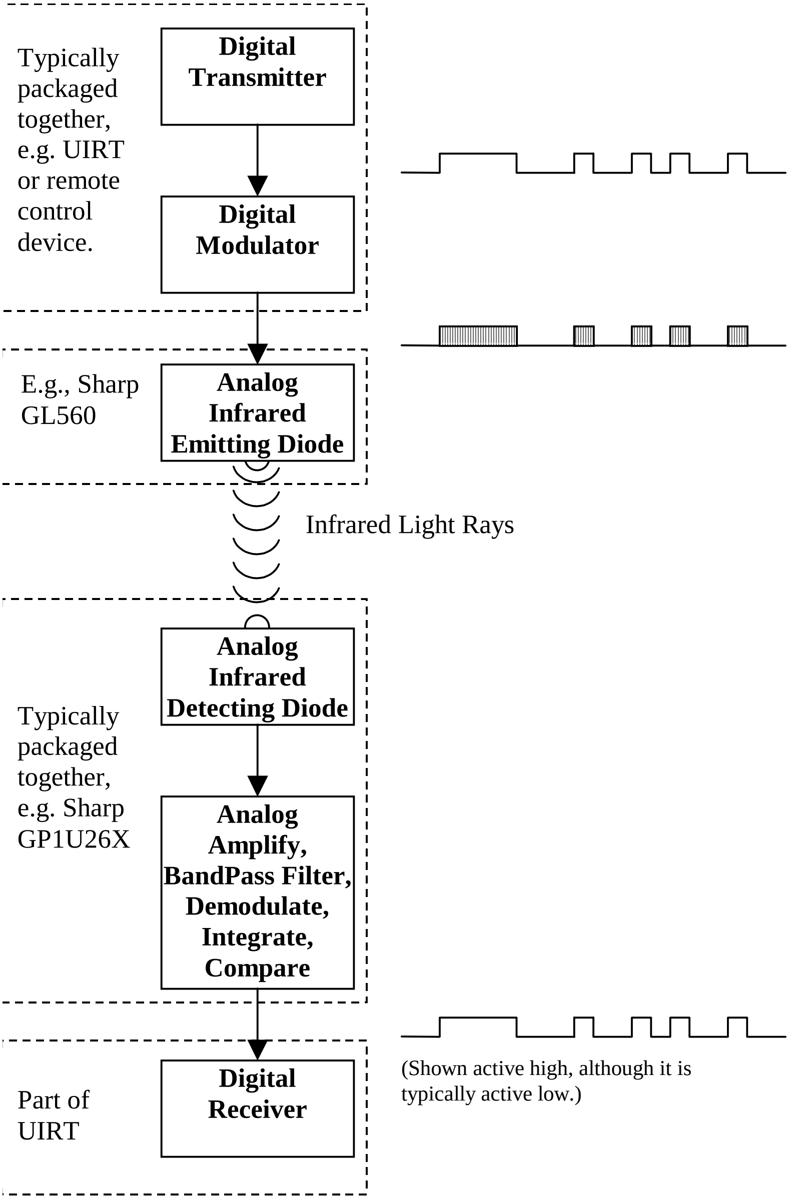

Typically packaged together, e.g. UIRT or remote control device.
Digital Transmitter
Digital Modulator
E.g., Sharp GL560
Analog Infrared Emitting Diode
Infrared Light Rays
Analog Infrared Detecting Diode
Typically packaged together, e.g. Sharp GP1U26X
Analog Amplify, BandPass Filter, Demodulate, Integrate, Compare
Part of UIRT
Digital Receiver
(Shown active high, although it is typically active low.)

Figure 7-17   Typical IR System Diagram

7.5.5.3 Data Layer

There are many different IR protocols that have come into use by the various manufacturers of consumer devices. To introduce the basic concepts of IR communication, this section gives an example of the most complex IR protocol. Most protocols are variations of this basic model, usually varying by:

- The presence or absence of a header.
- The number of bits per-packet.
- The encoding style of the code word (discussed further on in this section).
- How a “repeat key” event is conveyed.
- The duration of the pulses/spaces in the header or in the code word.

Figure 7-18,“A Typical IR Transmission Protocol,” on page 40 below shows a typical IR transmission, i.e., the digital signal that is fed into the IR-emitting diode.

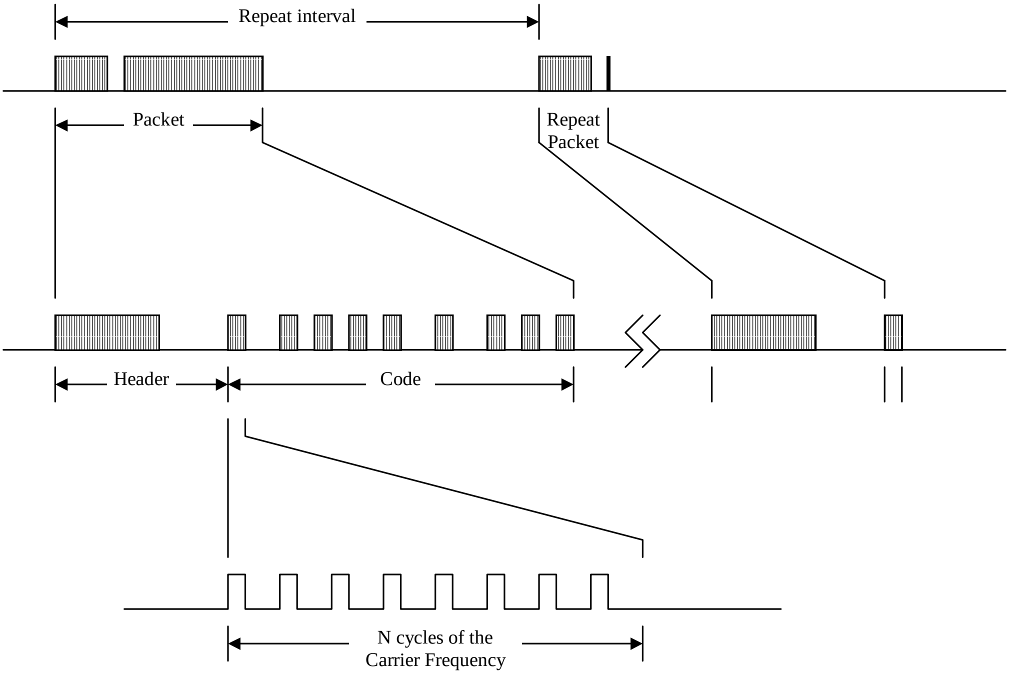


Figure 7-18 A Typical IR Transmission Protocol

An IR packet consists of a header, followed by a code word. The logical bits (‘1’ vs. ‘0’) in the code word are conveyed using one of three encoding methods: Pulse-coding, Space-coding, or Edge-coding. Note that some literature also refers to space-coding as pulse-position modulation (PPM), and edge-coding as bi-phase encoding or Manchester encoding.

Pulse Coded:
The duration of the pulse is varied to represent data, for example:

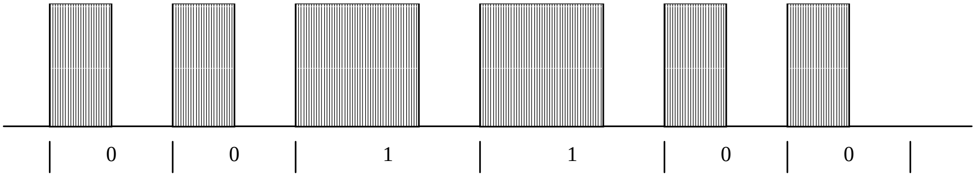


Figure 7-19 Pulse Coding Example

Space Coded:
The duration of the space is varied to represent data, for example:

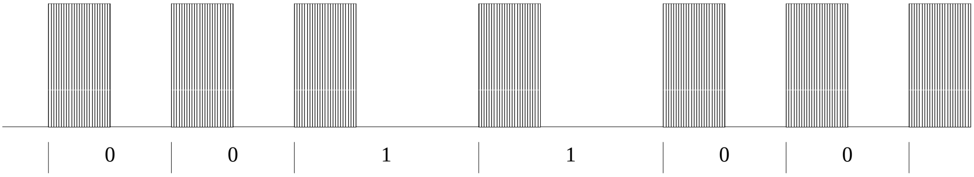


Figure 7-20 Space Coding Example

Edge Coded:
Each bit has a fixed duration; data is represented by the edge’s change of direction in the middle of the bit-period, i.e., high-to-low vs. low-to-high. For example:

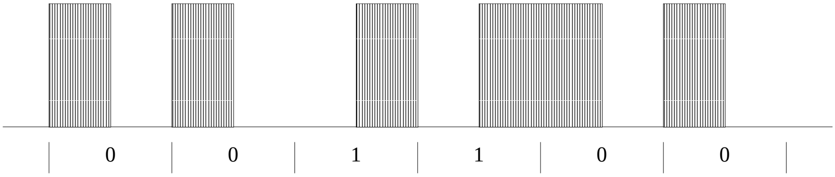


Figure 7-21 Edge Coding Example

This coding method is also known as bi-phase encoding, because each bit-period is broken into two pieces (called “chips”), and a ‘0’ or ‘1’ is conveyed by the chip that has the pulse in it.

Carrier Waveforms:

The most popular frequency range for the carrier signal is from 36 to 40 kHz. However, some commercial devices use carriers in the 450 to 500 kHz range. The Xilleon 220 UIRT has a completely programmable frequency range, from DC up to about 12 MHz.

Quote from the Standard of Electronic Industries Association of Japan for *Sub-carrier frequency allocation for infrared data transmission* (EIAJ CP-1205, Jan. 1993):

(1) Band I (33-40kHz): This band is for remote controller that Association for Electric Home Appliances defines. The band of the remote controller using this frequency has spread most widely in present. This frequency was proposed by IEC and 3-10kHz and 440-480kHz are also proposed by IEC. This is based on the remote controller used in Europe now.

The duty cycle of the carrier typically has a high-time in the 25% to 33% range; as opposed to 50%, presumably to minimize power consumption. Again, in the Xilleon 220 UIRT, this duty cycle is widely programmable, in increments of 20.83ns.

The “Repeat Packet”:

While a key is held depressed on a remote control device, the transmitter usually sends a packet at uniform intervals. There are three main methods that an IR protocol uses to convey the “repeat key” information:

1. Sends a “special” packet that only has a header and one or two code bits.
2. Sends the same packet again.
3. Sends another packet which varies from the initial packet in just a single bit (or two).

7.6 Peripheral DMA Unit

7.6.1 Overview

The DMA Unit (DMAU) of the Peripheral Controller Unit (PCU) has eight general-purpose channels that can be flexibly assigned to service the peripheral modules that are inside the PCU; namely, the two UARTs, the FlexBus Controller, and the UIRT (Infrared Receiver/Transmitter).

The DMA Unit contains eight identical, independent channels. From the programmer's perspective, a DMA channel can be used to move a block of linear-addressed data elements to/from memory from/to a peripheral device. The channel is set-up with an address to memory-space, an address to peripheral-space, a count for how many elements to transfer, and a direction (memory-to-peripheral, or peripheral-to-memory) for the transfer. Also, the programmer can control whether the peripheral address auto-increments or not with each transfer to support peripheral devices that are accessed through a single "port address". A "Transfer Size" control field also specifies the size of the data element to be moved to/from the peripheral for each access. For example, the various peripheral controllers are 8-bits, 16-bits, or 32-bits.

The DMA Unit also has the ability to generate an interrupt to the host when a DMA task has completed.

7.6.2 Feature List

- Eight identical, independent channels.
- Each channel can move data either from memory-to-peripheral or from peripheral-to-memory.
- Round-robin scheduling of the channels, with programmable Burst Size per-channel.
- Task-queuing structure supports double-buffering in memory.
- Can generate interrupt upon DMA task completion and/or upon error condition.
- Flexible control of DMA tasks. Tasks can be paused, resumed, saved, restored, and aborted.

7.6.3 Functional Description

7.6.3.1 Block Diagram

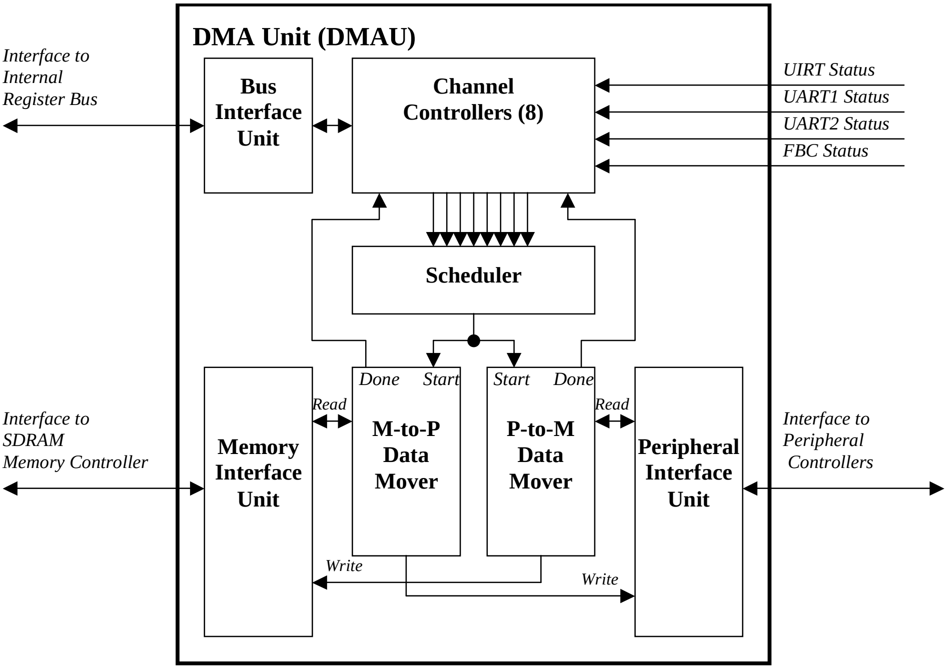


Figure 7-22 Block Diagram of the DMAU

7.6.3.2 Theory of Operation

The *Bus Interface Unit* accepts host read/write accesses to the registers in the DMA Unit. It also contains the interrupt-related circuitry.

There are eight copies of a *Channel Controller* module. Each Channel Controller contains the context for the two tasks that may be present in each channel (in the queue and in the “engine”). The Channel Controller’s job is to generate commands to one of the Data Movers. The command tells the DataMover to move a **burst** of data of a specified size between the memory and the peripheral. The channel controller samples the status from the peripheral devices, and when the peripheral has enough FIFO elements full (or empty), then the channel submits the command to the scheduler. After the DataMover has replied that the burst has been completed, then the channel controller goes back to sampling the peripheral status, waiting for it to become ready for service again.

The *Scheduler* contains a round-robin arbiter that decides which channel’s command is the next one to be presented to the Data Movers.

The *M-to-P Data Mover* contains a 2x128 read buffer for fetching data from memory. It requests 16-bytes per-transaction when reading from the memory. After the read buffer, there is a 32-bit data rotator that handles any mis-alignment between the reading address and the writing address.

The *P-to-M Data Mover* contains a 2x32 read buffer for reading data from the peripheral. This allows for two outstanding read requests to the peripheral to help reduce the latency of reading. After the read buffer, there is a 32-bit data rotator that handles any mis-alignment between the reading address and the writing address.

The *Memory Interface Unit* contains a simple read/write arbiter for scheduling requests from each of the two Data Movers. Writes always have priority over reads.

The *Peripheral Interface Unit* contains a simple read/write arbiter for scheduling requests from each of the two Data Movers. Reads always have priority over writes.

7.6.3.3 Queuing Model for DMA Task Management

DMA operations can be thought of as *tasks* that execute on the DMA Unit.

A *task* comprises two parts: its *context* and its *run-state*.

The *context* of the task is comprised of the values that are in the registers that control the task's operation, for example, a memory address register, a peripheral address register, a byte-count, and other control fields.

The *run-state* of the task indicates the state that the task is in. Possible states are:

- Not-Ready-to-Run
- Ready-to-Run
- Active, but blocked waiting on peripheral
- Active, and transferring data
- Paused

Each channel can hold up to two tasks. One task's context resides in the engine, and the other task's context is in the queue, "in-front of" the engine. See *Figure 7-23,"Task Context and Run-State Flow," on page 46* for a picture of a task's context and run-state. A given task moves among the five (5) states. If the task is in the queue, it is in either the Ready-to-Run or Not-Ready-to-Run state. If the task is in the engine, it is in one of the other three states.

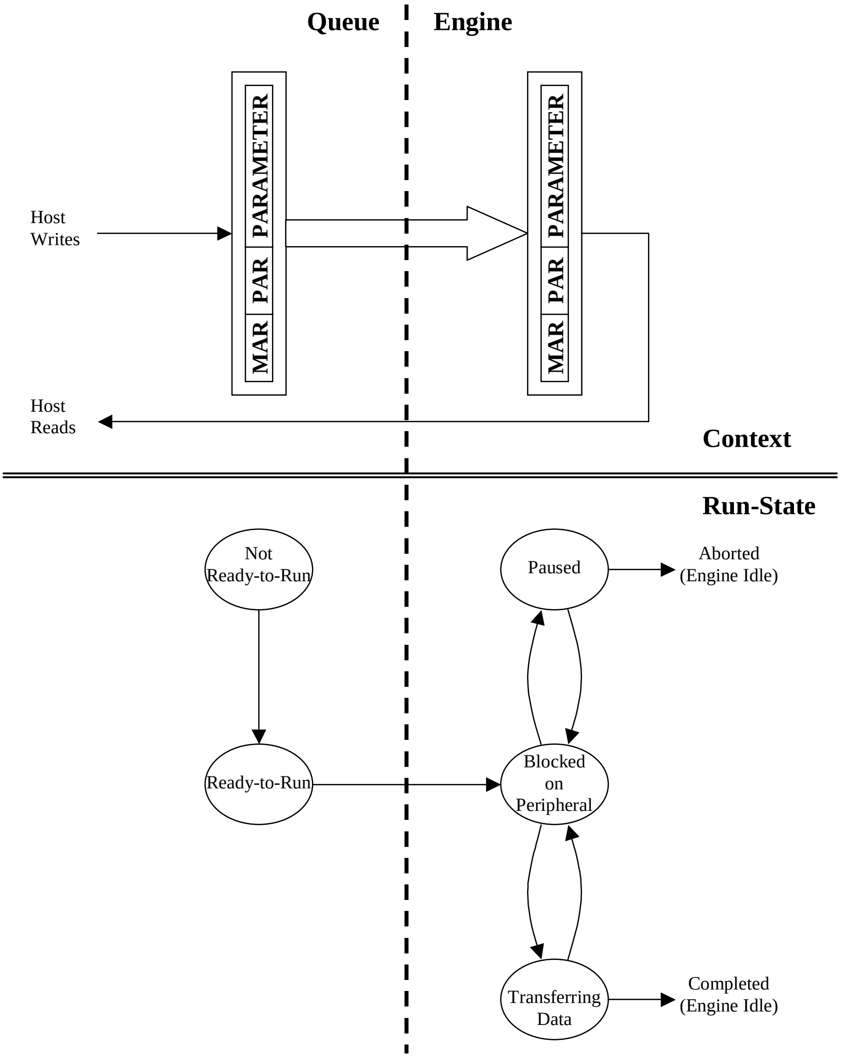


Figure 7-23 **Task Context and Run-State Flow**

If the engine is idle, host writes to registers flow freely through the queue and are shadowed into the engine's internal context registers. Host reads of registers obtain the contents of those engine registers. The queue is only one-deep, and holds the contents of one copy of each of the MAR, PAR, and PARAMETER registers. Note that the CNTLSTAT register does **not** reside in the queue; instead it controls, among other things, how the queue operates.

The table below shows how actions from the programmer can affect how the engine behaves:

Table 7-11 Table of Events and Actions

Task In Queue QUEUE_AVAIL	Task In Engine RUN_STATE	Event	Effect/Action

Ready-to-Run	Idle	No event necessary	ISSUE_START;
			ISSUE_LOAD(if AUTOLOAD_EN='1')

Not Ready-to-Run	Idle	WEN_CONTEXT	Write register, ISSUE_LOAD (AUTOLOAD_EN has no effect here)
Not Ready-to-Run	Active-Xferring	WEN_CONTEXT	Write register
Not Ready-to-Run	Active-Blocked	WEN_CONTEXT	Write register
Not Ready-to-Run	Paused	WEN_CONTEXT	Write register
Ready-to-Run	Active-Xferring	WEN_CONTEXT	Ignored
Ready-to-Run	Active-Blocked	WEN_CONTEXT	Ignored
Ready-to-Run	Paused	WEN_CONTEXT	Ignored
Not Ready-to-Run	Idle	TRIGGER	Task in queue goes to R2R (QUEUE_AVAIL goes Off)
Not Ready-to-Run	Active-Xferring	TRIGGER	Task in queue goes to R2R (QUEUE_AVAIL goes Off)
Not Ready-to-Run	Active-Blocked	TRIGGER	Task in queue goes to R2R (QUEUE_AVAIL goes Off)
Not Ready-to-Run	Paused	TRIGGER	Task in queue goes to R2R (QUEUE_AVAIL goes Off)
Ready-to-Run	Active-Xferring	TRIGGER	Ignored
Ready-to-Run	Active-Blocked	TRIGGER	Ignored
Ready-to-Run	Paused	TRIGGER	Ignored
Not Ready-to-Run	Idle	QUE_LOAD	ISSUE_LOAD
Not Ready-to-Run	Active-Xferring	QUE_LOAD	None
Not Ready-to-Run	Active-Blocked	QUE_LOAD	None
Not Ready-to-Run	Paused	QUE_LOAD	None
Ready-to-Run	Active-Xferring	QUE_LOAD	None
Ready-to-Run	Active-Blocked	QUE_LOAD	None
Ready-to-Run	Paused	QUE_LOAD	None
Not Ready-to-Run	Idle	ABORT	None
Not Ready-to-Run	Active-Xferring	ABORT	None
Not Ready-to-Run	Active-Blocked	ABORT	None
Not Ready-to-Run	Paused	ABORT	ISSUE_ABORT (Engine becomes Idle)
Ready-to-Run	Active-Xferring	ABORT	None
Ready-to-Run	Active-Blocked	ABORT	None
Ready-to-Run	Paused	ABORT	ISSUE_ABORT (Engine becomes Idle)

Events:

WEN_CONTEXT:	Write to a context register (MAR, PAR, PARAMETER)
TRIGGER:	Write to PARAMETER register (while TRIGGER_ENABLE='1')
QUE_LOAD:	Write to CNTLSTAT register, with OP_CNTL='1'
ABORT:	Write to CNTLSTAT register, with OP_CNTL='2'

Actions:

ISSUE_LOAD:	Copies the contents of the registers in the queue into the engine's registers.
ISSUE_START:	Starts the engine. Moves it out of the Idle state. If RUN_ENABLE=1, the engine goes active; otherwise the engine Pauses.
ISSUE_ABORT:	Engine is forced into the Idle state.

The task context moves from the queue into the engine upon any of the following conditions:

- Write to one of the MAR, PAR, or PARAMETER registers while the engine is Idle.
- A task is in the Ready-to-Run state in the queue, and the engine becomes Idle (and the AUTOLOAD_ENABLE bit of the CNTLSTAT register is a ‘1’).
- Write a ‘1’ to the OP_CNTL field of the CNTLSTAT register while the engine is Idle.

A write to the PARAMETER register (when the TRIGGER_ENABLE bit of the CNTLSTAT register is a ‘1’) moves the task in the queue from the Not Ready-to-Run state into the Ready-to-Run state. If the engine is idle, the task will immediately proceed into the Active/Blocked state inside the engine. If the engine is not idle, the task remains in the queue (in the Ready-to-Run state) until the engine becomes idle. While a queued task is in the Ready-to-Run state, all writes to the MAR, PAR, and PARAMETER registers for that channel are ignored. It is the programmer’s responsibility to check the status of the queue before writing to it (using the QUEUE_AVAIL bit of the CNTLSTAT register).

If the user sets the RUN_ENABLE bit of the CNTLSTAT register to a ‘0’, the engine will move to the Paused state as soon as it gets into the Active/Blocked state. It will stay in the Paused state until either:

- The RUN_ENABLE bit is set to ‘1’, OR
- The OP_CNTL field of the CNTLSTAT register is written with a ‘2’, which issues an ABORT_CURRENT command to the engine.

All channels of the DMA Unit are identical. The PERIPH_ID field of a task’s context tells the channel which peripheral it is serving. This allows any channel to be assigned to service any peripheral. It is up to the programmer to make sure that no more than one channel is assigned to each direction for the devices that have transmit/receive FIFOs; such as the UARTs and the UIRT. Multiple channels can be assigned to transmission/reception on behalf of the FlexBus.

7.6.3.4 Example Code for Executing a DMA Task

For “normal” operation, the DMA channel’s CNTLSTAT register should be set up as:

```
CNTLSTAT . OP_CNTL = 00
CNTLSTAT . TRIGGER_ENABLE = 1
CNTLSTAT . AUTOLOAD_ENABLE = 1
CNTLSTAT . RUN_ENABLE = 1
```

The Memory Address Register (MAR) and Peripheral Address Register (PAR) should be initialized to point to the appropriate place for the DMA transfer.

Finally, a write to the PARAMETER register will trigger the DMA Unit to execute the DMA task. The TASK_SIZE field of the register is set to the total number of bytes that the user wishes to transfer. The table below offers some guidelines for programming the other fields of this register:

Table 7-12 Guideline Values for Fields of PARAMETER Register

Type of Task	PERIPH_ID	DIRECTION	PAR_AIC	XFER_SIZE	BURST_SIZE
UART Transmit	1 or 2	0	1	0 (1-byte)	1/2 the TxFIFO depth
UART Receive	1 or 2	1	1	0 (1-byte)	1/4 the RxFIFO depth
UIRT Transmit	0	0	1	2 (4-byte)	1/2 the TxFIFO depth
UIRT Receive	0	1	1	1 (2-byte)	**Must be 1**
FlexBus Write	3	0	1 or 0 (depends on external device)	1 or 0 (depends on external device)	A value that optimizes memory efficiency, e.g. 16-bytes.
FlexBus Read	3	1	1 or 0 (depends on external device)	1 or 0 (depends on external device)	A value that optimizes memory efficiency, e.g. 16-bytes.

7.6.3.5 Pausing and Resuming a DMA Task

A DMA task which is currently active in the engine can be paused by setting the RUN_ENABLE bit of the CNTLSTAT register to ‘0’. Note that the engine will not **immediately** pause; rather, it will finish any burst that it may be currently processing, and move to a state where it can cleanly pause. For this reason, it is necessary for the programmer to poll the RUN_STATE field of the CNTLSTAT register in order to determine when the channel is indeed paused. Once paused, one can simply set the RUN_ENABLE bit back to ‘1’ to resume this task.

7.6.3.6 Aborting a DMA Task

The first step to aborting a DMA task is to pause it. This procedure is outlined in the preceding section (*“Pausing and Resuming a DMA Task” on page 7-49*).

Once a DMA task has been paused, the user can abort it by writing a ‘2’ to the OP_CNTL field of the CNTLSTAT register.

7.6.3.7 Pausing, Saving, Restoring, and Resuming DMA Tasks

Once a DMA task has been paused, the user can read the contents of the MAR, PAR, and PARAMETER registers and save them into local memory. These registers comprise the context of the current task. Then the task can be aborted by writing a ‘2’ to the OP_CNTL field of the CNTLSTAT register.

If there is a task in the Ready-to-Run state in the queue, and the AUTOLOAD_ENABLE bit is a ‘1’, then the RUN_STATE should eventually report as “Paused” (since the RUN_ENABLE bit is still ‘0’). This task can then be aborted using the same procedure.

In order to load the context of a previously saved task, simply write to the MAR and PAR registers. Then, if the TRIGGER_ENABLE bit is ‘1’, the write to the PARAMETER register will resume that previously saved task.

7.6.3.8 Interrupts

For each channel, there are two types of DMA_INTR_STATUS bits, one for “End-Of-Buffer” (EOB), and one for “Error” (ERR). These bits are read-only status bits.

The EOB status bit gets set to ‘1’ by the hardware when a DMA task **completes**.

The ERR status bit gets set to ‘1’ by the hardware when a DMA task **completes and the peripheral serviced by that task indicates that it had some kind of error**. Examples of peripheral errors are:

For FBC:	None.
For UIRT:	None.
For UARTs:	For transmission, none. For reception, the “Receiver Line Status” types of errors are the ones that are reported to the DMA Unit, namely, break detection, framing error, parity error, or receiver buffer overrun error.

The EOB and ERR status bits are cleared by a programmed write of a ‘1’ to the corresponding bit of the DMA_INTR_CNTL register. Writing a ‘0’ does nothing.

The DMA_INTR_ENABLE register is a read/write register. The hardware performs a bit-wise AND of this register with the contents of the DMA_INTR_STATUS register, and if **any** bits of the result are ‘1’, then the DMA hardware will assert the interrupt output signal.

7.6.4 External Chip Interfaces

The DMA Unit has no external chip interfaces.

7.7 Hardware Controlled I^{2}C Serial Ports

7.7.1 Overview

The I^{2}C bus in Xilleon 220 is designed as a low cost, low speed serial bus. The bus interface requires only two pins: SCL (clock) and SDA (data). SCL can clock up to a maximum speed of 100khz. The I^{2}C bus controller in Xilleon 220 is implemented as a master; its primary function is to transmit or receive bits via the bus interface. There are two identical I^{2}C master controllers in Xilleon 220, namely, I2C0 and I2C1. Each one has its own set of dedicated pins. This section will explain the features and functionality of I2C0.

7.7.2 Feature List

- Transmit or receive bits.
- Generate I^{2}C protocol.
- Noise immunity.
- Variable address bits generation.
- Status bits.
- SCL slowing.
- ACK stretching.

7.7.3 Functional Descriptions

7.7.3.1 Transmit or Receive

Register reads and writes in I^{2}C are performed by serially moving bits over the I^{2}C bus interface. For each register read or write, 8 bits are moved serially across the bus followed by an acknowledge phase. The acknowledge phase is only one SCL period long.

7.7.3.2 I^{2}C Protocol Generation

Normally, I^{2}C bus operations are controlled mainly by software bit-banging at the pins. However, this procedure ties up a fair amount of CPU resources. Therefore, rather than relying on software to generate the I^{2}C protocol, hardware is designed instead to perform this function.

7.7.3.3 Noise Immunity

Noise is a major issue with the I^{2}C bus. It can easily corrupt any data transmission if nothing is done to avoid it. In Xilleon 220, there are three solutions which are applied to reduce noise. The first solution is to apply a digital low-pass filter by super-sampling the inbound bits. The second solution is to actively drive the SDA pin during its rise. The third is to actively drive SCL during its rise.

7.7.3.4 Address and Data Buffer

There is 32 bytes worth of buffer space for storing I^{2}C addresses and data. Addresses and data are pooled together into this one buffer, which means that the total number of address and data bytes cannot exceed 32 bytes. Address bytes must be the first to go into buffer.

7.7.3.5 Status Bits

There are three status bits in the Xilleon 220 I^{2}C controller: DONE, NACK, HALT. These status bits are used to synchronize the I^{2}C driver with the controller. The DONE bit indicates that I^{2}C transfer is done. NACK indicates that there was a problem with the data transfer since no ACK has been received from the I^{2}C slave device. HALT indicates that the I^{2}C slave device is not responding.

7.7.3.6 SCL Slowing

SCL is primarily driven by the I^{2}C master, but in some situations, an I^{2}C slave device may wish to slow the transfer rate. To do so, the device will pull the SCL line low, stopping any transaction until it is ready to proceed, after which it will release the SCL line.

7.7.3.7 ACK Stretching

For some I^{2}C slave devices, the turnaround time between bytes may be longer than a normal SCL period. To address this situation, the ACK phase is stretched. This applies to both receiving and transmitting mode.

7.7.4 Block Diagram of I^{2}C Master

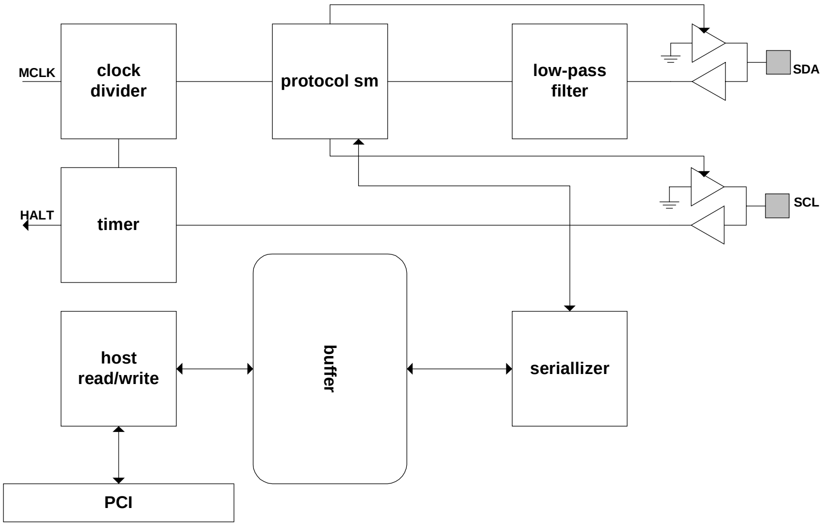


Figure 7-24 I^{2}C Master Block Diagram

Figure 7-24,"I2C Master Block Diagram," on page 53 shows the connectivity between each sub-block of the top-level view of the I^{2}C master controller. The "clock divider" generates SCL from MCLK. The "protocol sm" is responsible for generating the I^{2}C bus protocol. The "low-pass filter" filters out the in-coming SDA bit. The "timer" checks the state of SCL continuously to see if it is being pulled down by the I^{2}C slave device. The "buffer" contains 32 bytes for storing addresses and data. The "serialliser" serializes the byte data into bits to be transmitted and can also parallelize the in-coming bit stream into bytes. The "host read/write" reads or writes bytes into the buffer.

7.7.5 Theory of Operation

7.7.5.1 Boot-up States

The following table shows the states of the respective signals after boot-up:

Table 7-13 Signal States After Boot-up

Signal name	State after boot-up
SCL	Tri-state high
SDA	Tri-state high

The two I^2C bus interface signals are tri-state high when no activity occurs. Each is pulled up by an external pull-up resistor. The resistor value must strike a balance between what I^2C slave devices can drive and the current it can draw while pulling up. A weak pull-up may allow noise to get injected easily into the rising phase of the signal.

7.7.5.2 Initializations

A few of the register fields need to be initialized before performing any I^2C transactions. Register fields that need to initialized are as follows:

Table 7-14 Register Fields that Require Initialization

Register fields	Value	Comments
I2C_CNTL_0_0(I2C_DRIVE_EN)	1	Actively driving the SDA line during its low-high transition
I2C_CNTL_0_0(I2C_DRIVE_SEL)	1	Drive SDA pin for 10 MCLK/s. note MCLK is an internal clock which is roughly 100 MHz
I2C_CNTL_0_0(I2C_PRESCALE)	0x1615	Assuming MCLK is 100MHz, this value yields a SCL clock speed of ~100khz. I2C_PRESCALE can be broken up into two equal parts of 8bit each. Let M = 0x16 and N = 0x15 and C is constant which is 4 SCLperiod = C * M * N * MCLKperiod SCLperiod = 4 * 0x16 * 0x15 * (1/100MHz) M not only determines the SCL frequency, it also determines how many samples of in-coming SDA bit are taken per SCL clock.
I2C_CNTL_1_0(I2C_ACK_COUNT)	0	Does not stretch ACK phase
I2C_CNTL_1_0(I2C_TIME_LIMIT)	0x10	This field defines maximum wait-time, that is if I^2C slave device has SCL line pull-down for period longer than the wait-time, a time-out flag will be issued. Wait-time = 64 * 0x10 * SCLperiod

7.7.5.3 I^2C Bus Protocols

All basic protocols required by standard I^2C specs are implemented in Xilleon 220. Protocols such as *STOP*, *START*, no *STOP*, no *START*, *RESTART*. *SEND* or *RECEIVE* are automated in hardware. Variable address formats are also supported.

START protocol

Every time a new transaction is made with a START protocol, an address must first be supplied to the I^2C slave. Invoking a START protocol can have different implications, for example, addressing a new I^2C slave, or having change in direction (e.g. switching from transmit to receive), or having a change in address format. If a START protocol is not invoked, no address needs to be sent - the I^2C transaction is merely a continuation of the previous transaction which did not invoke the STOP protocol.

STOP protocol
The STOP protocol interpretation is simple. If its invoked, communication with the I^2C slave is terminated; otherwise, it will carry on to the next transaction.

Re-START protocol
Occasionally, the I^2C driver may want to change the address even though no STOP protocol has been invoked in the last transaction. In this case, Re-START is invoked.

7.7.5.4 I^2C Master Transmit Mode

Depending on whether the 7-bit or 10-bit address format is used, either one or two address bytes are first loaded into the buffer through the data port I2C_DATA_0(I2C_DATA). Afterwards, data bytes are loaded into the same buffer through the same register port I2C_DATA_0(I2C_DATA). Since the buffer size is only 32 bytes deep, the I^2C driver must ensure that the sum of address and data bytes does not exceed 32. Receive or transmit information is embedded in the address which the I^2C slave will decode appropriately.

Table 7-15 shows the register fields that are used to configure an I^2C master to use 7-bit address format, transmit 10 bytes of data, and invoke START and STOP protocols.

Table 7-15 Register Fields for Configuring I^2C Master to Transmit

Register fields	Value	Comments
I2C_CNTL_0_1(I2C_DATA_COUNT)	0xa	Data count
I2C_CNTL_0_1(I2C_ADDR_COUNT)	1	Address count
I2C_CNTL_0_0(I2C_START)	1	Transfer START protocol
I2C_CNTL_0_0(I2C_STOP)	1	Transfer STOP protocol
I2C_CNTL_0_0(I2C_RECEIVE)	0	This corresponds to transmit mode embedded in the address byte/s
I2C_CNTL_0_0(I2C_GO)	1	This is a trigger which starts I^2C transaction

Note: The last register programmed must be I2C_CNTL_0, as shown in *Table 7-15*. This is necessary since the bit trigger I2C_GO is a part of the register.

Upon triggering, the I^2C master controller will begin to fetch data bytes from the buffer and serialize out onto the I^2C bus. When the specified number of bytes have been transmitted, the transaction will stop and the status DONE will be flagged. The I^2C driver can either poll this bit or use interrupt to check when a job is finished.

7.7.5.5 I^2C Master Receive Mode

As in the case of the I^2C master transmit, depending on whether the 7-bit or 10-bit address format is used, either one or two address bytes are first loaded into the buffer through the data port I2C_DATA_0(I2C_DATA). Since the I^2C master is in receive mode, data bytes do not need to be loaded into the buffer. The maximum number of bytes that can be fetched from the I^2C slave in one transaction is 32 bytes. This is the full buffer size. The address bytes that have been loaded into the buffer will be replaced by data coming in from the I^{2}C slave.

Table 7-16 shows the register fields used to configure the I^{2}C master to use 7-bit address format, fetch 10 bytes of data from I^{2}C slave and invoke START and STOP.

Table 7-16 Register Fields for Configuring I^{2}C Master to Receive

Register fields	value	comments
I2C_CNTL_0_1(I2C_DATA_COUNT)	0xa	Data count
I2C_CNTL_0_1(I2C_ADDR_COUNT)	1	Address count
I2C_CNTL_0_0(I2C_START)	1	Transfer START protocol
I2C_CNTL_0_0(I2C_STOP)	1	Transfer STOP protocol
I2C_CNTL_0_0(I2C_RECEIVE)	1	This corresponds to transmit mode embedded in the address byte/s
I2C_CNTL_0_0(I2C_GO)	1	This is a trigger which starts I^{2}C transaction

Note: As with the master transmit, the last register programmed must be I2C_CNTL_0. This is necessary since the bit trigger I2C_GO is a part of the register.

The I^{2}C master controller will keep fetching and storing the bytes into the buffer. When all bytes specified have been transferred from the I^{2}C slave, the status DONE will be flagged. The I^{2}C driver can either poll this bit or use interrupt to check when a job is finished.

7.7.5.6 Host Read and Write Pointers

Similar to the case with the master transmit mode, every time a write is performed on the register port I2C_DATA_0(I2C_DATA), an internal write pointer to the buffer is incremented by one. Also, similar to the case with the master receive mode, every time a read is performed on port I2C_DATA_0(I2C_DATA), an internal read pointer is incremented by one. Therefore, before any transaction is performed, these two pointers must be initialized to ‘0’ by writing to register field I2C_CNTL_0_1(I2C_SOFT_RST).

7.7.5.7 Interrupts

The I^{2}C master controller supports interrupt. Interrupt is asserted when any of the status bits such as DONE, NACK or HALT is asserted. To enable interrupt, set the following fields as shown in *Table 7-17*. Note that these settings are used only with an external CPU in place (peer mode).

Table 7-17 Register Fields for Enabling Interrupt

Register fields	Value
PCU_INT_CNTL(I2C0_INT_EN)	1
GEN_INT_CNTL(PCU_INT_EN)	1
GEN_INT_CNTLS(CHIP_INT_EN)	1

7.7.6 Data Flow

7.7.6.1 Master Transmit Data Flow

In this mode, data bytes are pushed into the buffer through the PCI interface and the sub-block "host read/write". From the buffer, data is serialized and sent to "protocol sm", which then puts the data bit on the I^{2}C bus interface.

7.7.6.2 Master Receive Data Flow

In master receive mode, data flows in the opposite direction. It first reaches the "low-pass filter" and then passes through the "protocol sm". Bits are then packed into bytes in the "serializer". The bytes are stored in the buffer, and when the job is completed, the bytes are read out from the buffer through the "host read/write" block and PCI interface.

7.7.6.3 Jobs Submission

Only one job can be submitted at any one time. The job submitted must be either transmit or receive, not both. Note that every time a change is made from transmit to receive or vice-versa, a new address has to be loaded. This is necessary in order to indicate a direction change.

7.8 VIP Host Port

7.8.1 Overview

The VIP host port controller in Xilleon 220 is VIP1.0 compliant. It supports 1X type host bus interface, which requires only four pins: VIPCLK, HCTL (control) and HAD(1:0) (data). The specified frequency range of VIPCLK is 25 MHz – 33 MHz. VIPCLK is derived from the internal XCLK. The main purpose of the VIP host port interface is to communicate with an external VIP device, e.g. an MPEG audio decoder. It can support up to four VIP devices.

The interface supports two DMA channels, one that routes data from system or frame buffer to VIP device, and the other that routes data from VIP device to system or frame buffer. Each DMA channel is aided by the stream-interface (SI), which is responsible for moving DMA data to/from the system or frame.

7.8.2 Feature List

- Register access to the VIP device(s).
- Two opposite unidirectional DMA channels.
- Hardware polling.
- Register and DMA interleaving.
- Device_id detection.

7.8.3 Functional Descriptions

7.8.3.1 VIP Device Register Access

Register reads and writes can be performed through VIP host interface. Each read or write can be one byte, two bytes or four bytes. The register write operation involves writing register data to the data-port only. Register read is not as simple, however. Data must first be pre-fetched from the VIP device; then, a second read is performed to read the pre-fetched data.

7.8.3.2 DMA Channels

There are two DMA channels present in Xilleon 220; each one is unidirectional, and they route data in opposite directions. One goes from system or frame to VIP device via stream-interface and the other goes from VIP device to system or frame via stream-interface. Arbitration between the two channels is based on the round-robin scheme. For each DMA channel, there is a 4-dword buffer to absorb the data latency incurred at the VIP device or stream-interface.

7.8.3.3 Hardware Polling

During DMA reads, the VIP device read FIFO may not have any remaining data. Alternatively, during DMA writes, the slave device write FIFO may not have enough space for another byte. In these cases, the DMA transfer will be stopped and a retry will have to be performed at a later time. Here, hardware polling comes into action. The DMA controller will continuously poll different FIFO ports until data transfer is completed (recommended by VIP1.0 specs).

7.8.3.4 Register and DMA Interleaving

Since DMA transfer is usually very long and uninterrupted, there is a mechanism provided in Xilleon 220 which allows register access without terminating the DMA transfer. When a register access is requested, the DMA transfer is first temporarily suspended, and the register operation is passed on. Once the register access is completed, the DMA transfer will resume moving data across the VIP bus. This interleaving scheme reduces a lot of software overhead and complexity.

7.8.3.5 Device_id Detection

Upon booting up Xilleon 220 chip, the VIP driver may need to find out whether any devices are present on board. Since the maximum number of VIP devices supported is 4, there will be a for-loop used to poll each device_id. A signature will be returned if any VIP device exists; otherwise a time-out status will be issued.

7.8.4 VIP Host Controller Schematic Diagram

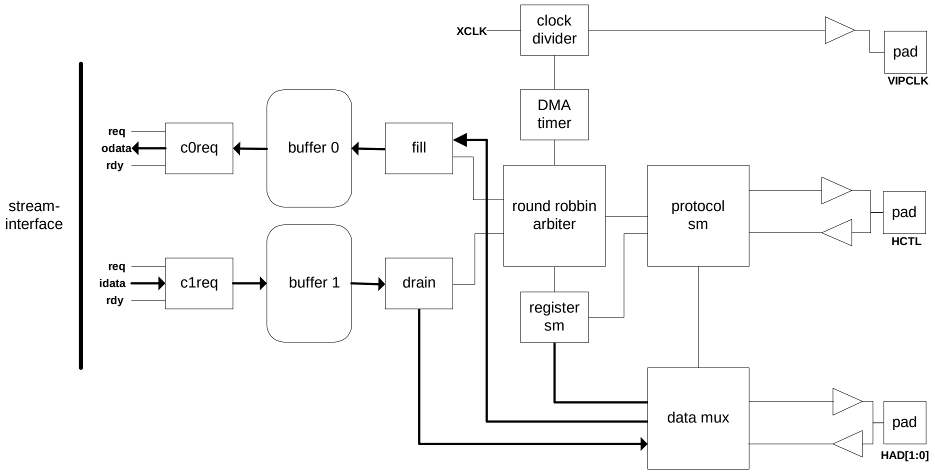


Figure 7-25 VIP Host Controller Schematic Diagram

Figure 7-4,"Overview of FlexBus Operation," on page 8 shows the top-level view and connectivity of the VIP host port control block. The "clock divider" generates the VIPCLK which is used to drive external VIP device(s). The divider value used must be larger than '1'. Two DMA channels are shown in the diagram. "buffer0" and "buffer1" are used as temporary data storages during DMA transfers. The "DMA timer" is used to allocate time for the two channels in a round-robin fashion. Whenever register access is requested, "register sm" will issue a request to "round robin arbiter", which will then take action to suspend any DMA transfer so that register access can occur.

7.8.5 Theory of Operation

7.8.5.1 Boot-up States

The following table shows the states of the respective signals after boot-up:

Table 7-18 Signal States After Boot-up

Signal name	State after boot-up
VIPCLK	low
HCTL	low
HAD(0)	Tri-state high
HAD(1)	Tri-state high

Before calling any VIP functions, a VIP driver must first check if any VIP device exists. It should check all four device_ids from 00 to 11. Devices that are present on board will respond with its unique signature. Otherwise, if there is no device present for a particular device_id, a time-out will be flagged in register bit VIPH_TIMEOUT_STAT(VIPH_REG_STAT). The time-out bit must be cleared by writing a ‘1’ to register field VIPH_TIMEOUT_STAT(VIPH_REG_AK). Prior to performing any device detection, ensure that the register field VIPH_CONTROL(VIPH_MAX_WAIT) is set to a non-zero value. Otherwise, no time-out will occur and the system will hang if detection fails.

7.8.5.2 VIP Device Register Access

Registers in slave devices can be accessed through the VIP host bus interface.

VIP Register Writes:
Prior to starting VIP register writes, register VIPH_REG_ADDR must be programmed with the appropriate slave register address and the command data. Command data consists of information specifying register or DMA transfer, read or write and device_id number. The next step will be to write the register data into register port VIPH_REG_DATA. Xilleon 220 supports byte, word or dword VIP register writes.

VIP Register Reads:
Similar to VIP register writes, VIPH_REG_ADDR must be pre-programmed with the appropriate slave register address and command information. Slave register read is broken up into two steps. The first step is data pre-fetch and the second step is reading the pre-fetched data. The reason for there being two steps is to prevent the host (CPU or MIPS) from waiting for the VIP device to finish the transfer and also in some cases to prevent system hang (e.g. if the device times out during device detection).

After having setup the register VIPH_REG_ADDR, perform a read on register port VIPH_REG_DATA. This is when data pre-fetch is performed. Check the register field VIPH_CONTROL(VIPH_REG_RDY) to ensure that pre-fetch is completed. If so, then disable pre-fetch by setting VIPH_TIMEOUT_STAT(VIPH_REGR_DIS) to 1. Perform another read on register port VIPH_REG_DATA to get the pre-fetched data. Turn the pre-fetch on for the next read.

7.8.5.3 DMA Transfers

There are two DMA channels in Xilleon 220. ***Channel0*** is dedicated to VIP device to system or frame transfer (C2S transfer) and ***channel1*** is dedicated to system or frame to VIP device data transfer (S2C transfer).

To use any channel, the DMA table must first be defined. The DMA table contains the following information: the size of the job transfer, the direction of data flow, and the data location as to whether it is in system or frame. Refer to *“The Stream Interface” on page 3-2* for information on DMA tables as well as guidelines for setting up these table in order to initiate DMA transfer on VIP clients. There are two VIP clients from the stream-interface point of view; one corresponds to ***channel0***, the other ***channel1***.

There are a few one-time register setups which could be performed during initialization. *Table 7-19* lists some of these register fields and their corresponding values:

Table 7-19 Register Fields and Corresponding Values

Register fields	Value	Comments
VIPH_CONTROL(VIPH_CLK_SEL)	3	This is assuming XCLK frequency ~ 100 MHz
VIPH_CONTROL(VIPH_CH0_CHUNK)	3	Use this value only
VIPH_CONTROL(VIPH_CH1_CHUNK)	15	Use this value only
VIPH_CONTROLVIPH_EN)	1	Turns on VIP host control block
VIPH_CONTROL(VIPH_VIPCLK_DIS)	0	For power saving only
VIPH_DV_LAT(VIPH_TIME_UNIT)	0xf	Basic time unit upon which VIPH_DV0_LAT and VIPH_DV1_LAT are based.
VIPH_DV_LAT(VIPH_DV0_LAT)	0xa	Uninterrupted time allocated for channel0(only register access can interrupt it). The length of uninterrupted time here is 0xf * 0xa * XCLK period.
VIPH_DV_LAT(VIPH_DV1_LAT)	0xb	Uninterrupted time allocated for channel1(only register access can interrupt it). The length of uninterrupted time here is 0xf * 0xb* XCLK period.
VIPH_CH0_ADDR(VIPH_CH0_AD)	0x76	The value set depends on device id, FIFO port number
VIPH_CH1_ADDR(VIPH_CH1_AD)	0x54	The value set depends on device id, FIFO port number

The length of a DMA job can be either ***finite*** or ***infinite***.

Finite mode

VIP device to system or frame DMA (C2S transfer)

In finite mode, the length of the DMA job is pre-determined and the buffer size is pre-defined in the DMA table. Register field VIPH_C2S_COUNT(VIPH_C2S_COUNT) must be loaded with the buffer size. DMA transfer can then be initiated by setting register bit VIPH_ST_TRIG(VIPH_ST0_START) to ‘1’. Register field VIPH_C2S_COUNT(VIPH_C2S_DONE) will flag a ‘1’ when the job is completed.

System or frame to VIP device DMA (S2C transfer)

Similarly, the length of the DMA job is pre-determined and the buffer size is pre-defined in the DMA table. Register field VIPH_S2C_COUNT(VIPH_S2C_COUNT) must be loaded with the buffer size. DMA transfer can then be initiated by setting register bit VIPH_ST_TRIG(VIPH_ST1_START) to ‘1’. Register field VIPH_S2C_COUNT(VIPH_S2C_DONE) will flag a ‘1’ when the job is completed.

Infinite mode

VIP device to system or frame DMA (C2S transfer)
The length of the DMA job here is not known, but it can be treated as infinite. One of the possible DMA configurations is to link two buffers in a loop, that is defining a circular buffer with a link-list of 2. This way, as far as system or frame buffer is concerned, its memory size is infinite. Register field VIPH_C2S_COUNT(VIPH_C2S_COUNT) should be loaded with a value of 0 to indicate that its transfer is infinite. Upon setting register bit VIPH_ST_TRIG(VIPH_ST0_START) to 1, the job will continue until the VIPH_ST_TRIG(VIPH_ST0_START) is turned off. Register bit VIPH_C2S_COUNT(VIPH_C2S_DONE) will flag a ‘1’ when the job is completed.

System or frame to VIP device DMA (S2C transfer)
Similarly, the length of the DMA job here is not known, but it can be treated as infinite. One of the possible DMA configurations is to link two buffers in a loop, that is, defining a circular buffer with a link-list of 2. This way, as far as the system or frame buffer is concerned, its memory size is infinite. Register field VIPH_S2C_COUNT(VIPH_S2C_COUNT) should be loaded with value of 0 to indicate that its transfer is infinite. Upon setting register bit VIPH_ST_TRIG(VIPH_ST1_START) to 1, the job will continue until the VIPH_ST_TRIG(VIPH_ST1_START) is turned off. Register bit VIPH_C2S_COUNT(VIPH_S2C_DONE) will flag a ‘1’ when the job is completed.

7.8.5.4 Interrupts

There are three sources of interrupt supported in VIP host port. The first interrupt source comes from an external VIP device via a dedicated pin. This interrupt source is enabled through VIPH_TIMEOUT_STAT(VIPH_INTPIN_EN). The other two sources come from DMA transfers. An interrupt will be asserted whenever a DMA transfer is completed. To enable these two interrupt sources, set register bit VIPH_S2C_COUNT(VIPH_S2C_DONE) and VIPH_C2S_COUNT(VIPH_C2S_DONE) to ‘1’.

7.8.5.5 Power Management

To conserve power consumption in the external VIP device, Xilleon 220 can stop feeding the VIPCLK out by disabling the VIPCLK through setting VIPH_CONTROL(VIPH_VIPCLK_DIS) to ‘1”. During power down mode, CPU or MIPS will notify the VIP driver ahead of time. The driver will then take action to stop submitting any new VIP register accesses or DMA jobs and also turn off any FIFO activities in VIP device(s), wait until the VIP host bus is completely idle, and then proceed to turn off the VIPCLK.

7.8.6 Data Flow

In the Xilleon 220 VIP control block, there are essentially three types of data flow: two DMA transfers: *C2S transfer* and *S2C transfer*, and register access through the VIP host interface bus.

7.8.6.1 C2S Transfer

In this transfer mode, data is fetched from the VIP device buffer and sent to “buffer0” via the block “fill”. Whenever there is enough data in “buffer0” for a burst, “c0req” will initiate data transfer to system or memory via stream-interface. Block c0req handles the write request and ready handshaking with the stream-interface. Block “protocol sm” is responsible for generating VIP host bus interface protocol.

7.8.6.2 S2C Transfer

In this transfer mode, data flow occurs in the opposite direction from that of *C2S transfer*. Block “c1req” will initiate a request to stream-interface to read data from system or frame buffer; stream-interface will then respond by sending a burst of data. The data will be stored in the “buffer1”, from which the block “drain” fetches the data and send it across VIP bus interface to VIP device.

7.8.6.3 Register Access

Register access is first initiated by the host (MIPS or CPU) which instructs the block “register sm” to perform a read or write over the VIP host bus interface. Register data will, like C2S or S2C transfers, go through the “protocol sm” block.

7.8.6.4 Relationship between each Transfer Mode

Register access takes the highest priority; it has the ability to preempt any DMA transfer. C2S and S2C transfers have the same priority, but the rate of bus occupancy for each channel can vary depending on what is being programmed into register fields VIPH_DV_LAT(VIPH_DV0_LAT) and VIPH_DV_LAT(VIPH_DV1_LAT).

This page intentionally left blank.

Chapter 8
Memory Controller

8.1 Introduction

The Xilleon 220 memory controller provides the write and read access path to the external SDRAMs (Synchronous DRAM). It arbitrates the requests from internal clients and executes the proper timing cycles with the SDRAM components. It also performs various SDRAM management functions such as initialization and refresh.

The Xilleon 220 memory controller is a dual-channel design with a 32-bit data interface for each channel. Single-Data-Rate (SDR) or Dual-Data-Rate (DDR) SDRAMs are supported using either or both channels. The two channels may be operated independently with separate memory partitions, or in tandem to form a single 64-bit channel (SDR only). The maximum SDRAM clock rate is 166 MHz. SGRAM (Synchronous Graphics DRAM) can also be used.

8.2 Feature List

- Industry Standard SDR SDRAMs
    - 2 or 4 internal banks
    - 3.3 volt LVTTL
    - CAS Latency = 2, 3, or 4
    - Fully compatible with SGRAMs
- Industry Standard DDR SDRAMs
    - 2.5 Volt SSTL-2
    - CAS Latency = 2, 3, or 4
    - With and without internal DLL
    - QS(Strobe)-per-byte and QS-per-32bits
    - Fully compatible with SGRAMs
- Clock frequency up to 166 MHz.
- Up to 128 MBytes total storage capacity
    - 15 address bits per channel provided
    - 2 chip-selects per channel provided
- Interface configurations:
    - Single channel, 32-bit SDR
    - Two independent channels, 32-bit per channel SDR

  - Single channel, 64-bit SDR by operating two channels in tandem
  - Single channel, 32-bit DDR
  - Two independent channels, 32-bit per channel DDR
- Programmable SDRAM timing parameters supports a variety of industry-standard SDRAM components.
- Separate programming register sets for each memory channel so that different memory sizes and types can be used on each channel at the same time.
- Memory allocation scheme provides separate memory partition for each channel. Protected memory region can be defined to prevent access by certain clients.
- Arbitration mechanism to prioritize real-time and non-real-time requests. Context sensitive priority for selected critical clients.
- Power management controller that enables the SDRAMs to be placed into self-refresh operation, and then awakened with memory data contents retained.

8.3 External Memory Support

The following series of tables offers examples of memory configurations supported by the Xilleon 220. Other configurations are possible using SDRAM types not specifically listed. There can also be different amounts of memory on the two channels. A restriction on mixing different memory types is that both channels must operate at the same clock frequency and cannot mix SDR and DDR.

Table 8-1 2 Bank SDR DRAM Chips

Type	# RAMs	FB Size	# Channels	Memory Width	Physical Ranks (chip selects)
SGRAM - 1M x 32	1	4MB	1	32-bit	1
SGRAM - 1M x 32	2	8MB	1	32-bit	2
SGRAM - 1M x 32	2	8MB	2	32-bit	1
SGRAM - 1M x 32	4	16MB	2	32-bit	2
SGRAM - 1M x 32	2	8MB	1	64-bit	1
SGRAM - 1M x 32	4	16MB	1	64-bit	2

Table 8-2 4 Bank SDR DRAM Chips

Type	# RAMs	FB Size	# Channels	Memory Width	Physical Ranks (chip selects)
SDRAM - 4M x 16	2	16MB	1	32-bit	1
SDRAM - 4M x 16	4	32MB	1	32-bit	2
SDRAM - 4M x 16	4	32MB	2	32-bit	1
SDRAM - 4M x 16	8	64MB	2	32-bit	2
SDRAM - 4M x 16	4	32MB	1	64-bit	1
SDRAM - 4M x 16	8	64MB	1	64-bit	2

Table 8-2 4 Bank SDR DRAM Chips (Continued)

Type	# RAMs	FB Size	# Channels	Memory Width	Physical Ranks (chip selects)
SDRAM - 8M x 16	2	32MB	1	32-bit	1
SDRAM - 8M x 16	4	64MB	1	32-bit	2
SDRAM - 8M x 16	4	64MB	2	32-bit	1
SDRAM - 8M x 16	8	128MB	2	32-bit	2
S(D/G)RAM - 2M x 32	1	8MB	1	32-bit	1
S(D/G)RAM - 2M x 32	2	16MB	1	32-bit	2
S(D/G)RAM - 2M x 32	2	16MB	2	32-bit	1
S(D/G)RAM - 2M x 32	4	32MB	2	32-bit	2
S(D/G)RAM - 2M x 32	2	16MB	1	64-bit	1
S(D/G)RAM - 2M x 32	4	32MB	1	64-bit	2
S(D/G)RAM - 4M x 32	1	16MB	1	32-bit	1
S(D/G)RAM - 4M x 32	2	32MB	1	32-bit	2
S(D/G)RAM - 4M x 32	2	32MB	2	32-bit	1
S(D/G)RAM - 4M x 32	4	64MB	2	32-bit	2
S(D/G)RAM - 4M x 32	2	32MB	1	64-bit	1
S(D/G)RAM - 4M x 32	4	64MB	1	64-bit	2

Table 8-3 4 Bank DDR DRAM Chips

Type	# RAMS	FB Size	# Channels	Memory Width	Physical Ranks (chip selects)
SDRAM - 4M x 16	2	16MB	1	32-bit	1
SDRAM - 4M x 16	4	32MB	2	32-bit	1
SDRAM - 4M x 16	4	32MB	1	32-bit	2
SDRAM - 4M x 16	8	64MB	2	32-bit	2
S(D/G)RAM - 2M x 32	1	8MB	1	32-bit	1
S(D/G)RAM - 2M x 32	2	16MB	2	32-bit	1
S(D/G)RAM - 2M x 32	2	16MB	1	32-bit	2
S(D/G)RAM - 2M x 32	4	32MB	2	32-bit	2
SDRAM - 8M x 16	2	32MB	1	32-bit	1
SDRAM - 8M x 16	4	64MB	2	32-bit	1
SDRAM - 8M x 16	4	64MB	1	32-bit	2
SDRAM - 8M x 16	8	128MB	2	32-bit	2
SDRAM - 16M x 16	2	64MB	1	32-bit	1
SDRAM - 16M x 16	4	128MB	2	32-bit	1

Table 8-3 4 Bank DDR DRAM Chips   (Continued)

Type	# RAMS	FB Size	# Channels	Memory Width	Physical Ranks (chip selects)
S(D/G)RAM - 4M x 32	1	16MB	1	32-bit	1
S(D/G)RAM - 4M x 32	2	32MB	2	32-bit	1
S(D/G)RAM - 4M x 32	2	32MB	1	32-bit	2
S(D/G)RAM - 4M x 32	4	64MB	2	32-bit	2

Table 8-4 Chip Type and Speed

Type	Speed
S(G/D)RAM - Single Data Rate	100 - 166 MHz
S(G/D)RAM - Double Data Rate	100 - 166 Mhz

8.4 Functional Description

8.4.1 Block Diagram

The Xilleon 220 memory controller block diagram is presented in *Figure 8-1,“Memory Controller Block Diagram,” on page 8-5*. The diagram shows the major components of the memory controller:

- Client Request Interfaces - provides communication handshake between the memory controller and each client and directs requests to one of the two memory channels. The channel split is based on the memory address.
- Arbiter - Examines pending requests and selects the next winner. There is a separate arbiter for each memory channel.
- Sequencer - State machine that generates the proper signalling protocol with the external SDRAMs. There is a separate sequencer for each memory channel.
- Registers - Mode and configuration settings.
- Initialization & Refresh - Executes the initialization sequence to the SDRAMs after chip reset. Maintains timers for SDRAM refresh to guarantee data retention.
- Power Management - Enables SDRAMs to be placed into a low-power self-refresh mode so that the Xilleon 220 clock may be turned off for power savings.

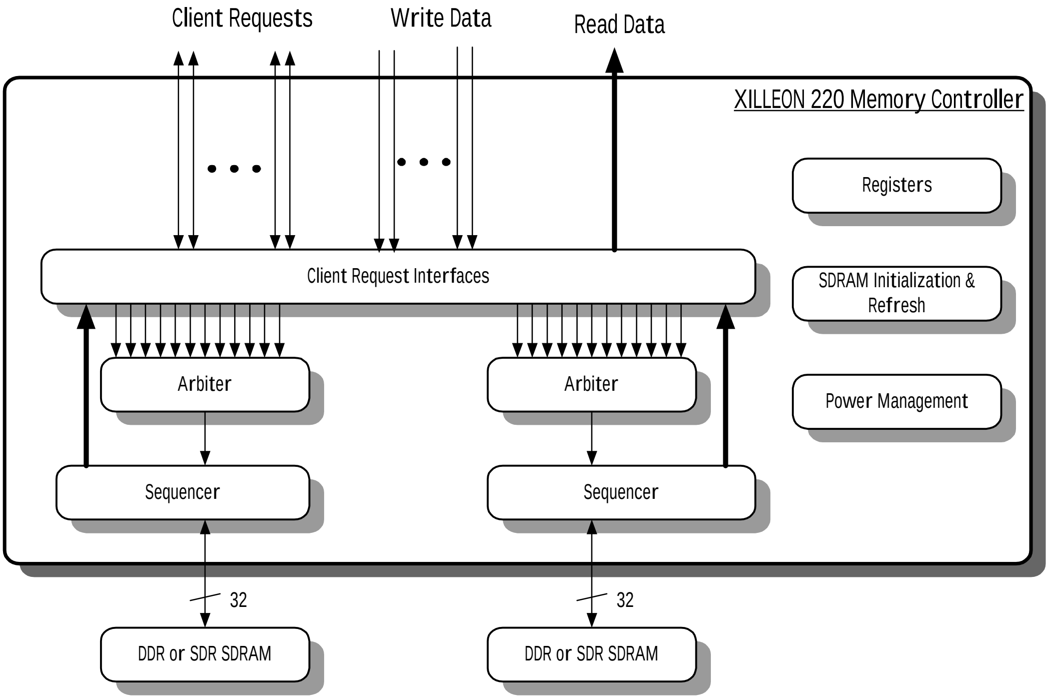


Figure 8-1 Memory Controller Block Diagram

8.4.2 Theory of Operation

8.4.2.1 Operating Modes

There are three fundamental operating modes for the memory controller:

1. Single 32-bit Channel - In this mode, memory channel A is used and memory channel B is idle. May be used in SDR or DDR. All logical memory space is allocated to channel A.
2. Dual 32-bit Channels - In this mode, both memory channels operate independently. May be used in SDR or DDR. Logical memory space is split between the two channels. An advantage of this mode is that processes can be assigned to memory channels and will be able to execute in parallel without competing with each other for memory bandwidth.
3. Single 64-bit Channel - In this mode, the two memory channels operate in tandem. May be used only in SDR. All logical memory space is allocated to channel A. An advantage of this mode is that the memory performance of Single 32-bit DDR is achieved using SDR memories.

8.4.2.2 Memory Address Assignment

The logical memory address space is a single contiguous partition allocated so that all of channel A memory is assigned to the lowest addresses and all of the channel B memory is assigned to the upper addresses. If the amount of memory on channels A and B are equal, then a continuous address space exists with the boundary between channels A & B at the halfway point. See *Figure 8-2,“Memory Channel Address Assignment,” on page 8-6*. Applications can select the memory channel to be used by adjusting the base-address offset when allocating memory. This method provides a mechanism for planning which process will execute on which memory channel in order to minimize competition between high-bandwidth processes. If the storage capacity of the two channels is not equal, then Channel A must have the larger capacity.

There is also a means of defining one protected memory region per memory channel. The graphics engine and host processor clients are then prohibited from accessing those regions. In the event that such an access is attempted, the memory controller will inhibit any writing to the locations or, in the case of a READ, will return blank data. This is intended to deter hackers from accessing sensitive areas of memory.

The graphics engines are limited to 8 MBytes of addressing. The memory controller allows four 8 MByte regions to be defined anywhere in the available memory space through a programming register. Each 8 MByte region is contiguous and can be specified at 1 MByte granularity.

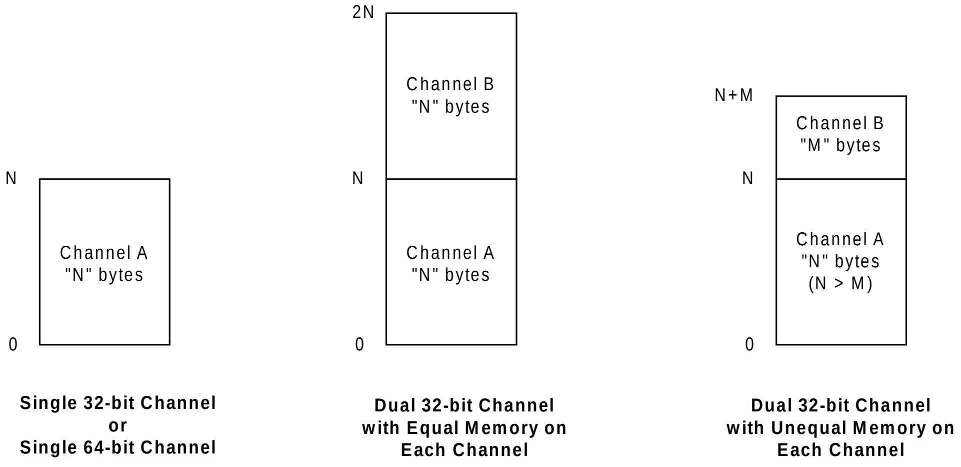


Figure 8-2 Memory Channel Address Assignment

8.4.2.3 Memory Tiling and Access Efficiency

A typical memory access consists of opening a page, or row (RAS-cycle), waiting the requisite number of clocks (t_{RCD}), reading or writing data (CAS-cycle), closing the page (precharge), and waiting the requisite number of clocks(t_{RP}). This results in a substantial number of wasted clock cycles (often called a “page fault”) for a small number of data transfers, and hence a poor SDRAM Data Bus efficiency.

One of the advantages of modern SDRAMs is that they have multiple internal banks and the ability to have a page open in each bank at the same time. This enables the concept of “page fault hiding”, or “bank interleaving”, whereby the RAS cycle for a second bank is squeezed into wasted cycles and the second bank is ready for a CAS cycle without experiencing the full page fault. This also helps hide the precharge time of the first bank. With good planning of successive memory access, the efficiency can be dramatically improved. This is illustrated in *Figure 8-3,“Memory Access Cycles,” on page 8-8* where it is seen that after the initial page fault, the Data Bus is completely utilized in the interleaved case.

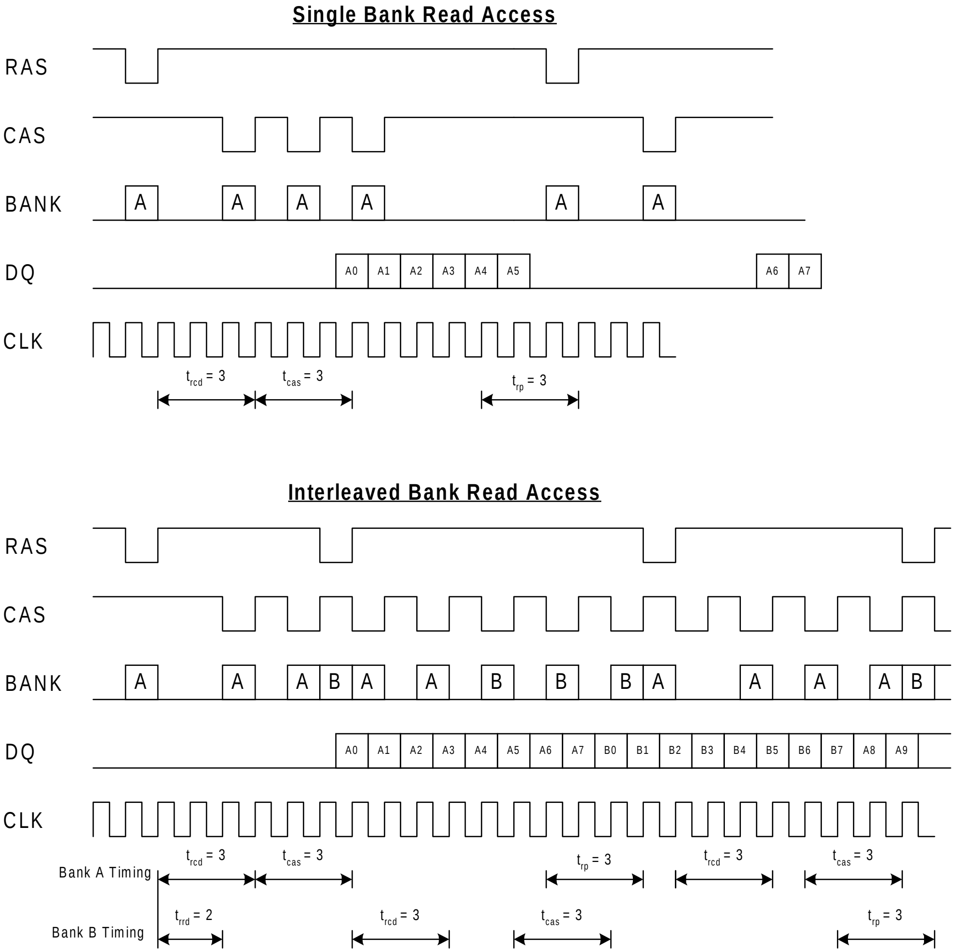


Figure 8-3 Memory Access Cycles

The Xilleon 220 implements a tiling scheme designed to achieve a very high efficiency. Tiling is a method of mapping sequential memory addresses to the SDRAM’s bank and row address bits to specifically cause the bank interleaving described above. The tiling is implemented such that a long sequential access will experience only one page fault with all intermediate ones completely hidden. The mapping is completely internal to the Xilleon 220 and does not require any intervention by the user.

8.4.2.4 SDRAM Refresh

The Xilleon 220 uses the Auto-Refresh method for refreshing the rows of the external SDRAMs. This method is executed by periodically issuing the “Auto-Refresh” command to the SDRAMs, and the SDRAMs maintaining their internal row counter. The task for the memory controller is to issue the refresh command often enough to meet the SDRAM’s specifications.

A programmable timer inside the memory controller establishes the frequency that refresh commands are issued. Typical rates are 7.8 us for DDR and 15.6 us for SDR. The value that is programmed into the timer, N, is determined from:

$$\text{Refresh Rate} = \text{Clock Period} * 64 * N$$

Refresh is normally treated as the lowest-priority client by the Arbiter so that all refreshing is done during idle periods when no other clients are requesting service. But the memory controller keeps a count of missed refreshes to a point and then changes its priority to “very high”, forcing a burst of refreshes to be completed.

8.4.2.5 Clocks and DLLs

The entire memory controller operates on a set of clocks, all derived from a single clock, called XCLK. The frequency of the XCLK is the same as that of the clock used on the SDRAM interface.

Another clock, YCLK, is 2*XCLK. It is used in the final logic of the SDRAM interface to establish the timing relationships between the different SDRAM interface signals. As an example, in the DDR SDRAM interface, the Xilleon 220 generates Data (DQ) on the rising edges of YCLK, which is equivalent to using both edges of XCLK. The Data Strobe (QS) is generated on the falling edges of YCLK to produce the required relationship where QS is delayed by XCLK/4 so that it is centered on the DQ valid timing window.

A similar relationship can be created using falling edges for DQ and rising edges for QS. Every group of SDRAM interface signals can be programmed to select the edge of the YCLK that is used for final clocking. This flexibility enables a wide variety of timing and loading conditions to be met on the PC Board with good timing margin.

The Xilleon 220 also has an internal Delay-Locked Loop (DLL) which can be used to generate the clock output signal at the CLKA or CLKB output pin instead of YCLK. This option provides a finer degree of variability over the exact position of the CLK. The clock generated by the DLL is referred to as HCLK. There is a separate DLL for each memory channel.

8.4.2.6 SDRAM Read Data Capture Methods

When attempting to operate the SDRAM interface at the highest possible clock rate, the Read Data capture timing can be rather difficult to set up. This is due to the accumulation of uncertainties in the I/O timing of the SDRAMs, the Xilleon 220, as well as the printed traces of the PC Board. Several methods are available, and the best choice will depend on the particulars of the situation. The methods are somewhat different for SDR and DDR.

DDR

For DDR, the read data is captured in the Xilleon 220 with the QS data strobe sent with the DQ data by the SDRAM. The DQ and QS arrive at the Xilleon 220 after delays in the SDRAM and the PC Board. As received, the QS is “edge-aligned” with the DQ data, so the Xilleon 220 must delay the QS so that it is centered on the data-valid window with good setup and hold timing margin.

Since the QS is eventually used as the data-capture clock inside the Xilleon 220, some gating signals must be aligned to the Xilleon 220. First of all, the QS input receiver (QS_RECVREN) needs to be turned on, but since the QS is a bi-directional signal that may be in a Hi-Z state, it cannot be turned on until the QS preamble has started; otherwise, spurious clock edges may occur. Second of all, the ERST signal should be turned on before the first edge of QS; it initializes the data capture circuitry.

Alignment of QS to the data is accomplished by selecting an appropriate delay path via register settings. There is a tapped delay-line inside the Xilleon 220 to shift the QS into the data-valid window. This tapped delay line is part of a second DLL. There is a separate DLL for each memory channel and this DLL is not the same as the one used to position the CLK output.

The ERST position and QS_RCVREN position are generated with the XCLK and are programmable in 1/2 clock steps.

SDR

For SDR, the read data is captured using the Xilleon 220 XCLK. The read data from the SDRAM will arrive at the Xilleon 220 after delays in the SDRAM and the PC Board. First, the read data capture clock must be aligned with the data arrival time so that it is centered on the data-valid window. Then, the ERST signal must be aligned with the read data capture clock to identify the first meaningful clock edge of a read data burst. Positioning ERST can be more difficult for SDR than DDR because the read data capture clock is continuous and the DDR uses the QS strobe.

The ERST position is generated with the XCLK and is programmable in 1/2 clock steps.

The read capture clock may be derived in one of three ways:

Method 1: Use an artificial QS Strobe that is created inside the Xilleon 220. This strobe is similar to the DDR QS, except that only rising edges are used. Then, the DLL can be used, if necessary, for fine positioning. The advantage of this method is that a clean and stable clock (strobe) is used, and it is easier to position the ERST because the QS is not a continuous clock.

Method 2: Position the read data capture clock using the DLL inside the Xilleon 220 to shift the XCLK into the data-valid window. There is a separate DLL for each memory channel. This is a second DLL, and is not the same as the one used to position the CLK output. The advantage of this method is that a clean and stable clock is used. The disadvantage is that the exact positioning may vary with different circuit board designs and clock frequencies, thus requiring variations in register programming.

Method 3: Use the CLKFB output to drive a PC Board trace that is matched in length to the round-trip delay between the Xilleon 220 and the SDRAMs, and drive the QS1 input pin of the Xilleon 220. This will place the clock edge approximately in the data valid window. Then the DLL can be used, in necessary, for fine positioning. The advantage is that the clock positioning can be done externally by adjusting the PC Board CLKFB trace length for different memory configurations without having to change any register settings. The disadvantage is that extreme care must be taken when laying out the PC Board CLKFB trace; otherwise, the clock will be too noisy to be useful.

8.4.2.7 Reset and SDRAM Initialization

The Xilleon 220 memory controller has two reset mechanisms: Hard reset and Soft reset. The general chip reset is the Hard reset and will set all registers in the memory controller to their default value and will set all state machines to their initial state. The Hard reset is part of the normal power-up initialization of the Xilleon 220.

The Soft reset is under software control through a memory controller programming register. It sets all state machines to their initial state but it does not affect the memory controller programming registers. Therefore, memory controller registers will not need to be reprogrammed after a Soft reset. However, state machines, such as the power manager and refresh counters will be reset and lose awareness of their previous state.

The memory controller also executes the SDRAM initialization sequence under software control via the memory controller’s programming registers. The desired value for the SDRAM Mode Register is loaded into a memory controller register, then the SDRAM Reset command bit is set. The memory controller will issue the following command sequence to the SDR SDRAMs:

1. Precharge All
2. 12 Refreshes
3. Mode Register Set
4. 2 Refreshes

The sequence is slightly different for DDR because the extended mode register must also be set, and the SDRAM’s DLL, if present, must be reset and enabled. For DDR, the desired value for the Extended Mode Register is loaded first, and the SDRAM Reset command bit is set. Then the desired value for the Mode Register is loaded and the SDRAM Reset command bit is set a second time. The memory controller will issue the following command sequence to the DDR SDRAMs:

1. Precharge All
2. 2 Refreshes
3. DLL Reset = 0; or Refresh if No DLL
4. DLL Reset = 1; or Refresh if No DLL
5. 8 Refreshes
6. Extended Mode Register Set, or Mode Register Set
7. 2 Refreshes

A status bit in the memory controller registers indicates when the initialization sequence is completed so that the second one can be initiated.

The clocks must be stable and the Xilleon 220 DLLs must be reset and locked before the SDRAM Initialization can be executed.

8.4.2.8 Power Management

The Xilleon memory controller is capable of acquiring four power states:

D0 - Fully operational (next state is D1)
D1 - Fully operational at 1/2 clock rate (next state is either D0 or D2)
D2 - SDRAMs are in self refresh; XCLK is off (next state is either D1 or D3)
D3 - SDRAMs off; XCLK is off (next state must be D0)

The memory controller has a state machine that manages the transition between power levels to ensure that no client requests are dropped. The SDRAMs maintain their contents (except for D3), and all DLLs are reset properly and re-locked after the clock frequency changes.

Power state changes are initiated through the clock block circuitry, which informs the memory controller of a requested power state change. The memory controller responds by blocking any new requests from clients and completing any client requests that are in progress. The CLK output to the SDRAMS is disabled (via CKE) and the clock block is notified that it is safe to change the clock. The clock block changes the clock frequency, resets the Xilleon DLLs and informs the memory controller when the clock is stable. The memory controller re-enables the CLK output to the SDRAMs, re-initializes the SDRAMs (if D3->D0), and re-enables the clients request inputs.

Another power-saving feature in the memory controller is the Dynamic-CKE, which can be enabled with a programming register. When enabled, the memory controller will disable the clock to the SDRAMs (via CKE) whenever there are no memory transactions being processed. This results in power saving in the SDRAM components.

8.4.2.9 Programming Register Groups

There are numerous programming registers for the Xilleon 220 memory controller, most of which are programmed during initialization after power up and then remain static. The following list identifies the types of registers that are available in the memory controller:

1. Mode and Configuration **-** declare physical configuration information, such as number of channels, number of row and column address bits, DDR or SDR, etc.
2. SDRAM Timing **-** program the timing requirements of the SDRAM components being used with values for Latency, RAS-CAS delay, refresh rate, etc.
3. Read Timing **-** position the signals for Read Data Capture
4. GUI Paging **-** set the base address offset for the Graphics Buffers
5. Protected Address - define protected address regions
6. SDRAM Mode Register **-** load the values needed for SDRAM initialization

7 IO Driver Controls **-** adjust drive strengths and slew rates and Output Enable timing

8 Clock Selection **-** output clock source and edge selection for all outputs to SDRAMs

9 DLL Selections **-** adjust skews and delay-line taps in DLLs

10 Client Disables **-** disable selected clients from accessing memory

The following table lists registers in the memory controller and the corresponding functions that they affect:

Table 8-5 Functions Affected by Memory Controller Registers

Register Name	Functional Block Affected by Register					
	Client Interfaces	**Arbiter**	**Sequencer**	**Init. & Refresh**	**Power Mgmt.**	**Drive Strength Control**
MEM_BUF_CNTL	X					
MEM_CNTL	X		X	X	X	
EXT_MEM_CNTL			X			
MEM_ADDR_CONFIG	X		X		X	
MEM_RD_TIMING_CNTL			X			
GUI_ADDR_CONFIG	X					
PROTECT_ADDR		X	X			
MEM_SDRAM_MODE_REG			X	X	X	
MEM_IO_CNTL_DQA			X			
MEM_IO_CNTL_AA			X			
MEM_IO_CNTL_CLKA			X			
MEM_IO_CNTL_DQB			X			
MEM_IO_CNTL_AB			X			
MEM_IO_CNTL_CLKB			X			
MEM_DLL_CNTL_HCLKA			X			
MEM_DLL_CNTL_RDCLKA			X			
MEM_DLL_CNTL_HCLKB			X			
MEM_DLL_CNTL_CLKB			X			
MEM_IO_OE_CNTL			X			
MEM_IMP_COMP1						X
MEM_CLIENT_DISABLES	X	X				

Please refer to the Register Reference Manual for details about each register.

8.4.3 Performance Limitations and Restrictions

- For 32-bit single channel configurations, Channel A must be used.
- For 64-bit single channel configurations, both Channel A and B are populated with SDRAMs, but only channel A of the memory controller is used.
- Mixing SDR and DDR in the same configuration is prohibited.

- SGRAMs can be used instead of SDRAMs, but the “block move” feature is not supported.
- Maximum clock frequency for SDRAMs is 166 MHz.
- When two physical ranks (chip selects) are used, the additional loading on the CLK, Command, DQ and QS signals may make it difficult to achieve the maximum clock rate unless extremely good PC Board layout techniques are used.
- When x8-bit SDRAMs are used, the additional loading on the CLK and Command signals may make it difficult to achieve the maximum clock rate unless extremely good PC board layout techniques are used.
- x8-bit SDRAMs in a two-rank configuration is not recommended.
- In general, a 2-channel/1-rank configuration will have higher performance than an equal-sized 1-channel/2-rank configuration.
- Internal DLLs should not be used at clock frequencies below 66 MHz.
- For two channel configurations, different SDRAM timing parameters can be programmed for each channel; however, they will always operate at the same clock frequency.
- The same programmed refresh rate is used for both channels.
- If a client attempts to read an address beyond the defined physical memory space, the memory controller will return null data and set a flag. If a client attempts to write an address beyond the defined physical memory space, it will be neutralized and a flag will be set.
- If a graphics client or the host-processor attempts to read a protected address, the memory controller will return null data and set a flag. If a graphics client or the host-processor attempts to write a protected address, it will be neutralized and a flag will be set.
- The memory controller does not provide inter-client coherence checking between different clients. Certain clients have a single read/write interface with the memory controller, so intra-client coherence is automatically provided because the memory controller processes each client’s request stream in the order in which it is received.

8.4.4 PC Board Design Considerations

In order to achieve the highest memory clock frequency using the Xilleon 220, good PC Board layout techniques are crucial. Thus, an understanding of transmission line effects and termination methods is essential. Sufficiently matching the loading and trace lengths of particular signal groups is also key. The following are guidelines for the SDRAM interface that will allow high-performance to be achieved.

Guidelines for SDRAM interface trace layout:

- Strobe to clock trace length difference (QS : CLK) should be within +/-3 mm.
- Data to strobe trace length difference (DQ, DQM : QS) should be within +/-7.5 mm within each QS group (e.g. DQ[7:0], DQM[0] : QS[0])

- Command to clock trace length difference (ADDR, CS, RAS, CAS, WE, CKE : CLK) should be within +/-30 mm.
- Clock differential pair trace length difference (CLK : CLKb) should be within +/-3 mm.
- Maximum trace length should be up to 50 mm.
- Strobe and corresponding data (QS,DQ,DQM) should be routed together on the same layers.
- Clock differential pair should be routed together on the same layers.
- Clocks and strobes should be shielded with ground on the same routing layer.
- Ground shields should be stitched with vias to ground layer every 10-20 mm.
- Non point-to-point connections should be routed in the form of balanced T branches.
- Spacing should be 2:1 (space:trace-width) except for clock differential pair (1:1).
- If serpentine delays are used to extend trace length, then, the spacing between curves should be at least 2:1 to trace width.
- Signal routing should not be on adjacent layers; board stackup should separate signal routing layers with power or ground layers.
- Unused areas should be filled with ground or power and stitched to appropriate ground or power layer with multiple vias.

Guidelines for terminations:

- Resistor packs used for termination should not be shared between different signal groups.
- Series termination resistors should be placed as close as possible to drivers, less than 10 mm away.
- Parallel termination resistors should be placed as close as possible to signal trace, less than 5 mm away.
- Parallel termination voltage should be treated as power, and distributed through a dedicated layer or with wide traces, and filtered appropriately.
- Trace impedance should be kept constant - if the same trace width is used for routing in all layers, board stack-up should be chosen to preserve constant trace impedance.

Guidelines for Power and Ground

- Power and ground should be whole non-interrupted plane layers, with filtering capacitors directly under and as close as possible to the ASIC and memories.
- Filtering capacitors should be a mix of tantalum 10-22 uF, ceramic 100 nF and ceramic 10nf (preferably dielectric NP0).
- Every filtering capacitor should have its own power and ground via going directly to the power and ground planes.

- Every ASIC power/ground ball should have its own via going directly to the power/ground planes.
- Power and ground connections from a connector to the distribution planes, and to/from any series filtering inductance/ferrite, should be in the form of double/triple via connections.

Chapter 9
Pinout and Strap Descriptions

9.1 Pin Summary

For ease of reference, pins are grouped according to their interface functionality in this chapter. In the table below, pins multiplexed with General Purpose pins are shaded. For top view/pin assignment, please refer to *Figure 9-1* and *Figure 9-2*. For a full listing of signal names vs ball (pin) reference numbers or vice-versa, refer to *Appendix A*.

Table 9-1 Pin Groups

Group by Functionality	Pin Count
Host (PCI)	62
Memory	131
Smart Card	21
Flex Bus	41 (8 pins muxed w/ Smart Card)
Transport Stream	36 (12 pins muxed w/ G P pins)
PLL & Xtal	10
I^2S	11
I^2C	4
SPDIF	2
AC-Link	7
Timers Port	8
GP and Test	34+1
CRTC	4
TV Out DACs (Primary and Secondary)	25
DVI Out	21
ITU-656	31
USB	5
IDE	27
VIP	5
IR	2
LPC	9
Serial Port	16 (8 pins muxed w/ G P pins, 8 with PCI pins)
IEEE 1394	13
Out of Band (OOB)	5
VDDC, VDDC18	34
VPP	9
VDDR1, VDDR3	37
VSS	121

9.2 Xilleon 220 Pin Assignment Top-View

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	DQB(17)	DQB(16)	WEBb	CSBb(1)	AB(13)	CLKFBB	AB(8)	AB(4)	AB(0)	DQB(12)	DQB(10)	DQB(6)	DQMBb(0)	DQB(0)	XMVDD
B	DQMBb(2)	DQB(19)	DQB(18)	RASBb	AB(14)	AB(10)	AB(9)	AB(5)	AB(1)	DQB(13)	DQB(11)	DQB(7)	QSB(0)	DQB(1)	VREF
C	DQB(23)	DQB(22)	QSB(2)	CASBb	CKEB	AB(11)	CLKB	AB(6)	AB(2)	DQB(14)	DQMBb(1)	DQB(8)	DQB(4)	DQB(2)	MEMTEST
D	DQB(25)	DQB(24)	DQB(21)	DQB(20)	CSBb(0)	AB(12)	CLKBb	AB(7)	AB(3)	DQB(15)	QSB(1)	DQB(9)	DQB(5)	DQB(3)	MEMVMODE
E	QSB(3)	DQMBb(3)	DQB(27)	DQB(26)	VDDR1	VDDR1	VDDR1	VDDR1	VDDC	VDDC	VDDR1	VDDR1	VDDC	VDDC	VDDC18_RIGHT
F	DQB(31)	DQB(30)	DQB(28)	DQB(29)	VDDR1										
G	SMDATAA	SMDETECTA	SMRSTA	SMCLKA	VDDR1										
H	SMVCCB	SMRSTB	SMCLKB	SMVCCA	VDDR3										
J	SMRSTS	SMCLKS	SMDATAB	SMDETECTB	VDDR3										
K	IRRXD	SMDATAS	SMDETECTS	SMVCCS	VDDC										
L	FAD(15)	FGNTb	FREQb	IRTXD	VDDC						VSS	VSS	VSS	VSS	VSS
M	FAD(12)	FAD(13)	FAD(14)	FCLK	VDDR3						VSS	VSS	VSS	VSS	VSS
N	FAD(8)	FAD(9)	FAD(11)	FAD(10)	VDDR3						VSS	VSS	VSS	VSS	VSS
P	FAD(4)	FAD(5)	FAD(6)	FAD(7)	VDDC18_TOP						VSS	VSS	VSS	VSS	VSS
R	FAD(3)	FAD(2)	FAD(1)	FBE(1)	VDDC						VSS	VSS	VSS	VSS	VSS
T	FAD(0)	FBE(0)	FALE(1)	FALE(0)	VDDC						VSS	VSS	VSS	VSS	VSS
U	FSTB	FRDYb	FWE	FRD	VDDC						VSS	VSS	VSS	VSS	VSS
V	FCEb(5)	FCEb(4)	FCEb(3)	FCEb(1)	VDDR3						VSS	VSS	VSS	VSS	VSS
W	FCEb(2)	FCEb(0)	NRSSCLKA	NRSSDATA_INA	VDDR3						VSS	VSS	VSS	VSS	VSS
Y	NRSSDATA_OUTA	TCLKA	TDATA(7)	TDATA(6)	VDDC										
AA	TDATA(5)	TDATA(4)	TDATA(3)	TDATA(2)	VDDC										
AB	TDATA(1)	TDATA(0)	TSTRTA	NRSSCLKB	VDDR3										
AC	TVLDA	TERA	NRSSDATA_INB	NRSSDATA_OUTB	VDDR3										
AD	TCLKB	TDATB(7)	TDATB(6)	TDATB(5)	VDDR3										
AE	TDATB(4)	TDATB(3)	TDATB(2)	TSTRTB	VDDR3	VDDR3	VDDR3	VDDR3	VDDC	VDDC	VDDR3	VDDR3	VDDC	VDDC	VDDC18_LEFT
AF	TDATB(0)	TDATB(1)	TVLDB	APVDD	I2SWS_OUTA	I2CDATAA	GPIOA(12)	GPIOA(10)	GPIOA(4)	GPIOA(0)	VIPCLK	ACDATA_OUT	ACXTAL_OUT	USBNB	IDE_DD(13)
AG	TERB	VCXO_INA	PWMA	APVSS	I2SSD_OUTA	I2CCLKB	GPIOA(11)	GPIOA(7)	GPIOA(3)	TESTEN	VIPHCTL	ACDATA_IN(1)	USBPB	IDE_DA(1)	IDE_CSb(0)
AH	VCXO_INB	XTALIN	XTALOUT	PWMB	I2SSOSCK_OUTA	I2CDATAB	GPIOA(9)	GPIOA(6)	GPIOA(2)	VIPHAD(1)	ACSYNC	ACDATA_IN(0)	USBNA	IDE_DA(2)	IDE_CSb(1)
AJ	I2SSCK_IN	I2SWS_IN	I2SSD_IN	I2SSCK_OUTA	I2CCLKA	GPIOA(13)	GPIOA(8)	GPIOA(5)	GPIOA(1)	VIPHAD(0)	ACBIT_CLK	ACRESETb	USBPA	USB_OVRCUR	IDE_DA(0)
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figure 9-1 Xilleon 220 Top View (Left Half)

16	17	18	19	20	21	22	23	24	25	26	27	28	29	
XMVSS	DQA(28)	DQA(26)	DQA(22)	DQMAb(2)	DQA(16)	CSAb(1)	AA(13)	CLKFBA	AA(8)	AA(4)	AA(1)	DQA(15)	DQA(14)	A
DQA(31)	QSA(3)	DQA(25)	DQA(21)	DQA(19)	WEAb	CSAb(0)	AA(12)	CLKAb	AA(7)	AA(3)	AA(0)	DQA(13)	DQA(12)	B
DQA(30)	DQMAb(3)	DQA(24)	DQA(20)	DQA(18)	CASAb	CKEA	AA(11)	CLKA	AA(6)	AA(2)	DQMAb(1)	QSA(1)	DQA(11)	C
DQA(29)	DQA(27)	DQA(23)	QSA(2)	DQA(17)	RASAb	AA(14)	AA(10)	AA(9)	AA(5)	DQA(10)	DQA(9)	DQA(8)	DQA(7)	D
VDDC	VDDC	VDDR1	VDDR1	VDDC	VDDC	VDDR1	VDDR1	VDDR1	VDDR1	DQA(6)	DQA(5)	DQA(4)	QSA(0)	E
									VDDR1	DQMAb(0)	DQA(3)	DQA(2)	DQA(1)	F
									VDDR1	DQA(0)	AD(0)	AD(1)	AD(2)	G
									VDDP	AD(3)	AD(4)	AD(5)	AD(6)	H
									VDDP	AD(7)	CBEb(0)	AD(8)	AD(9)	J
									VDDC	AD(10)	AD(11)	AD(12)	AD(13)	K
VSS	VSS	VSS	VSS						VDDC	AD(14)	AD(15)	CBEb(1)	PAR	L
VSS	VSS	VSS	VSS						VDDP	SERRb	PERRb	STOPb	DEVSELb	M
VSS	VSS	VSS	VSS						VDDP	TRDYb	IRDYb	FRAMEb	CBEb(2)	N
VSS	VSS	VSS	VSS						VDDC18_BOTTOM	AD(16)	AD(17)	AD(18)	AD(19)	P
VSS	VSS	VSS	VSS						VDDC	CBEb(3)	AD(22)	AD(21)	AD(20)	R
VSS	VSS	VSS	VSS						VDDC	AD(25)	AD(24)	AD(23)	IDSEL	T
VSS	VSS	VSS	VSS						VDDC	AD(28)	AD(29)	AD(27)	AD(26)	U
VSS	VSS	VSS	VSS						VDDP	REQb(1)	REQb(0)	AD(31)	AD(30)	V
VSS	VSS	VSS	VSS						VDDP	RESETOUTb	GNTb(1)	GNTb(0)	REQb(2)	W
									VDDC	GNTb(2)	PCICLKOUT(2)	PCICLKOUT(1)	PCICLKOUT(0)	Y
									VDDC	INTROUTb	RESETb	PCICLKOUT(3)	PCICLK	AA
									VDDP	SERIRQb	INTRb(0)	INTRb(2)	INTRb(1)	AB
								VDDP	VDDP	LFRAMEb	LCLK	LDRQb(1)	LDRQb(0)	AC
VDDC	VDDR3	VDDR3	VDDR3	VDDC	VDDC	VDDR3	VDDR3	VSSDI	AVSSN	LAD(3)	LAD(2)	LAD(1)	LAD(0)	AD
VDDC	VDDR3	IDE_DD(1)	IDE_DMACKb	VSS2DI	A2VDDQ	A2VSSQ	A2VSSN	VDDDI	AVSSN	GPIOB(0)	GPIOB(2)	GPIOB(1)	GPIOB(3)	AE
IDE_DD(11)	IDE_DD(7)	IDE_DD(3)	IDE_IOWb	RDB	VDD2DI	A2VSSN	AVSSQ	AVDD	AVDD	GPIOB(5)	GPIOB(9)	GPIOB(17)	GPIOB(16)	AF
IDE_DD(12)	IDE_DD(8)	IDE_DD(4)	IDE_IORb	IDE_INTRQ	PMIVSS	A2VDD	A2VDD	AVDDQ	DAC3	GPIOB(4)	GPIOB(13)	GPIOB(7)	GPIOB(6)	AG
IDE_DD(15)	IDE_DD(9)	IDE_DD(5)	IDE_DD(0)	IDE_DMARQ	PMIVDD	DAC5	DAC4	DAC0	DAC2	GPIOB(18)	GPIOB(14)	GPIOB(10)	GPIOB(8)	AH
IDE_DD(14)	IDE_DD(10)	IDE_DD(6)	IDE_DD(2)	IDE_IORDY	TDB	DAC6	R2SET	DAC1	RSET	GPIOB(19)	GPIOB(15)	GPIOB(11)	GPIOB(12)	AJ
16	17	18	19	20	21	22	23	24	25	26	27	28	29	

Figure 9-2 Xilleon 220 Top View (Right Half)

9.3 Signal List

9.3.1 PCI Interface

Table 9-2 PCI Interface Pins

Signal Name	Type	Description
RESETb	I	Active Low PCI Reset
RESETOUTb	O	Active Low PCI Reset Out in Solo mode
PCICLK	I	Bus Clock IN
PCICLKOUT(3:0)	O	Bus Clocks Out in Solo mode
AD(31:0)	I/O	Multiplexed Address/Data (31:0)
CBEb(3:0)	I/O	Bus Command/Byte Enable(3:0)
FRAMEb	I/O	Cycle Frame
IRDYb	I/O	Initiator (bus master) ready
TRDYb	I/O	Target device ready
DEVSELb	I/O	Device select
STOPb	I/O	Target transaction termination request
PAR	I/O	Parity bit for AD(31:0) and CBEb(3:0)
INTROUTb	O	Interrupt request out
INTRb(2:0)	I	Interrupt inputs
REQb0	I/O	PCI Bus Master Request signal to external arbiter in Peer modes, **OR** PCI Bus Master Request signal to Xilleon 220's arbiter in Solo modes
REQb(2:1)	I	PCI Bus Master Request signal to Xilleon 220's arbiter.
IDSEL	I	PCI initialization device select
PERRb	I/O	Parity Error
SERRb	I/O	System Error
GNTb(0)	I/O	PCI Bus Master Grant signal from external arbitern Peer modes, **OR** PCI Bus Master Grant signals generated by Xilleon 220's arbiter in Solo modes
GNTb(2:1)	O	PCI Bus Master Grant signals generated by Xilleon 220's arbiter

9.3.2 Smart Card Interface

There are three groups of pins for this interface: Card A, Card B, and Shopping. They are separated by the shading of Card B pins.

Table 9-3 Smart Card Interface

Signal Name	Type	Description
SMCLKA	O	Smart Card Clock, **OR** general purpose IO

Table 9-3 Smart Card Interface (Continued)

Signal Name	Type	Description
SMRSTA	O	Smart Card Reset, **OR** general purpose IO
SMVCCA	O	Smart Card Power Control, **OR** general purpose IO
SMDETECTA	I	Smart Card Present Detect, **OR** general purpose IO
SMDATAA	I/O	Smart Card Serial Data, **OR** general purpose IO
NRSSCLKA	O	NRSS CA module Clock, **OR** general purpose IO
NRSSDATA_INA	I	NRSS CA module Data In, **OR** general purpose IO
NRSSDATA_OUTA	O	NRSS CA module Data Out, **OR** general purpose IO
SMCLKB	O	Smart Card Clock, **OR** general purpose IO
SMRSTB	O	Smart Card Reset, **OR** general purpose IO
SMVCCB	O	Smart Card Power Control, **OR** general purpose IO
SMDETECTB	I	Smart Card Present Detect, **OR** general purpose IO
SMDATAB	I/O	Smart Card Serial Data, **OR** general purpose IO
NRSSCLKB	O	NRSS CA module Clock, **OR** general purpose IO
NRSSDATA_INB	I	NRSS CA module Data In, **OR** general purpose IO
NRSSDATA_OUTB	O	NRSS CA module Data Out, **OR** general purpose IO
SMCLKS	O	Smart Card Clock, **OR** general purpose IO
SMRSTS	O	Smart Card Reset, **OR** general purpose IO
SMVCCS	O	Smart Card Power Control, **OR** general purpose IO
SMDETECTS	I	Smart Card Present Detect, **OR** general purpose IO
SMDATAS	I/O	Smart Card Serial Data, **OR** general purpose IO

9.3.3 Memory (Dual Channel) Interface

Table 9-4 Memory Interface Pins

Signal Name	Type	Description
DQA(31:0)	I/O	Channel A BiDirectional Data Bus
DQMA(3:0)	O	Channel A Data Byte Mask
QSA(3:0)	I/O	Channel A BiDirectional Data Strobe Bus
AA(14:0)	O	Channel A Address Bus
RASAb	O	Channel A Row Address Select
CASAb	O	Channel A Column Address Select
WEAb	O	Channel A Write Enable
CSAb(1)	O	Channel A Chip Select 1
CSAb(0)	O	Channel A Chip Select 0
CKEA	O	Channel A Clock Enable
CLKA	O	Channel A Non-Inverted Clock
CLKAb	O	Channel A Inverted Clock
CLKFBA	O	Channel A Feedback Non-Inverted Clock
DQB(31:0)	I/O	Channel B BiDirectional Data Bus
DQMB(3:0)	O	Channel B Data Byte Mask
QSB(3:0)	I/O	Channel B BiDirectional Data Strobe Bus
AB(14:0)	O	Channel B Address Bus
RASBb	O	Channel B Row Address Select
CASBb	O	Channel B Column Address Select
WEBb	O	Channel B Write Enable
CSBb(1)	O	Channel B Chip Select 1
CSBb(0)	O	Channel B Chip Select 0
CKEB	O	Channel B Clock Enable
CLKB	O	Channel B Non-Inverted Clock
CLKBb	O	Channel B Inverted Clock
CLKFBB	I	Channel B Feedback Non-Inverted Clock
VREF	I	Reference Voltage: - 1.25V Typ. for SSTL-2 / 0.5 * VDD - 1.50V Typ. for SSTL-3 / 0.45 *VDD Note: if the differential signaling interface is not used, this pin must be connected to 1.8V
MEMTEST	I	External fixed reference for memory interface compensator. This fixed reference is a resistor to ground. The compensator will drive the resistor with an output buffer in the MEMTEST pad, and measure the voltage. This voltage is an indication of the strength of the output driver. The compensator will adjust it accordingly. Resistor value should be 45 ohm +/-1%.

Table 9-4 Memory Interface Pins   (Continued)

Signal Name	Type	Description
MEMVMODE	I	Voltage value for S(D/G)RAM chips: - 0 or 0V for VDDR1 2.5V - 1 or 1.8V for VDDR1 3.3V

9.3.4 Transport Stream Interface

There are three groups of pins for this interface: Stream A, Stream B, and Stream C. They are separated by the shading of Stream B pins. Stream A and Stream B pins are "in" only (single parallel/ dual serial, see *Table 9-6* below); Stream C pins are "out" only (parallel/serial) and are multiplexed in the General Purpose bus (*see "GPIO Pins Multiplexing" on page 9-16*).

Table 9-5 Transport Stream Pins

Signal Name	Type	Description
TCLKA	I	Serial/Parallel Clock, **OR** general purpose IO
TDATA(0)	I	Serial/Parallel Data In, **OR** general purpose IO
TDATA(7:1)	I	Parallel Data In, **OR** general purpose IO
TSTRTA	I	Stream Start, **OR** general purpose IO
TVLDA	I	Stream Valid, **OR** general purpose IO
TERA	I	Stream Error, **OR** general purpose IO
TCLKB	I	Serial/Parallel Clock
TDATB(0)	I	Serial/Parallel Data In
TDATB(7:1)	I	Parallel Data In
TSTRTB	I	Stream Start
TVLDB	I	Stream Valid
TERB	I	Stream Error
TCLKC	O	Serial/Parallel Clock
TDATC(0)	O	Serial/Parallel Data In
TDATC(7:1)	O	Parallel Data In
TSTRTC	O	Stream Start
TVLDC	O	Stream Valid
TERC	O	Stream Error

Table 9-6 Transport Stream A Single Parallel / Dual Serial Multiplexing

Single Parallel	Dual Serial
TCLKA	TCLKAS1
TDATA(0)	TDATAS1
TDATA(1)	TCLKAS2
TDATA(2)	TDATAS2
TDATA(3)	TSTRTAS2
TDATA(4)	TVLDAS2
TDATA(5)	TERAS2
TDATA(6)	
TDATA(7)	
TSTRTA	TSTRTAS1
TVLDA	TVLDAS1
TERA	TERAS1

9.3.5 PLLs & XTAL

Table 9-7 PLLs & XTAL

Signal Name	Type	Description
XTALIN	I	PLL Reference Clock or MXCLK source (14.318 – 29.4989Mhz)
XTALOUT	O	
VCXO_INA	I	STC Counter post-tracked clock
PWMA/ AUDIO_CLKOUTA	O	STC Counter tracking direction (VCXO Control for VCXO A), **OR** 27Mhz audio output clock
VCXO_INB	I	STC Counter post-tracked clock
PWMB/ AUDIO_CLKOUTB	O	STC Counter tracking direction (VCXO Control for VCXO B), **OR** 27Mhz audio output clock
XMVDD	I	Memory/Engine Phase Locked Loop Power
XMVSS	O	Memory/Engine Phase Locked Loop Ground
PMIVDD	I	Display/MIPS Phase Lock Loop Power
PMIVSS	O	Display/MIPS Phase Lock Loop Ground
APVDD	I	Audio/SB Phase Lock Loop Power
APVSS	O	Audio/SB Phase Lock Loop Ground

9.3.6 CRTC Interface

Note: These signals are multiplexed in the General Purpose Bus (*see "GPIO Pins Multiplexing" on page 9-16)*.

Table 9-8 CRTC

Signal Name	Type	Description
HSYNC	O	Horizontal Sync for CRT monitor
VSYNC	O	Vertical Sync for CRT monitor
CRTDDCDATA	I/O	DDC data
CRTDDCLK	O	DDC clock

9.3.7 Primary and Secondary Display DAC

Table 9-9 DAC 1 Pins

Signal Name	Type	Description
DAC0	O	R, C, Pr, Comp
DAC1	O	G, Y, Comp
DAC2	O	B, Comp, Pb, C
DAC3	O	Sync, Comp, Y
RSET	O	Internal Reference
VDDDI	I	DAC macro digital VDD **(1.8V)**
AVDD (2 balls)	I	DAC macro VDD **(2.5V)**
AVDDQ	I	DAC Band Gap Ref. Voltage **(1.8V)**
VSSDI	O	DAC macro VSS (digital ground)
AVSSQ	O	DAC macro VSS (quiet ground; used for bandgap)
AVSSN (2 balls)	O	DAC macro VSS (noisy ground; used as current dump)

Table 9-10 DAC 2 Pins

Signal Name	Type	Description
DAC4	O	R, C, Pr, Comp
DAC5	O	G, Y, Comp
DAC6	O	B, Comp, Pb
R2SET	O	Internal Reference
VDD2DI	I	DAC macro digital VDD **(1.8V)**
A2VDD (2 balls)	I	DAC macro VDD **(2.5V)**
A2VDDQ	I	DAC Band Gap Ref. Voltage **(1.8V)**
VSS2DI	O	DAC macro VSS (digital ground)

Table 9-10 DAC 2 Pins (Continued)

Signal Name	Type	Description
A2VSSQ	O	DAC macro VSS (quiet ground; used for bandgap)
A2VSSN (2 balls)	O	DAC macro VSS (noisy ground; used as current dump)

DAC Configurations

Every DAC can be configured to carry any of 9 possible outputs : S-Video Luma (Y), S-Video Chroma (C), Component Output Y, Component Output Pr, Component Output Pb, Red (R), Green (G), Blue (B), or Composite.

The tables below show some possible output configurations but are not exhaustive. The configurations also demonstrate the capabilities of the Xilleon 220 by showing all DACs being used, even though an actual implementation may not require the use of all DACs (e.g. an S-Video plus Composite scenario uses only 3 DACs; however, it is possible to have the 4th DAC also put out Composite as shown in *Table 9-11*).

In the case of Component Video, note that eitherY/Pr/Pb or Composite may be active, but not both simultaneously.

Table 9-11 DAC Configurations - Primary Video Encoder

Example Output Configurations	First Video Encoder (a.k.a. HD0 and SD0)			
Signal Name/Ball Reference	DAC0/AH24	DAC1/AJ24	DAC2/AH25	DAC3/AG25
(1) Component Video *or* Composite	Pr	Y	Pb	(Not used, or Composite if not using Y/Pr/Pb)
(2) S-Video + Composite + Composite	C	Y	Composite	Composite
(3) S-Video + S-Video	C	Y	C	Y
(4) Composite Video x 4	Composite	Composite	Composite	Composite
(5) RGB / SCART	R	G	B	Comp/Sync
Fields in Register DAC_MUX_OUT_CNTL	DAC_MUX0_OUT	DAC_MUX1_OUT	DAC_MUX2_OUT	DAC_MUX3_OUT

Table 9-12 DAC Configurations - Secondary Video Encoder

Example Output Configurations	Second Video Encoder (a.k.a. SD1)		
Signal Name/Ball Reference	DAC4/AH23	DAC5/AH22	DAC6/AJ22
(6) Component Video	Pr	Y	Pb
(7) S-Video + Composite	C	Y	Composite
(8) Composite Video x 3	Composite	Composite	Composite
(9) RGB / SCART	R	G	B
Fields in Register DAC_MUX_OUT_CNTL	DAC_MUX4_OUT	DAC_MUX5_OUT	DAC_MUX6_OUT

9.3.8 DVI Out Interface

Note: These signals are multiplexed in the General Purpose Bus (*see “GPIO Pins Multiplexing” on page 9-16*).

Table 9-13 DVI Out Interface

Signal Name	Type	Description
DVIDDCDATA	I/O	I^2C Data for panel HDCP and DDC
DVIDDCCLK	O	I^2C Clock for panel HDCP and DDC
DVOHPD	I	Panel hot-plug support
DVOCNTL3	O	TMDS Control Bit-3
DVOCLK0	O	Data Clock +
DVOCLK1	O	Data Clock -
DVODE	O	Data Enable
DVOHSYNC	O	Horizontal Sync input control signal
DVOVSYNC	O	Vertical Sync input control signal
DVODATA(11:0)	O	Pixel Data

9.3.9 ITU-656 In/Out Interface

Table 9-14 ITU-656 Interface

Signal Name	Type	Description
DVSCLKA	I	Port A Clock
DVSYDATA(7:0)	I	Port A Data
DVSCLKB/HREF	I	Port B Clock, **OR** HSYNC reference
DVSYDATB(7:0)	I	Port B Data
VREF	I	VSYNC reference
DVSCLKC	O	Port A Clock
DVSYDATC(9:0)	O	Port A Data

Note: Port A pins “in” only, multiplexed in Transport Stream B, see *Table 9-15* below.
Port B “in” only, multiplexed in PCI and FlexBus/IR (for ASIC A21 and higher only) and GP bus, see *Table 9-16* below.
Port C “out” only, multiplexed in PCI and GP bus.

Table 9-15 ITU-656 and Transport Stream B Multiplexing

Transport (Stream B, Single Parallel)	Transport (Stream B, Dual Serial)	ITU-656 (Port A – IN)
TCLKB	TCLKBS1	DVSCLKA
TDATB(0)	TDATBS1	DVSYDATA(0)

Table 9-15 ITU-656 and Transport Stream B Multiplexing (Continued)

Transport (Stream B, Single Parallel)	Transport (Stream B, Dual Serial)	ITU-656 (Port A – IN)
TDATB(1)	TCLKBS2	DVSYDATA(1)
TDATB(2)	TDATBS2	DVSYDATA(2)
TDATB(3)	TSTRTBS2	DVSYDATA(3)
TDATB(4)	TVLDAS2	DVSYDATA(4)
TDATB(5)	TERBS2	DVSYDATA(5)
TDATB(6)		DVSYDATA(6)
TDATB(7)		DVSYDATA(7)
TSTRTB	TSTRTBS1	
TVLDB	TVLDBS1	
TERB	TERBS1	

Table 9-16 ITU-656 and FlexBus and IR Multiplexing

FlexBus and IR	ITU-656 (Port B IN)
FREQb	HREF/DVSCLKB
FGNTb	DVSYDATB(7)
FCEb(5)	DVSYDATB(6)
FCEb(4)	DVSYDATB(5)
FCEb(3)	DVSYDATB(4)
FCEb(2)	DVSYDATB(3)
FCEb(1)	DVSYDATB(2)
FSTB	DVSYDATB(1)
FRDYb	DVSYDATB(0)
IRTXD	VREF

Note:

- Set GPIO_SELECTION(1) = '1' to enable ITU-656 Port B on FlexBus and IR.
- This function is not available for A11 and A12 chips. It is only available for revisions A21 and on.

9.3.10 SPDIF Interface

Note: These signals are multiplexed in the General Purpose bus (*see “GPIO Pins Multiplexing” on page 9-16*).

Table 9-17 SPDIF Interface

Signal Name	Type	Description
SPDIFA	O	Channel-A SPDIF serial data out

Table 9-17 SPDIF Interface (Continued)

Signal Name	Type	Description
SPDIFB	O	Channel-B SPDIF serial data out

9.3.11 I2S Interface

There are three groups of pins for this interface: Channel A, Channel B, and Channel C. They are separated by the shading of Channel B pins. Channels A and B signals are “out”, Channel C signals are “in”, Channel B and Channel C are multiplexed in the General Purpose Bus (*see “GPIO Pins Multiplexing” on page 9-16*).

Table 9-18 I2S Interface

Signal Name	Type	Description
I2SSCK_OUTA	O	Serial Clock
I2SWS_OUTA	I/O	Word Select/Data Request
I2SSD_OUTA	O	Serial Data
I2SSOSCK_OUTA	O	Over Sampling Clock Out (going to the MCLK of DAC)
DREQAb	I	Data Request. This pin is multiplexed in the GP bus.
I2SSCK_OUTB	O	Serial Clock
I2SWS_OUTB	I/O	Word Select/Data Request
I2SSD_OUTB	O	Serial Data
I2SSOSCK_OUTB	O	Over Sampling Clock Out (going to the MCLK of DAC)
DREQBb	I	Data Request
I2SSCK_IN	I/O	Serial Clock, **OR** general purpose IO
I2SWS_IN	I/O	Word Select, **OR** general purpose IO
I2SSD_IN	I	Serial Data, **OR** general purpose IO

9.3.12 I^{2}C Interface

Table 9-19 I^{2}C Interface

Signal Name	Type	Description
I2CCLKA	I/O	Channel A Serial Data Clock, **OR** general purpose IO
I2CDATAA	I/O	Channel A Serial Bi-directional Data, **OR** general purpose IO
I2CCLKB	I/O	Channel B Serial Data Clock, **OR** general purpose IO
I2CDATAB	I/O	Channel B Serial Bi-directional Data, **OR** general purpose IO

9.3.13 AC-Link Interface

Table 9-20 AC Link Interface

Signal Name	Type	Description
ACSYNC	O	Fixed Rate Sample Sync, **OR** general purpose IO
ACBIT_CLK	I	Serial Data Clock, **OR** general purpose IO
ACDATA_OUT	O	Serial Data Out, **OR** general purpose IO
ACDATA_IN(1:0)	I	Serial Data in, **OR** general purpose IO
ACRESETb	O	Master Reset, **OR** general purpose IO
ACXTAL_OUT	O	Source Serial Clock to AC97 CODECs, **OR** general purpose IO

9.3.14 Timers Port Interface

These signals are multiplexed in both AC-Link and PCI (see *Table 9-22* below).

Table 9-21 Timers Port Interface

Signal Name	Type	Description
CAPT0	I	Capture Timer0
CAPT1	I	Capture Timer1
CAPT2	I	Capture Timer2
CAPT3	I	Capture Timer3
CMP0	O	Compare Timer0
CMP1	O	Compare Timer1
CMP2	O	Compare Timer2
CMP3	O	Compare Timer3

Table 9-22 Timers Port /AC-Link Multiplexing

AC Link	Timers Port
ACSYNC	CAPT0
ACBIT_CLK	
ACDATAT_OUT	COMP0
ACDATA_IN(0)	CAPT1
ACDATA_IN(1)	CAPT2
ACRESETb	CMP1
ACXTAL_OUT	CMP2

Note: Use HBIU.ACLINK_TIMER_GPIO_FUNCTION_SELECT register to control it.

9.3.15 General Purpose BUS Interface

Table 9-23 General Purpose IO Pins

Signal Name	Type	Description
GPIOA(13:0)	I/O	1394 interface, **OR** transport out, **OR** I2S port B out, **OR** Out of Band, **OR** Debug Bus A, **OR** SPDIF A, **OR** 2-bit Host VIP interrupt, **OR** misc test features, **OR** general purpose *see "GPIO Pins Multiplexing" on page 9-16*
GPIOB(19:0)	I/O	SPDIF B, **OR** DVI Out & HDCP & DDC, **OR** ITU-656 video capture/out, **OR** CRT VSYNC/HSYNC/DDC, **OR** EJTAG, **OR** Debug Bus B, **OR** general purpose *see "GPIO Pins Multiplexing" on page 9-16*

Table 9-24 GPIO Pins Multiplexing

GPIOA /GPIOB BUSES	1394 IN/OUT (Serial/Parallel) Transport Out (Serial/Parallel	I2S B OUT	I2S A OUT	SPDIF A/B OUT	DVI OUT	OOB	Host VIP Interrupt	ITU-656 In (Port B)	ITU-656 Out (Port C)	CRT	EJTAG	Serial Port A
GPIOA(0)	TNFCLK/TCLKC											
GPIOA(1)	TNFDAT(0)/TDATC(0)											
GPIOA(2)	TNFDAT(1)/TDATC(1)			SPDIFA								
GPIOA(3)	TNFDAT(2)/TDATC(2)			SPDIFB								
GPIOA(4)	TNFDAT(3)/TDATC(3)	DREQBb				OOBLK						
GPIOA(5)	TNFDAT(4)/TDATC(4)	I2SSCK_OUTB				OOBDAT						
GPIOA(6)	TNFDAT(5)/TDATC(5)	I2SWS_OUTB				OOBSTRT						
GPIOA(7)	TNFDAT(6)/TDATC(6)	I2SSD_OUTB				OOBVLD						
GPIOA(8)	TNFDAT(7)/TDATC(7)	I2SSOSCK_OUTB				OOBERA						
GPIOA(9)	TNFEN/TSTRTC											
GPIOA(10)	TNFSYNC/TVLDC											
GPIOA(11)	TNFRW/TERC											
GPIOA(12)	TNFAV											
GPIOA(13)			DREQAb		DVOHPD		VIPIRQ					
GPIOB(0)					DVOCNTRL3							
GPIOB(1)					DVOCLK0							
GPIOB(2)					DVOCLK1					CRTDDCDATA		
GPIOB(3)					DVODE					CRTDDCCLK		
GPIOB(4)					DVOHSYNC					HSYNC		
GPIOB(5)					DVOVSYNC					VSYNC		
GPIOB(6)					DVODATA(0)							
GPIOB(7)					DVODATA(1)			HREF/DVSCLKB	DVSCLKC			CTSAb
GPIOB(8)					DVODATA(2)			DVSDATB(0)	DVSDATC(0)			DCDAb
GPIOB(9)					DVODATA(3)			DVSDATB(1)	DVSDATC(1)			DSRAb
GPIOB(10)					DVODATA(4)			DVSDATB(2)	DVSDATC(2)			DTRAb
GPIOB(11)					DVODATA(5)			DVSDATB(3)	DVSDATC(3)			RIAb
GPIOB(12)					DVODATA(6)			DVSDATB(4)	DVSDATC(4)			RTSAb
GPIOB(13)					DVODATA(7)			DVSDATB(5)	DVSDATC(5)		TRSTb	RDA
GPIOB(14)					DVODATA(8)			DVSDATB(6)	DVSDATC(6)		TMS	TDA
GPIOB(15)					DVODATA(9)			DVSDATB(7)	DVSDATC(7)		TCK	
GPIOB(16)					DVODATA(10)				DVSDATC(8)		TDI	
GPIOB(17)					DVODATA(11)			VREF	DVSDATC(9)		TDO	
GPIOB(18)					DVIDDCDATA						DINT	
GPIOB(19)					DVIDDCCLK						TAPSEL	

Notes: The register to control this table is HBIU.GPIO_SEL.GPIO_SELECTION

- Set GPIO_SELECTION(15) = '1' to turn on the GPIOA(12) to clk_debug_out.
- Set GPIO_SELECTION(16) = '1' to choose DREQBb coming in from GPIOA(6) instead of GPIOA(4).
- Set GPIO_SELECTION(17) = '1' to select ITU-656 port B input from PCI pads instead of GPIOB pads.
- Set HBIU.TEST_DEBUG_CNTL.TEST_DELAY_EN = '1' to force GPIOA(11) as delay chain input, GPIOA(12) as delay chain output.
- Set TPS.TPS_TNF_HSDP_CFG.SERIAL_MODE_EN = '1' to turn on the serial mode for 1394 and transport OUT.
- GPIO_SELECTION(1) is used for multiplexing 656 capture port B with FlexBus and IR ports (only applies to A21 and higher revisions of Xilleon 220).

GPIO_SELECTION(11) is not in used. The EJTAG control is inside the IO block.

9.3.16 VIP Interface

Table 9-25 VIP Interface

Signal Name	Type	Description
VIPCLK	O	VIP Host Clock, **OR** general purpose IO
VIPHAD(1:0)	I/O	VIP Host Address/Data Bus, **OR** general purpose IO
VIPHCTL	I/O	VIP Host Control, **OR** general purpose IO
VIPIRQ*	I	VIP Host Interrupt

* This signal is multiplexed in the General Purpose Bus
see "GPIO Pins Multiplexing" on page 9-16

9.3.17 Test Pin

Table 9-26 Test Pin

Signal Name	Type	Description
TESTEN	I	Test mode enable

9.3.18 Serial Port Interface

Note: The first seven pins are Port A pins. They are multiplexed in both the General Purpose bus and the LPC (see *Table 9-29, "Serial Port and LPC Pins Multiplexing," on page 9-18*). The remaining pins are Port B pins, they are multiplexed in PCI, except RDB and TDB.

Table 9-27 Serial Port Pins

Signal Name	Type	Description
CTSAb	I	Clear to Send
DCDAb	I	Data Carrier Detect
DSRAb	I	Data Set Ready
DTRAb	O	Data Terminal Ready
RIAb	I	Ring Indicator
RTSAb	O	Request to Send
RDA	I	Serial Receive Data
TDA	O	Serial Transmit Data
CTSBb	I	Clear to Send
DCDBb	I	Data Carrier Detect
DSRBb	I	Data Set Ready
DTRBb	O	Data Terminal Ready

Table 9-27 Serial Port Pins (Continued)

Signal Name	Type	Description
RIBb	I	Ring Indicator
RTSBb	O	Request to Send
RDB	I	Serial Receive Data (Port B)
TDB	O	Serial Transmit Data (Port B)

9.3.19 LPC Interface

Table 9-28   LPC Interface

Signal Name	Type	Description
LAD(3:0)	I/O	Multiplexed Command/Address/Data
LFRAMEb	I/O	Frame Indicator
LCLK	O	LPC Clock
LDRQb(1:0)	I	Encoded DMA/Bus Master Request
SERIRQb	I/O	Serial Interrupt Request

9.3.20 Serial Port A and LPC Multiplexing

Table 9-29 Serial Port and LPC Pins Multiplexing

Bit - 5	PCI_SELECT
Serial Port A	**LPC**
CTSb	LAD(0)
DCDb	LAD(1)
DSRb	LAD(2)
RIb	LAD(3)
DTRb	LFRAMEb
RTSb	LCLK
RDA	LDRQb(0)
TDA	LDRQb(1)
	SIRIRQb

Note: Set HBIU.PCI_SELECT.PCI_SELECTION(5) = '1' to use Serial Port A on LPC pads.

9.3.21 PCI Bus Multiplexing

Table 9-30 PCI Bus Multiplexing

PCI_SEL	0	1	2	3	4
Host (PCI)	ITU-656 (Port B - IN)	ITU-656 (Port C - OUT)	DVI IN (reserved only)*	TIMER PORT	Serial Port B
RESETb	No multiplexing				
RESETOUTb					DTRBb
PCLCLK	No multiplexing				
PCICLKOUT(0)	DVSYDATB(0)				
PCICLKOUT(1)	DVSCLKB/HREF				
PCICLKOUT(2)					RTSBb
PCICLKOUT(3)					RIBb
AD(0)				CAPT3	
AD(1)				COMP3	
AD(2)			HDVIDDATA(23)		
AD(3)			HDVIDDATA(22)		
AD(4)			HDVIDDATA(21)		
AD(5)			HDVIDDATA(20)		
AD(6)			HDVIDDATA(19)		
AD(7)			HDVIDDATA(18)		
AD(8)			HDVIDDATA(16)		
AD(9)			HDVIDDATA(15)		
AD(10)			HDVIDDATA(14)		
AD(11)			HDVIDDATA(13)		
AD(12)			HDVIDDATA(12)		
AD(13)			HDVIDDATA(11)		
AD(14)			HDVIDDATA(10)		
AD(15)			HDVIDDATA(9)		
AD(16)			HDVIDVSYNC		
AD(17)			HDVIDDE		
AD(18)			HDVIDCLK		
AD(19)		DVSYDATC(9)			
AD(20)		DVSYDATC(8)			
AD(21)		DVSYDATC(7)			
AD(22)		DVSYDATC(6)			
AD(23)		DVSYDATC(5)			
AD(24)		DVSYDATC(3)			
AD(25)		DVSYDATC(2)			
AD(26)		DVSYDATC(1)			
AD(27)		DVSYDATC(0)			
AD(28)		DVSCLKC			
AD(29)	VREF				
AD(30)					
AD(31)	DVSYDATB(7)				
CBEb(0)			HDVIDDATA(17)		
CBEb(1)			HDVIDDATA(8)		
CBEb(2)			HDVIDHSYNC		

Table 9-30 PCI Bus Multiplexing (Continued)

PCI_SEL	0	1	2	3	4
Host (PCI)	**ITU-656 (Port B - IN)**	**ITU-656 (Port C - OUT)**	**DVI IN (reserved only)***	**TIMER PORT**	**Serial Port B**
CBEb(3)		DVSYDATC(4)			
FRAMEb			HDVIDDATA(0)		
IRDYb			HDVIDDATA(1)		
TRDYb			HDVIDDATA(2)		
DEVSELb			HDVIDDATA(3)		
STOPb			HDVIDDATA(4)		
PAR			HDVIDDATA(7)		
INTROUTb	No multiplexing				
INTRb(0)					DSRBb
INTRb(1)					DCDBb
INTRb(2)					CTSBb
REQb(0)	DVSYDATB(6)				
REQb(1)	DVSYDATB(5)				
REQb(2)	DVSYDATB(4)				
IDSEL	No multiplexing				
PERRb			HDVIDDATA(5)		
SERRb			HDVIDDATA(6)		
GNTb(0)	DVSYDATB(3)				
GNTb(1)	DVSYDATB(2)				
GNTb(2)	DVSYDATB(1)				

* DVI IN function is not implemented in any of the Xilleon 220 revisions. The pin allocation for DVI IN above only serves as a place holder for potential future chip revisions.

Note: The register to control this table is HBIU.PCI_SELECT.PCI_SELECTION.

9.3.22 FlexBus Interface

Table 9-31 FlexBus Interface

Signal Name	Type	Description
FCLK	O	Bus Clock
FAD(15:0)	I/O	Triple-Multiplexed Address/Data
FBE(1:0)	O	Byte Enables
FALE(1:0)	O	Address Latch Enable: - 0 = A[15:0] - 1 = A[31:16]
FSTB	O	Strobe, aka Transfer Start
FRDYb	I	Data Ready, aka Transfer Ack
FRD	O	Read
FWE	O	Write Enable
FCEb(5:0)	O	Chip Enables

Table 9-31 FlexBus Interface (Continued)

Signal Name	Type	Description
FREQb	I	Request for Bus Ownership
FGNTb	O	Grant of Bus Ownership
FINTRb(7:0)	I	Interrupt Inputs These 8 signals are multiplexed with Smart Card pins, i.e., FINTRb(0) > NRSSDATA_INA FINTRb(1) > NRSSDATA_OUTA FINTRb(2) > NRSSDATA_INB FINTRb(3) > NRSSDATA_OUTB FINTRb(4) > SMRSTS FINTRb(5) > SMVCCS FINTRb(6) > SMDETECTS FINTRb(7) > SMDATAS

9.3.23 USB Interface

Table 9-32 USB Interface

Signal Name	Type	Description
USBPA	I/O	USB Port A positive
USBNA	I/O	USB Port A negative
USBPB	I/O	USB Port B positive
USBNB	I/O	USB Port B negative
USB_OVRCUR	I	Over-Current Indicator

9.3.24 IEEE 1394 Serial Interface

Note: These signals are multiplexed in the General Purpose Bus (*see "GPIO Pins Multiplexing" on page 9-16*).

Table 9-33 1394 Interface

Signal Name	Type	Description
TNFCLK	O	1394 Clock
TNFDAT(0)	I/O	1394 Serial/Parallel Data
TNFDAT(7:1)	I/O	1394 Parallel Data
TNFEN	O	1394 Enable
TNFSYNC	O	1394 Sync to indicate start of transport packet
TNFRW	O	1394 read/write (1 – Xilleon 220 read)
TNFAV	I	1394 data available

9.3.25 Out of Band Interface

Note: These signals are multiplexed in the General Purpose Bus (*see "GPIO Pins Multiplexing" on page 9-16*).

Table 9-34 Out of Band Interface

Signal Name	Type	Description
OOBLK	I	Serial/Parallel Clock
OOBDAT	I	Serial/Parallel Data In
OOBSTRT	I	Stream Start
OOBVLD	I	Stream Valid
OOBERA	I	Stream Error

9.3.26 Infrared Receive/Transmit Interface

Table 9-35 IR Interface

Signal Name	Type	Description
IRRXD	I	Infrared Receive Data
IRTXD	O	Infrared Transmit Data

9.3.27 IDE Interface

Table 9-36 IDE Interface

Signal Name	Type	Description
IDE_DA(2:0)	O	Device Address
IDE_CSb(1:0)	O	Chip Select
IDE_DD(15:0)	I/O	Data Bus
IDE_IORb	O	I/O Read
IDE_IOWb	O	I/O Write
IDE_IORDY	I	I/O Ready
IDE_DMARQ	I	DMA Request
IDE_DMACKb	O	DMA Acknowledge
IDE_INTRQ	I	Interrupt Request Input

9.3.28 EJTAG Interface

Note: These signals are multiplexed in the General Purpose Bus (*see "GPIO Pins Multiplexing" on page 9-16*).

Table 9-37 EJTAG Interface

Signal Name	Type	Description
TRSTb	I	Test Reset, resets the TAP controller
TMS	I	Test Mode Select
TCK	I	Test Clock
TDI	I	Test Data Input
TDO	O	Test Data Output
DINT	I	Debug Interrupt Request
TAPSEL	I	Tap Controller Select (only activated when TESTEN = 1) 0 = native Xilleon 220 tap controller 1 = MIPS' EJTAG tap controller

9.3.29 Power and Ground Pins

Table 9-38 Power and Ground Pins

Signal Name	Type	Description
VDDP	I	Powering PCI 33/66 IOs. **3.3V**
VDDC	I	Powering ASIC core logic. **1.2V/1.5V**
VDDR1		Powering memory. **2.5V/3.3V**
VDDR3	I	Powering multi-media IOs. **3.3V**
VCC18	I	Powering all I/Os. **1.8 V**

Table 9-38 Power and Ground Pins (Continued)

Signal Name	Type	Description
VSS	I	Common digital ground

9.4 Configuration/Straps

Table 9-39 Straps

Strap Name	Pin Name	Function	Default
ENDIAN	FAD(0)	Controls the endian mode of the internal MIPS processor. 0 – Little Endian 1 – Big Endian	0 (internal pull-down)
MERGE_MODE	FAD(1)	0 – No merge 1 – Full merge	0 (internal pull-down)
EB_BLKTYPE	FAD(2)	Leave as no connect. Sets block type to sub-block ordering.	0 (internal pull-down)
BOOTROM_DBUSWID	FAD(3)	Indicates the data bus width of the BootROM residing on the FlexBus. 0 – 8-bit 1 – 16-bit	0 (internal pull-down)
BOOTROM_ENABLE	FAD(4)	Controls whether the HBIU's PCI target state machine will respond to PCI cycles to the "Boot Aperture" before it has gone through PCI configuration. 0 – ROM Enabled 1 – ROM Disabled	0 (internal pull-down)
BOOT_VECTOR(1:0)	FAD(6:5)	Controls the address range that the HBIU's PCI target state machine will respond to for the "Boot Aperture" (which can be enabled by the BOOTROM_ENABLE strap before PCI configuration.) 00 – MIPS 01 – x86-Family 10 – SH-Family 11 – StrongARM/ARM Family	00 (internal pull-down)
SYS_CFG(1:0)	FAD(8:7)	System Configuration. 00 – SOLO (no external SB chip support) 01 – SOLO- (external SB chip support) 10 – PEER (w/ peripherals i.e. no external SB chip support) 11 – PEER+ (w/o peripherals i.e. external SB chip support)	00 (internal pull-down)
PCI_66	FAD(9)	Indicates the Bus Speed of the external PCI bus. 0 – PCI 33Mhz (33 MHz or less) 1 – PCI 66Mhz (above 33, up to 66 MHz)	0 (internal pull-down)

Table 9-39 Straps (Continued)

Strap Name	Pin Name	Function	Default
IPCI_CONFIG_DIS	FAD(10)	Internal PCI Devices (IDE, USB, LPC, DAIO) Disabled from responding to PCI Configuration-space cycles. (Primarily intended for debugging). 0 – IDE, USB, LPC, and DAIO respond to PCI Config cycles normally. 1 – IDE, USB, LPC, and DAIO only respond to PCI Config cycles if their associated ‘EN_xxx_CONFIG’ bit is a ‘1’ in the PCIC_DEBUG_CNTL register.	0 (internal pull-down)
PCI_MUX	FAD(11)	0 – PCI muxing disabled (only PCI function enabled) 1 – PCI muxing enabled (PCI function disabled)	0 (internal pull-down)
USBNOPWRSW_CLR	FAD(12)	0 – Set USB NoPowerSwitching bit on reset 1 – Reset USB NoPowerSwitching bit on reset	0 (internal pull-down)
BOOTMIPSONHARDRESET	FAD(13)	Boot the Internal MIPS Upon Hard Reset 0 – MIPS comes out of reset under the control of a specific bit in the MIPS_CNTL register. (see the register spec) 1 – MIPS comes out of reset when the RESETb input pin becomes de-asserted.	0 (internal pull-down)
PCIMEMAPERSIZE	FAD(14)	PCI Memory Aperture Size Controls the size of two of the PCI memory-space apertures. 0 – HBIU_MEM_BASE:128MB, HBIU_PCU_BASE: 64MB 1 – HBIU_MEM_BASE: 64MB, HBIU_PCU_BASE: 32MB	0 (internal pull-down)

9.5 Default Internal Pull-up/down Resistor Mapping

The following is the pull-up/down mapping table for all pins in the Xilleon 220.

Table 9-40 Pulup/down Mapping Table

Signal Name	Pull-up/down	Signal Name	Pull-up/down
AA(0)	down	AD(10)	HZ
AA(1)	down	AD(11)	HZ
AA(10)	down	AD(12)	HZ
AA(11)	down	AD(13)	HZ
AA(12)	down	AD(14)	HZ
AA(13)	down	AD(15)	HZ
AA(14)	down	AD(16)	HZ
AA(2)	down	AD(17)	HZ
AA(3)	down	AD(18)	HZ
AA(4)	down	AD(19)	HZ
AA(5)	down	AD(2)	HZ
AA(6)	down	AD(20)	HZ
AA(7)	down	AD(21)	HZ
AA(8)	down	AD(22)	HZ
AA(9)	down	AD(23)	HZ
AB(0)	down	AD(24)	HZ
AB(1)	down	AD(25)	HZ
AB(10)	down	AD(26)	HZ
AB(11)	down	AD(27)	HZ
AB(12)	down	AD(28)	HZ
AB(13)	down	AD(29)	HZ
AB(14)	down	AD(3)	HZ
AB(2)	down	AD(30)	HZ
AB(3)	down	AD(31)	HZ
AB(4)	down	AD(4)	HZ
AB(5)	down	AD(5)	HZ
AB(6)	down	AD(6)	HZ
AB(7)	down	AD(7)	HZ
AB(8)	down	AD(8)	HZ
AB(9)	down	AD(9)	HZ
ACBIT_CLK	down	CASAb	down
ACDATA_IN(0)	down	CASBb	down
ACDATA_IN(1)	down	CBEb(0)	HZ
ACDATA_OUT	down	CBEb(1)	HZ
ACRESETb	down	CBEb(2)	HZ
ACSYNC	down	CBEb(3)	HZ
ACXTAL_OUT	down	CKEA	down
AD(0)	HZ	CKEB	down
AD(1)	HZ	CLKA	none

Table 9-40 Pulup/down Mapping Table (Continued)

Signal Name	Pull-up/down	Signal Name	Pull-up/down
CLKAb	none	DQA(8)	down
CLKB	none	DQA(9)	down
CLKBb	none	DQB(0)	down
CLKFBA	none	DQB(1)	down
CLKFBAb	none	DQB(10)	down
CLKFBB	none	DQB(11)	down
CLKFBBb	none	DQB(12)	down
CSAb(0)	down	DQB(13)	down
CSAb(1)	down	DQB(14)	down
CSBb(0)	down	DQB(15)	down
CSBb(1)	down	DQB(16)	down
CWIV_READ	down	DQB(17)	down
DEVSELb	up	DQB(18)	down
DQA(0)	down	DQB(19)	down
DQA(1)	down	DQB(2)	down
DQA(10)	down	DQB(20)	down
DQA(11)	down	DQB(21)	down
DQA(12)	down	DQB(22)	down
DQA(13)	down	DQB(23)	down
DQA(14)	down	DQB(24)	down
DQA(15)	down	DQB(25)	down
DQA(16)	down	DQB(26)	down
DQA(17)	down	DQB(27)	down
DQA(18)	down	DQB(28)	down
DQA(19)	down	DQB(29)	down
DQA(2)	down	DQB(3)	down
DQA(20)	down	DQB(30)	down
DQA(21)	down	DQB(31)	down
DQA(22)	down	DQB(4)	down
DQA(23)	down	DQB(5)	down
DQA(24)	down	DQB(6)	down
DQA(25)	down	DQB(7)	down
DQA(26)	down	DQB(8)	down
DQA(27)	down	DQB(9)	down
DQA(28)	down	DQMAb(0)	down
DQA(29)	down	DQMAb(1)	down
DQA(3)	down	DQMAb(2)	down
DQA(30)	down	DQMAb(3)	down
DQA(31)	down	DQMBb(0)	down
DQA(4)	down	DQMBb(1)	down
DQA(5)	down	DQMBb(2)	down
DQA(6)	down	DQMBb(3)	down
DQA(7)	down	FAD(0)	down

Table 9-40 Pulup/down Mapping Table (Continued)

Signal Name	Pull-up/down	Signal Name	Pull-up/down
FAD(1)	down	GPIOA(3)	down
FAD(10)	down	GPIOA(4)	up
FAD(11)	down	GPIOA(5)	up
FAD(12)	down	GPIOA(6)	up
FAD(13)	down	GPIOA(7)	up
FAD(14)	down	GPIOA(8)	up
FAD(15)	down	GPIOA(9)	down
FAD(2)	down	GPIOB(0)	down
FAD(3)	down	GPIOB(1)	down
FAD(4)	down	GPIOB(10)	down
FAD(5)	down	GPIOB(11)	down
FAD(6)	down	GPIOB(12)	down
FAD(7)	down	GPIOB(13)	down
FAD(8)	down	GPIOB(14)	down
FAD(9)	down	GPIOB(15)	down
FALE(0)	up	GPIOB(16)	down
FALE(1)	up	GPIOB(17)	down
FBE(0)	up	GPIOB(18)	down
FBE(1)	up	GPIOB(19)	down
FCEb(0)	up	GPIOB(2)	down
FCEb(1)	up	GPIOB(3)	down
FCEb(2)	up	GPIOB(4)	down
FCEb(3)	up	GPIOB(5)	down
FCEb(4)	up	GPIOB(6)	down
FCEb(5)	up	GPIOB(7)	down
FCLK	up	GPIOB(8)	down
FGNTb	up	GPIOB(9)	down
FRAMEb	up	I2CCLKA	up
FRD	up	I2CCLKB	up
FRDYb	up	I2CDATAA	up
FREQb	up	I2CDATAB	up
FSTB	up	I2SSCK_IN	down
FWE	up	I2SSCK_OUTA	down
GNTb(0)	up	I2SSD_IN	down
GNTb(1)	up	I2SSD_OUTA	down
GNTb(2)	up	I2SSOSCK_OUTA	down
GPIOA(0)	down	I2SWS_IN	down
GPIOA(1)	down	I2SWS_OUTA	down
GPIOA(10)	down	IDE_CSb(0)	up
GPIOA(11)	down	IDE_CSb(1)	up
GPIOA(12)	down	IDE_DA(0)	down
GPIOA(13)	down	IDE_DA(1)	down
GPIOA(2)	down	IDE_DA(2)	down

Table 9-40 Pulup/down Mapping Table (Continued)

Signal Name	Pull-up/down	Signal Name	Pull-up/down
IDE_DD(0)	down	NRSSDATA_OUTB	down
IDE_DD(1)	down	PAR	down
IDE_DD(10)	down	PCICLK	none
IDE_DD(11)	down	PCICLKOUT(0)	down
IDE_DD(12)	down	PCICLKOUT(1)	down
IDE_DD(13)	down	PCICLKOUT(2)	down
IDE_DD(14)	down	PCICLKOUT(3)	down
IDE_DD(15)	down	PERRb	up
IDE_DD(2)	down	PWMA	down
IDE_DD(3)	down	PWMB	down
IDE_DD(4)	down	QSA(0)	down
IDE_DD(5)	down	QSA(1)	down
IDE_DD(6)	down	QSA(2)	down
IDE_DD(7)	down	QSA(3)	down
IDE_DD(8)	down	QSB(0)	down
IDE_DD(9)	down	QSB(1)	down
IDE_DMACKb	up	QSB(2)	down
IDE_DMARQ	down	QSB(3)	down
IDE_INTRQ	down	RASAb	down
IDE_IORb	down	RASBb	down
IDE_IORDY	up	RDB	down
IDE_IOWb	down	REQb(0)	up
IDSEL	none	REQb(1)	up
INTRb(0)	up	REQb(2)	up
INTRb(1)	up	RESETb	none
INTRb(2)	up	RESEToutb	none
INTROUTb	none	SERIRQb	up
IRDYb	up	SERRb	up
IRRXD	up	SMCLKA	down
IRTXD	down	SMCLKB	down
LAD(0)	down	SMCLKS	down
LAD(1)	down	SMDATAA	down
LAD(2)	down	SMDATAB	down
LAD(3)	down	SMDATAS	down
LCLK	down	SMDETECTA	down
LDRQb(0)	up	SMDETECTB	down
LDRQb(1)	up	SMDETECTS	down
LFRAMEb	up	SMRSTA	down
NRSSCLKA	down	SMRSTB	down
NRSSCLKB	down	SMRSTS	down
NRSSDATA_INA	down	SMVCCA	down
NRSSDATA_INB	down	SMVCCB	down
NRSSDATA_OUTA	down	SMVCCS	down

Table 9-40 Pulup/down Mapping Table (Continued)

Signal Name	Pull-up/down	Signal Name	Pull-up/down
STOPb	up		
TCLKA	down		
TCLKB	down		
TDATA(0)	down		
TDATA(1)	down		
TDATA(2)	down		
TDATA(3)	down		
TDATA(4)	down		
TDATA(5)	down		
TDATA(6)	down		
TDATA(7)	down		
TDATB(0)	down		
TDATB(1)	down		
TDATB(2)	down		
TDATB(3)	down		
TDATB(4)	down		
TDATB(5)	down		
TDATB(6)	down		
TDATB(7)	down		
TDB	down		
TERA	down		
TERB	down		
TESTEN	none		
TRDYb	up		
TSTRTA	down		
TSTRTB	down		
TVLDA	down		
TVLDB	down		
USB_OVRCUR	up		
USBNA	none		
USBNB	none		
USBPA	none		
USBPB	none		
VCXO_INA	down		
VCXO_INB	down		
VIPCLK	up		
VIPHAD(0)	up		
VIPHAD(1)	up		
VIPHCTL	up		
WEAb	down		
WEBb	down		

Chapter 10
Electrical and Physical Characteristics

10.1 DC/AC Characteristics

The signal pins on Xilleon are divided into several groups: power, analog, memory and buffer.

The data provided below pertain to the buffer and memory signals.

- Buffer signals: PCI, smart card, 1394, OOB, transport, AC link, SPDIF, I2S, I2C, DVI, ITU-656, GPIO, VIP, LPC, Flexbus, USB, IR, IDE, Serial, Timers, EJTAG.
- Memory signals: All signals relating to memory, including address, data, control and clock.

10.1.1 DC Characteristics

Buffer:

- Input Capacitance = 5pf (typ)
- Vil = 2.0V (min)
- Vih = 0.8V (max)
- Output Capacitance = 1.3pf + 1.0pf = 2.3pf (max)
- Vol = 0.4V @ 10.26mA (max)
- Voh = 2.4V @ -23.28mA (min)

Memory:

- Input Capacitance = 5pf (typ)
- Vil = Vref-300mV (min)
- Vih = Vref+300mV (max)
- Output Capacitance = 1.3pf + 1.0pf = 2.3pf (max)
- Vol = 0.45V @ 9.04mA (max)
- Voh = 1.7V @ -26mA (min)

10.1.2 AC Characteristics:

Buffer:

- Rise time = 3.1ns @ Cl = 20pf (typ)
- Fall time = 3.1ns @ Cl = 20pf (typ)

Memory:

- Rise time = 2.4ns @ Cl = 20pf (typ)
- Fall time = 2.4ns @ Cl = 20pf (typ)

10.2 Power Dissipation

Power dissipation for Xilleon 220 is dependent on the applications being run. The following two examples illustrate the power dissipation for a couple of different scenarios. Data on additional scenarios, as well as further detailed information on the test setups, can be obtained by contacting ATI.

10.2.1 Test Conditions

Environment

- Room temperature ambient: 25°C
- Elevated temperature ambient: 70°C

Clock Speeds

The following clock speeds are used for all tests unless otherwise noted:

- MCLK: 100MHz
- XCLK for SDR configurations: 166MHz
- XCLK for DDR configurations: 183MHz
- MIPS clk: 250MHz for test #1, 300MHz for test #2
- PCU pll: 530MHz
- PCI clk: 25MHz
- USB clk: 33MHz
- SIR clk: 33MHz
- FBC clk: 33MHz
- IDE clk: 33MHz

Supplies Monitored for Current Measurements

- VDDC (Core voltage) – 1.5V
- VDDP (PCI I/O supply voltage) – 3.3V
- VDDR1 (memory supply voltage) – 3.3V
- VDDR3 (multi-media I/O supply voltage) – 3.3V
- VDD18 (I/O supply voltage) – 1.8V
- VDDDI (DAC macro digital supply voltage) – 1.8V
- AVDD (DAC macro supply voltage) – 2.5V
- XMVDD (Memory/engine PLL supply voltage) – 1.8V
- APVDD (Audio/SB PLL supply voltage) – 1.8V
- PMIVDD (Display/MIPS PLL supply voltage) – 1.8V

Power Supply Notes

- VDDR1 set to 3.3V for SDR configuration and 2.5V for DDR configuration
- 3.3V supply derived from dedicated on board regulator for SDR board
- 3.3V supply derived from PC power supply for DDR board

Memory Configuration

- Memory Configuration 1: SDR dual channel 32 bit wide
- Memory Configuration 2: SDR single channel 64 bit wide
- Memory Configuration 3: DDR dual channel 32 bit wide

10.2.2 Test #1 - Dual Display SDTV – MPEG Only

Test Setup

- Both primary and secondary displays active – SD on both (NTSC)
- Primary TVOut connected to display via S-VIDEO
- Secondary TVOut connected to display via composite
- Primary stream feed over PCI bus – toystory.pes (480i SD source)
- Secondary stream feed over PCI bus – mummycut.pes (480i SD source)
- 2 MPEG engines active
- 2 video overlays active
- 2 stream buffers active

Average Power Dissipation

Table 10-1 Test Setup #1 - Dual Display SDTV - MPEG Only - Average Power Dissipation

	SDR Dual Channel 32 Bit			SDR Single Channel 64 Bit			DDR Dual Channel 32 Bit		
	V (V)	I (mA)	P (W)	V (V)	I (mA)	P (W)	V (V)	I (mA)	P (W)
VDDC (1.5V)	1.5	1010.49	1.52	1.5	1010.61	1.52	1.5	1049.91	1.57
VDDP (3.3V)	3.4	1.80	0.01	3.4	1.87	0.01	3.1	1.21	0.00
VDDR1 (3.3V)	3.4	32.48	0.11	3.4	34.21	0.12	2.5	60.78	0.15
VDDR3 (3.3V)	3.4	2.88	0.01	3.4	2.95	0.01	3.1	2.38	0.01
VDD18 (1.8V)	2.0	52.67	0.11	2.0	53.37	0.11	2.0	64.11	0.13
VDDDI (1.8V)	2.0	10.88	0.02	2.0	10.93	0.02	2.0	11.10	0.02
AVDD (2.5)	2.5	274.28	0.69	2.5	274.21	0.69	2.5	272.43	0.68
XMVDD (1.8V)	1.8	21.45	0.04	1.8	21.49	0.04	1.8	16.60	0.03
APVDD (1.8V)	1.8	21.16	0.04	1.8	21.16	0.04	1.8	16.25	0.03
PMIVDD (1.8V)	1.8	21.19	0.04	1.8	21.21	0.04	1.8	16.97	0.03
Total Power			**2.57**			**2.58**			**2.66**

Peak Power Dissipation

Table 10-2 Test Setup #1 - Dual Display SDTV - MPEG Only - Peak Power Dissipation

	SDR Dual Channel 32 Bit			SDR Single Channel 64 Bit			DDR Dual Channel 32 Bit		
	V (V)	I (mA)	P (W)	V (V)	I (mA)	P (W)	V (V)	I (mA)	P (W)
VDDC (1.5V)	1.5	1025.79	1.54	1.5	1024.83	1.54	1.5	1062.69	1.59

Table 10-2 Test Setup #1 - Dual Display SDTV - MPEG Only - Peak Power Dissipation

	SDR Dual Channel 32 Bit			SDR Single Channel 64 Bit			DDR Dual Channel 32 Bit		
	V (V)	I (mA)	P (W)	V (V)	I (mA)	P (W)	V (V)	I (mA)	P (W)
VDDP (3.3V)	3.4	1.68	0.01	3.4	1.69	0.01	3.1	0.87	0.00
VDDR1 (3.3V)	3.4	42.33	0.14	3.4	46.87	0.16	2.5	78.04	0.20
VDDR3 (3.3V)	3.4	2.84	0.01	3.4	2.93	0.01	3.1	2.34	0.01
VDD18 (1.8V)	2.0	54.09	0.11	2.0	55.24	0.11	2.0	65.71	0.13
VDDDI (1.8V)	2.0	11.03	0.02	2.0	11.06	0.02	2.0	11.25	0.02
AVDD (2.5)	2.5	274.17	0.69	2.5	274.12	0.69	2.5	272.29	0.68
XMVDD (1.8V)	1.8	21.44	0.04	1.8	21.48	0.04	1.8	16.59	0.03
APVDD (1.8V)	1.8	21.15	0.04	1.8	21.16	0.04	1.8	16.26	0.03
PMIVDD (1.8V)	1.8	21.20	0.04	1.8	21.20	0.04	1.8	16.97	0.03
Total Power			**2.63**			**2.65**			**2.72**

10.2.3 Test #2 - Dual HD/HD with Graphic Overlay – 2D/3D Engine Clk On

Test Setup

- Both primary and secondary displays active – 1st HD on primary (720p) & 2nd HD on secondary (NTSC)
- Primary TVOut connected to display via RGB
- Secondary TVOut connected to display via composite
- Primary stream feed over PCI bus – norway.tsp (1080i HD source)
- Secondary stream feed over PCI bus – tw_ldv.tsp (1080i HD source)
- 2 MPEG engines active
- 2 transport engines active
- 2 stream buffers active
- 2 video overlays active
- Graphic overlay active
- Primary audio out active
- 2D, 3D and RCP active continuously (wr 0xf to address 0x8; set bits 16, 17, 18 at address 0xc)

Average Power Dissipation

Table 10-3 Test Setup #2 - Dual HD/HD with Graphic Overlay - 2D/3D Engine Clk On - Average Power Dissipation

	SDR Dual Channel 32 Bit			SDR Single Channel 64 Bit			DDR Dual Channel 32 Bit		
	V (V)	I (mA)	P (W)	V (V)	I (mA)	P (W)	V (V)	I (mA)	P (W)
VDDC (1.5V)	1.5	1341.70	2.01	1.5	1334.98	2.00	1.5	1303.69	1.96
VDDP (3.3V)	3.4	2.73	0.01	3.4	2.71	0.01	3.2	2.41	0.01

Table 10-3 Test Setup #2 - Dual HD/HD with Graphic Overlay - 2D/3D Engine Clk On - Average Power Dissipation (Continued)

	SDR Dual Channel 32 Bit			SDR Single Channel 64 Bit			DDR Dual Channel 32 Bit		
	V (V)	**I (mA)**	**P (W)**	**V (V)**	**I (mA)**	**P (W)**	**V (V)**	**I (mA)**	**P (W)**
VDDR1 (3.3V)	3.4	92.88	0.32	3.4	79.91	0.27	2.5	143.37	0.36
VDDR3 (3.3V)	3.4	11.37	0.04	3.4	11.30	0.04	3.2	7.78	0.02
VDD18 (1.8V)	2.0	66.56	0.13	2.0	69.00	0.14	2.0	114.72	0.23
VDDDI (1.8V)	2.0	10.46	0.02	2.0	10.46	0.02	2.0	10.89	0.02
AVDD (2.5)	2.5	246.11	0.62	2.5	246.02	0.62	2.5	226.99	0.57
XMVDD (1.8V)	1.8	21.35	0.04	1.8	21.30	0.04	1.8	18.05	0.03
APVDD (1.8V)	1.8	21.08	0.04	1.8	21.04	0.04	1.8	18.52	0.03
PMIVDD (1.8V)	1.8	21.12	0.04	1.8	21.09	0.04	1.8	18.45	0.03
Total Power			**3.26**			**3.21**			**3.26**

Peak Power Dissipation

Table 10-4 Test Setup #2 - Dual HD/HD with Graphic Overlay - 2D/3D Engine Clk On - Peak Power Dissipation

	SDR Dual Channel 32 Bit			SDR Single Channel 64 Bit			DDR Dual Channel 32 Bit		
	V (V)	**I (mA)**	**P (W)**	**V (V)**	**I (mA)**	**P (W)**	**V (V)**	**I (mA)**	**P (W)**
VDDC (1.5V)	1.5	1374.85	2.06	1.5	1372.31	2.06	1.5	1358.03	2.04
VDDP (3.3V)	3.4	3.20	0.01	3.4	2.83	0.01	3.2	2.11	0.01
VDDR1 (3.3V)	3.4	119.39	0.41	3.4	115.84	0.39	2.5	172.39	0.43
VDDR3 (3.3V)	3.4	11.35	0.04	3.4	11.29	0.04	3.2	7.80	0.02
VDD18 (1.8V)	2.0	71.46	0.14	2.0	70.47	0.14	2.0	116.33	0.23
VDDDI (1.8V)	2.0	10.73	0.02	2.0	10.31	0.02	2.0	10.85	0.02
AVDD (2.5)	2.5	246.04	0.62	2.5	246.92	0.62	2.5	227.07	0.57
XMVDD (1.8V)	1.8	21.31	0.04	1.8	21.33	0.04	1.8	17.97	0.03
APVDD (1.8V)	1.8	21.06	0.04	1.8	21.03	0.04	1.8	18.57	0.03
PMIVDD (1.8V)	1.8	21.07	0.04	1.8	21.09	0.04	1.8	18.37	0.03
Total Power			**3.41**			**3.39**			**3.42**

10.3 Power Up Sequence

The Xilleon 220 has a few strict requirements that must be followed when designing the supplies that will power the rails of the ASIC. The main concerns are:

- Protection of the output buffers in the I/O pads.

- Prevention of large current spikes during power supply ramp up via the ESD (electro-static discharge) protection diodes.

Ideally, all of the power supplies should be designed to ramp at the same time. This would ensure correct sequencing and the relative difference between supplies would never be large enough to damage the ASIC or cause large current spikes.

In practical implementations, however, the 3.3V rail or some other higher voltage rail is used to supply the regulator(s) that source the power rails of the Xilleon 220.

In these cases, care must be taken to ensure that the following guidelines are met:

- To avoid large current spikes during the ramp up period, none of the supply voltages must exceed the 3.3V supply rail by more than 0.3V. If any supply does exceed the 3.3V (VDDR3) supply rail by more than 0.3V, this will forward bias the ESD protection diode for that particular rail and a large current may flow to ground depending upon the impedance of the path.
- The 1.8V supply rail (most notably the VDDC18 rail) must not lag the 3.3V supply rail by more than 1.2V when the 3.3V rail reaches 2.0V. This means that when the 3.3V rail is 2.0V the 1.8V rail must be at least 0.8V or more. Subsequently, the 1.8V rail must not dip below 0.8V and must not lag the 3.3V rail by more than 2.1V. Please refer to the *Figure 10-1.* below for clarification.

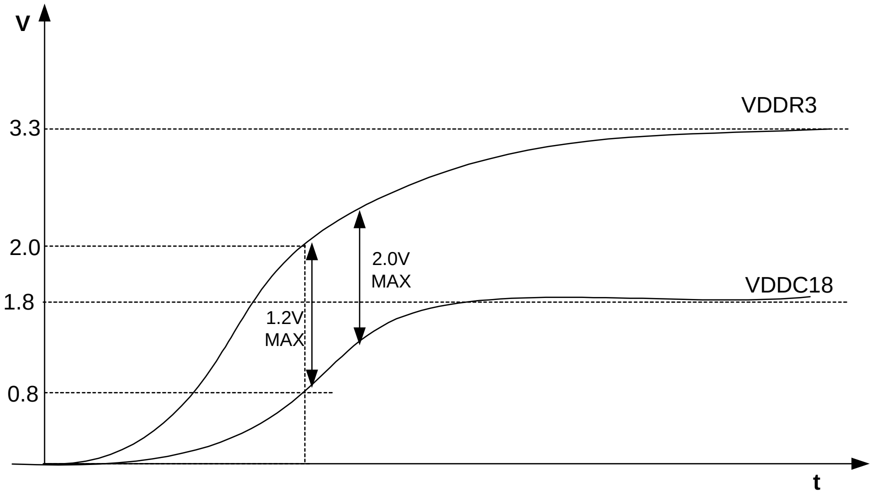


Figure 10-1. Power up sequence when VDDC & VDDC1.8 derive from VDDR

In terms of supply rail sequencing, it is recommended that the 3.3V (VDDR3) supply rail ramp up first. The 1.8V rail should track the 3.3V (VDDR3) rail as described above. There is flexibility as to when all of the other rails can ramp up; however, all rails must reach 90% of their final values no more than 3 ms after the 3.3V (VDDR3) rail reaches 90% of its final value.

10.4 Physical Dimensions

571-Pin BGA package: PBGA 37.5x37.5 mm - 490 + 81

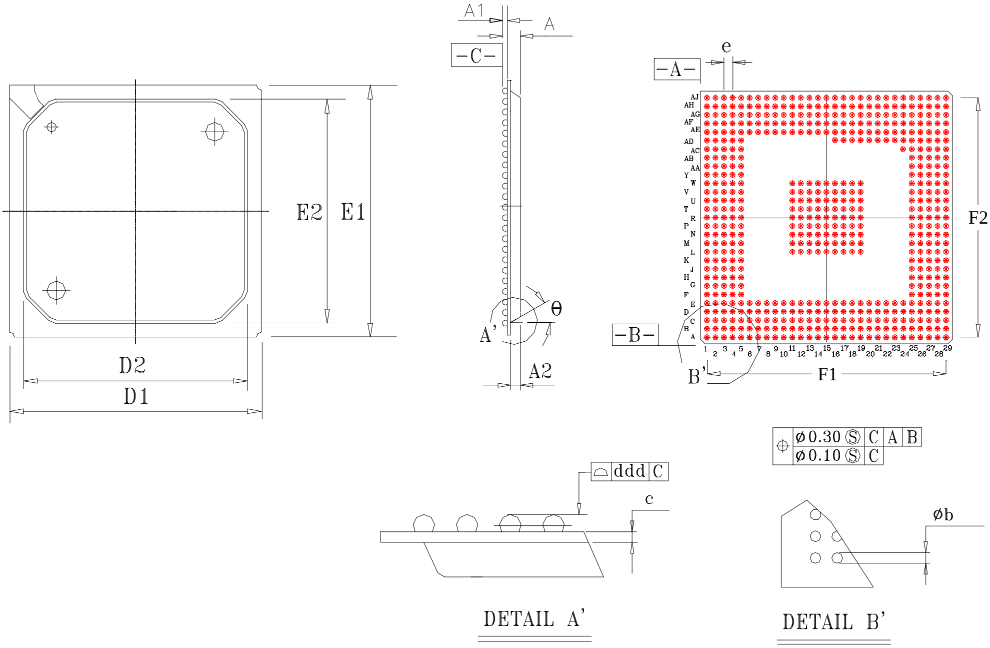


Figure 10-2. Xilleon 220 Physical Dimensions

Table 10-5 Xilleon 220 Physical Dimensions (in mm)

Ref	Min	Normal	Max
c	0.51	0.56	0.61
A	2.20	2.33	2.50
A1	-	0.60	-
A2	1.12	1.17	1.22
ϕb	-	0.75	-
D1	37.30	37.50	37.70
D2	34.30	34.50	34.70
E1	37.30	37.50	37.70
E2	34.30	34.50	34.70
F1	-	35.56	-
F2	-	35.56	-
e	-	1.27	-
ddd	-	-	0.20
θ	-	30 deg typ	-

10.5 Soldering / Reflow Profile

The Xilleon 220 uses a PBGA package that is qualified to JEDEC MSL Level 3. The maximum allowable reflow temperature is 225 degrees C. The following diagram shows the recommended reflow profile. The actual reflow profile used will be determined by the customer, taking into account specific items such as the board design, manufacturing equipment, and manufacturing process variables.

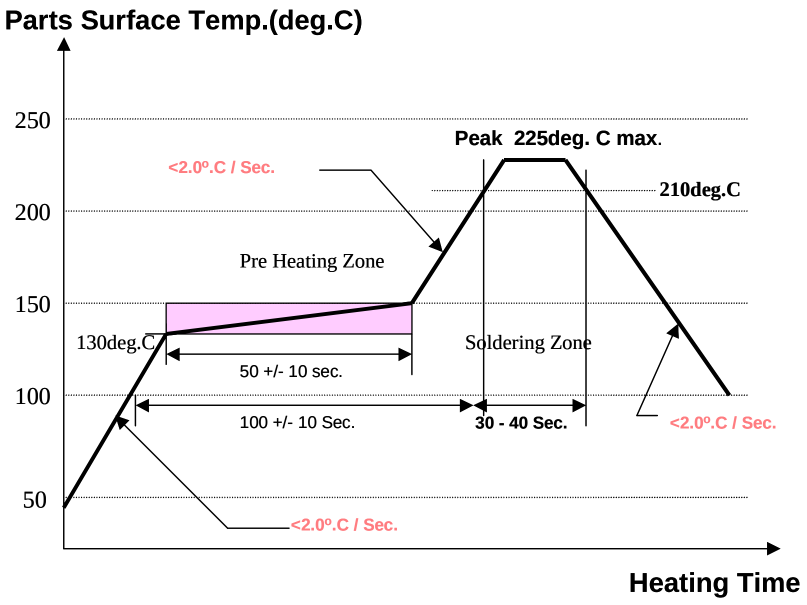


Figure 10-3. Soldering / Reflow Profile

Note: The final reflow profile will depend on the solder paste or flux used in the surface mount assembly process. Modification to this profile may also be required in order to accommodate the requirements of the other components on the circuit board.

Chapter 11
Timing Specifications

This chapter describes timing specifications for Xilleon 220's interfaces. To link to a section of interest, use the following list of linked cross references:

11.1 PCI Bus Timing

Table 11-1 PCI Bus Input and Output Timing Specifications

Signal	Min (ns)	Max (ns)
Clock Timing		
PCICLK and PCICLKOUT(3:0) Cycle Time	15 (66 MHz) 30 (33 MHz)	30 (66 MHz) ∞(33 MHz)
PCICLK and PCICLKOUT(3:0) Clock High Time	6	-
PCICLK and PCICLKOUT(3:0) Clock Low Time	6	-
Input Setup Timing		
Input RESETb Setup Time to PCICLK	N/A	N/A
Input: AD(3:0), CBEb(3:0), FRAMEb, IRDYb, TRDYb, DEVSELb, STOPb, INTRb(2:0), REQb(2:0), IDSEL, PERRb, SERRb, GNTb(0) Setup Time to PCICLK	3	-
Input Hold Timing		
Input RESETb Hold Time from PCICLK	N/A	N/A
Input: AD(3:0), CBEb(3:0), FRAMEb, IRDYb, TRDYb, DEVSELb, STOPb, INTRb(2:0), REQb(2:0), IDSEL, PERRb, SERRb, GNTb(0) Hold Time from PCICLK	0	-
Output Timing		
PCICLK to Output RESETOUTb Valid Delay	N/A	N/A
PCICLK to Outputs: AD(31:0), CBEb(3:0), FRAMEb, IRDYb, TRDYb, DEVSELb, STOPb, PAR, INTROUTb, REQb(0), PERRb, SERRb, GNTb(2:0) Valid Delay	2	6
PCICLK to Output RESETOUTb Active Delay	N/A	N/A
PCICLK to Output: AD(31:0), CBEb(3:0), FRAMEb, IRDYb, TRDYb, DEVSELb, STOPb, PAR, INTROUTb, REQb(0), PERRb, SERRb, GNTb(2:0) Avtive Delay	2 (on)	10 (off)
RESETb Active to Output: AD(31:0), CBEb(3:0), FRAMEb, IRDYb, TRDYb, DEVSELb, STOPb, PAR, INTROUTb, REQb(0), PERRb, SERRb, GNTb(2:0) Float Delay	5	10

11.1.1 Single Read Cycle Timing

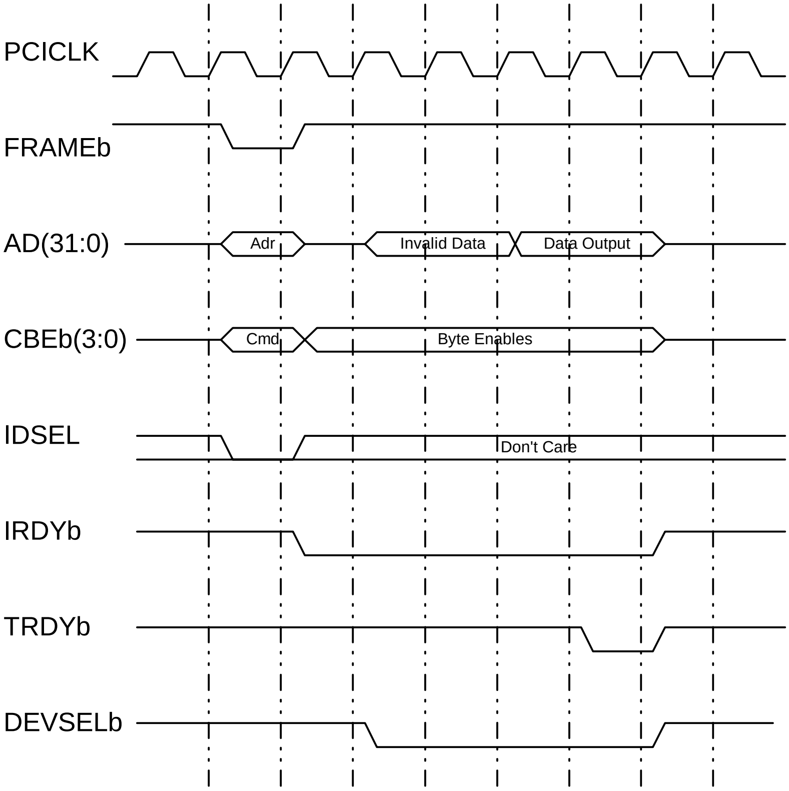


Figure 11-1. PCI Bus Single Read Cycle Time

11.1.2 Single Write Cycle Timing

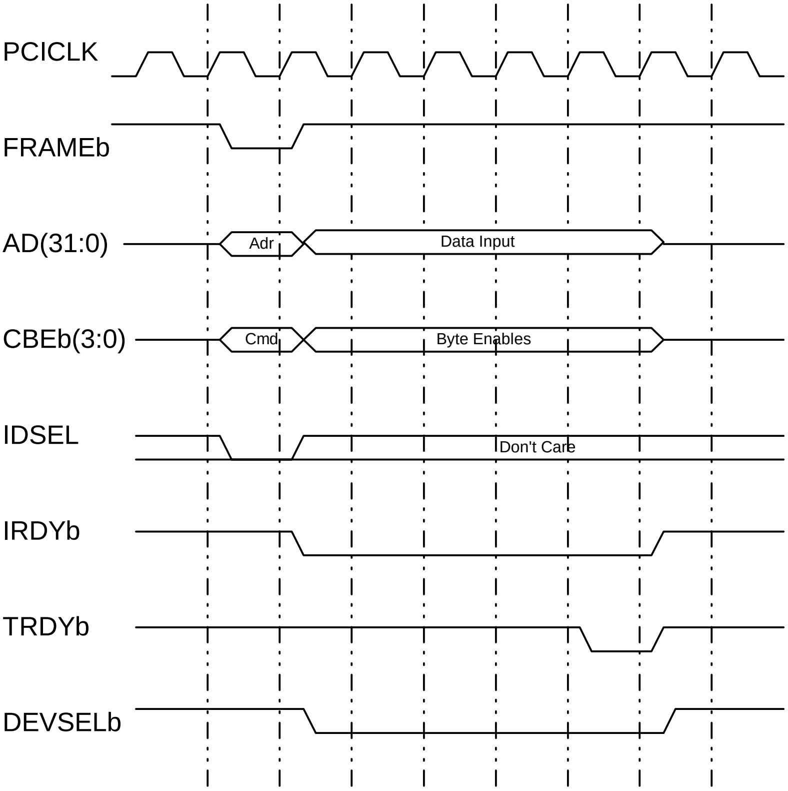


Figure 11-2. PCI Bus Single Write Cycle Time

11.1.3 Burst Read Cycle Timing

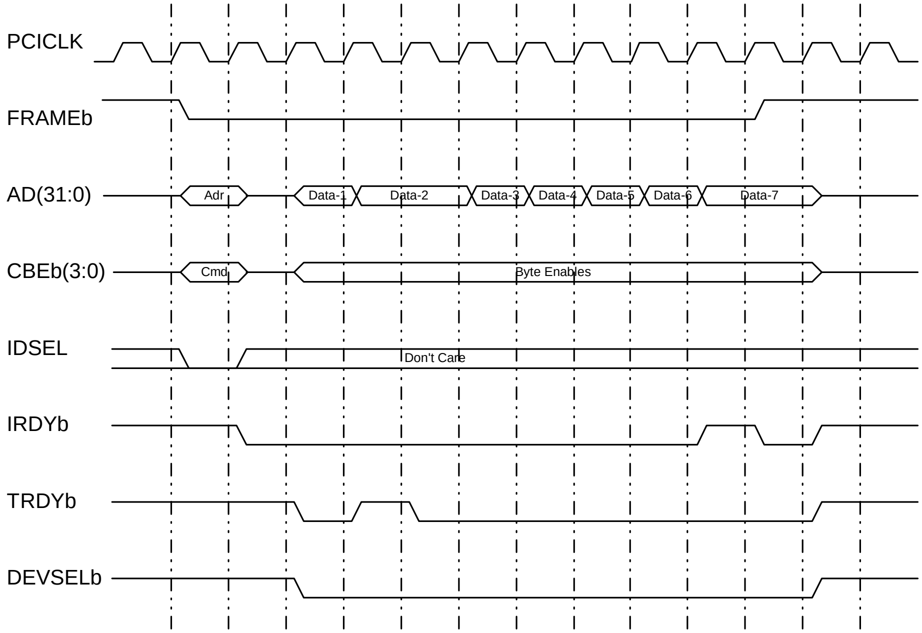


Figure 11-3. PCI Bus Burst Read Cycle Time

11.1.4 Burst Write Cycle Time

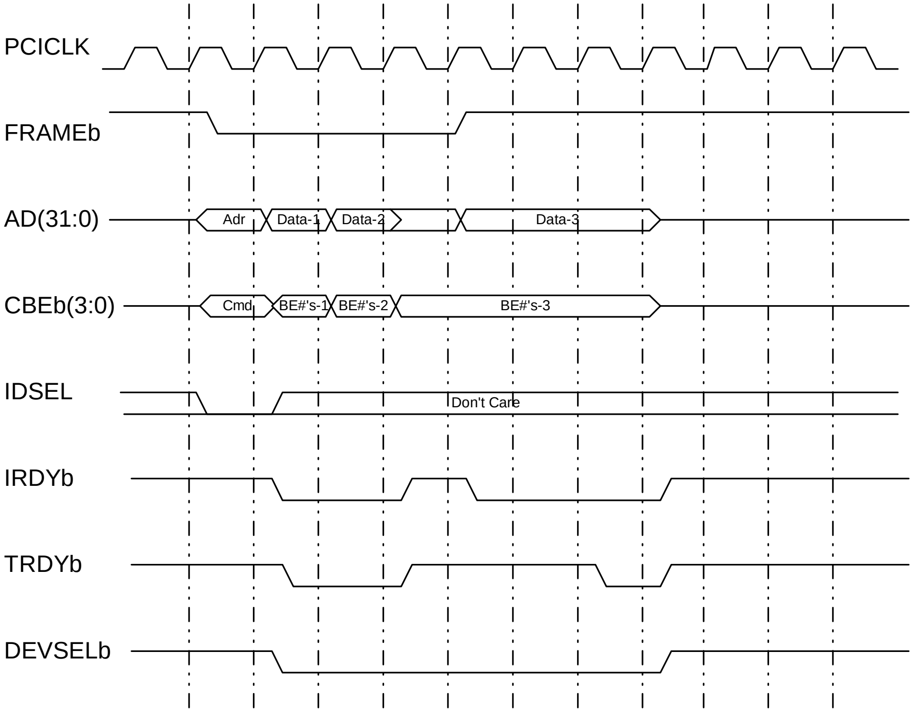


Figure 11-4. PCI Bus Burst Write Cycle Time

11.1.5 PCI Bus Master Operation

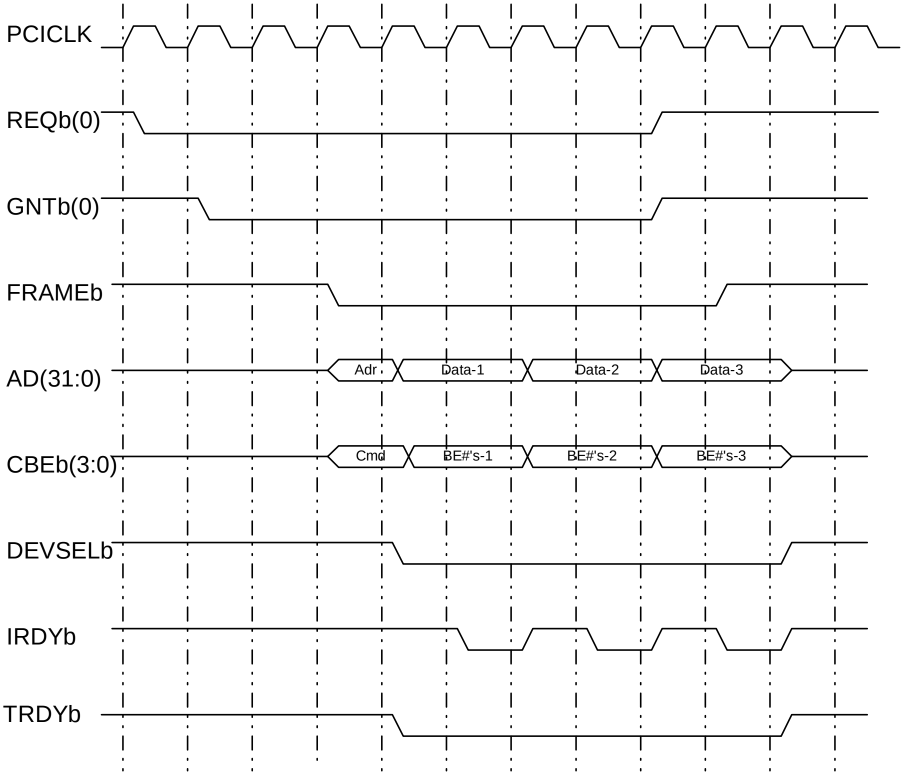


Figure 11-5. PCI Bus Master Operation

11.1.6 Target Abort Termination

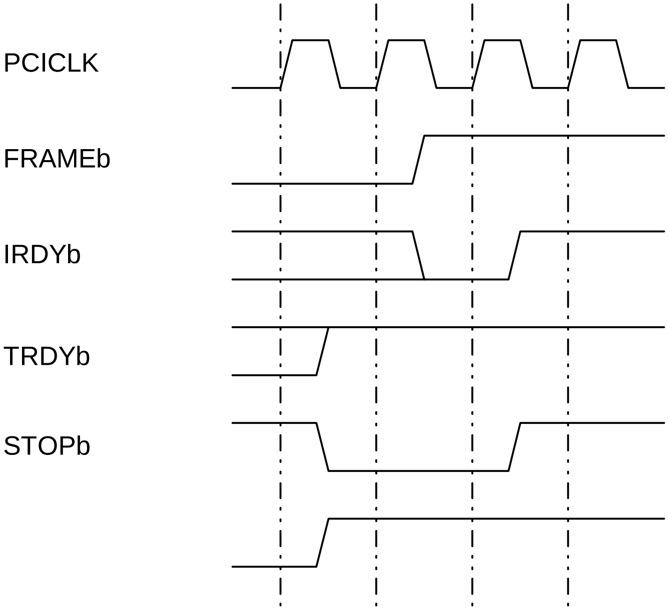


Figure 11-6. Target Abort Termination

11.1.7 Target Disconnect with or without Data

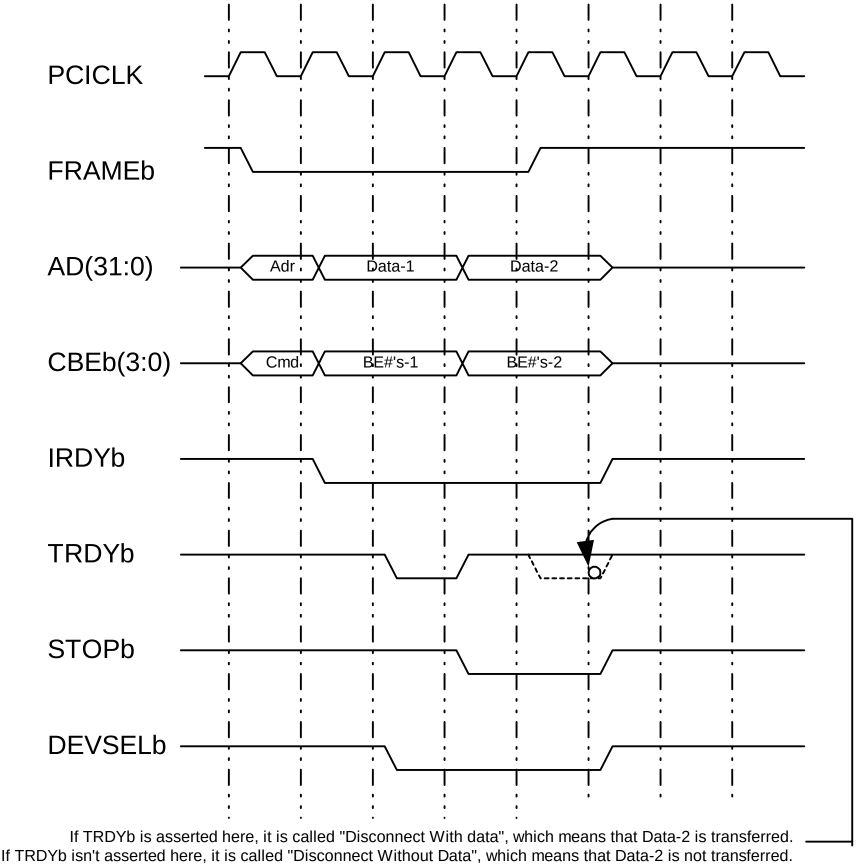

PCICLK
FRAMEb
AD(31:0)
Adr
Data-1
Data-2
CBEb(3:0)
Cmd
BE#'s-1
BE#'s-2
IRDYb
TRDYb
STOPb
DEVSELb
If TRDYb is asserted here, it is called "Disconnect With data", which means that Data-2 is transferred.
If TRDYb isn't asserted here, it is called "Disconnect Without Data", which means that Data-2 is not transferred.

Figure 11-7. Target Disconnect with or without Data

11.1.8 Master Abort Termination

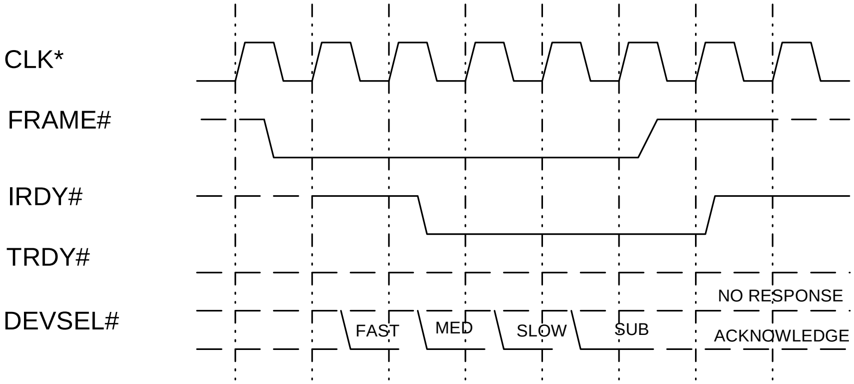


Figure 11-8. Master Abort Termination

11.1.9 Target Abort Termination

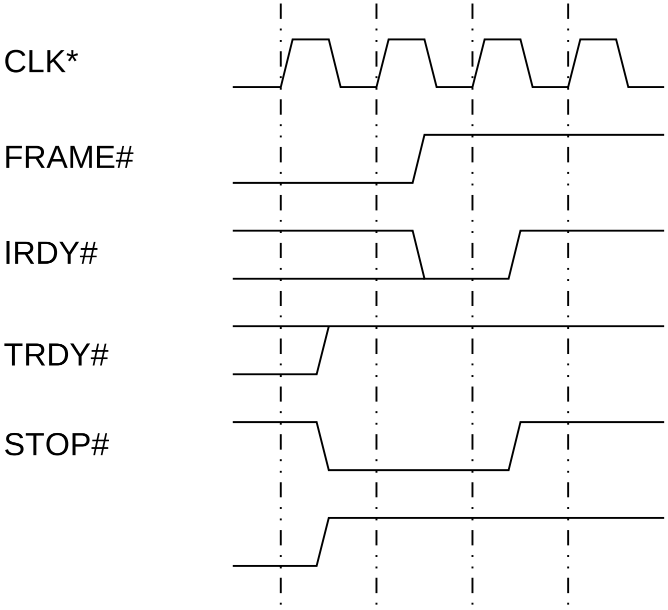


Figure 11-9. Target Abort Termination

11.1.10 SERRb Timing

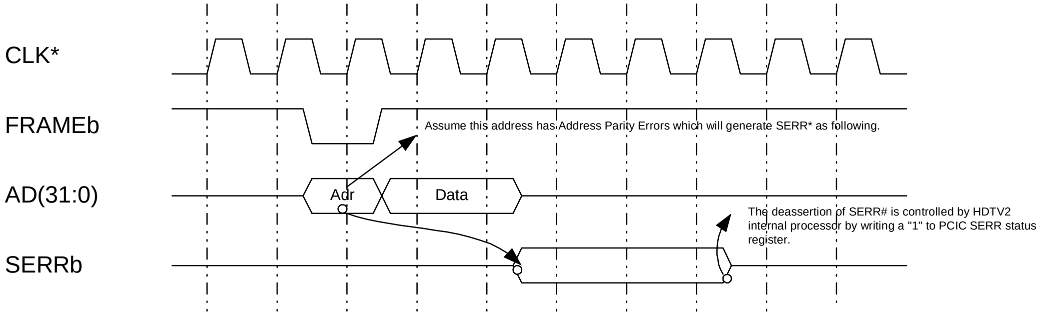

CLK*
FRAMEb
AD(31:0)
SERRb
Adr
Data
Assume this address has Address Parity Errors which will generate SERR* as following.
The deassertion of SERR# is controlled by HDTV2 internal processor by writing a "1" to PCIC SERR status register.

Figure 11-10. SERRb Timing

11.1.11 PERRb Timing

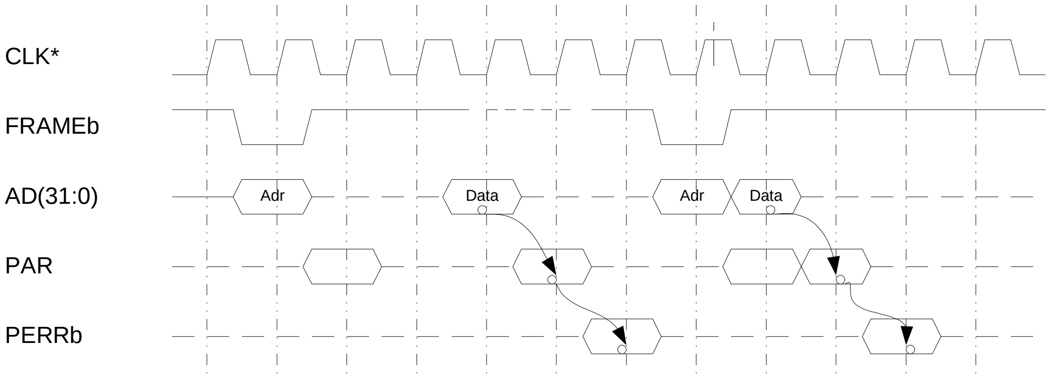


Figure 11-11. PERRb Timing

***Note** (applies to all of Master Abort Termination, Target Abort Termination, SERR and PERR Timing):

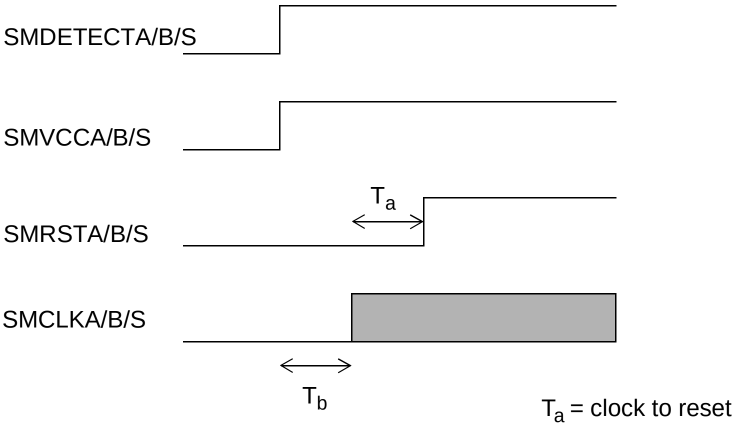
a Signal CLK in these timing diagrams can be both PCICLK (or PCICLKOUT(3:0)). Input clock PCICLK is used for all Xilleon 220 internal block's PCI clock. In Peer modes, PCICLK is driven by external bus. In Solo mode, PCICLK is the same as Xilleon 220 output PCICLKOUT(3:0), which is generated from Xilleon 220 internal clkblk block. PCICLK(3:0) are used for external devices when Xilleon 220 is operating at Solo mode.

b RESETb is an Xilleon 220 input pin. It is used for Xilleon 220 asynchronous global chip reset. RESETOUTb is an Xilleon 220 output pin. In Solo modes, it drives the PCI slot resets, FlexBus device resets, and resets for external I^2C devices. In Peer modes, it drives the FlexBus device resets, and resets for external I^2C devices. RESETOUTb is generated from input RESETb and programmable register bit called "SOFTRESET_PCI" with trailing edge delayed.

c The timing for REQb(0) and REQb(2:1) (or GNTb(0) and GNTb(2:1)) are the same. In Peer modes; external devices generate REQb(2:1) to Xilleon 220. Xilleon 220's internal PCI Sub-Arbiter will generate a global request REQb(0) to external Host Bridge, which is an Xilleon 220 output pin. GNTb(0) is an input grant from external Host Bridge. In Solo modes, external devices generate REQb(2:0) to Xilleon 220. Xilleon 220's internal central resource controller will generate output GNTb(3:0) to external devices.

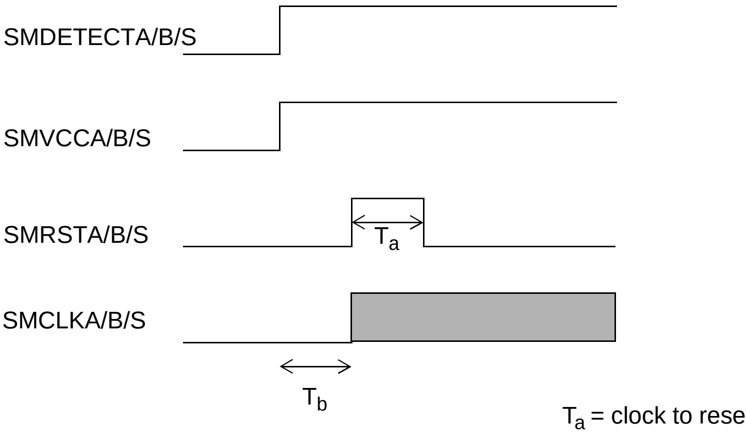
11.2 Smart Card Timing

The following timing diagrams apply to all of Smart Cards A, B and Shopping.



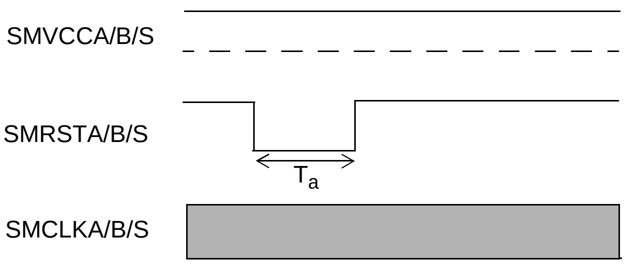
T_a = clock to reset
T_b = Vcc to clock

Figure 11-12 Smart Card Cold Reset (Mode 1)



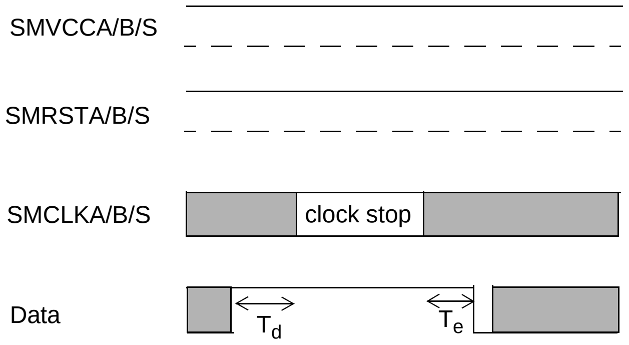
T_a = clock to reset
T_b = Vcc to clock

Figure 11-13 Smart Card Cold Reset (Mode 2)



T_a = clock to reset

Figure 11-14 Smart Card Warm Reset



T_d >=1860/f
T_e >= 700/f

f = clock frequency

Figure 11-15 Smart Card Clock Stop

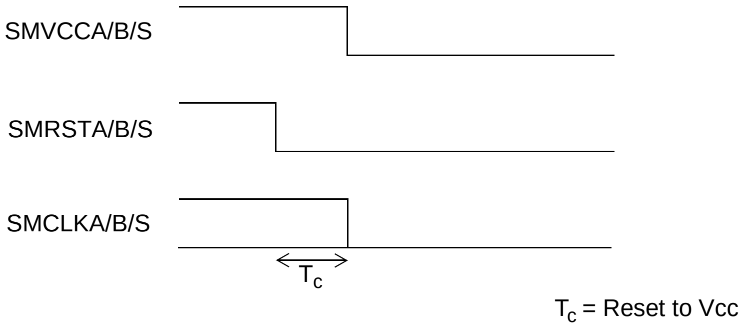


Figure 11-16 Smart Card Deactivation

Table 11-2 Smart Card Interface Timing Parameters

Parameter	Description	Min (ns)	Max (ns)
Ta	Clock to Reset		
Tb	Vcc to Clock		
Tc	Reset to Vcc		
Td		1860/f	-
Te		700/f	-
f	Clock Frequency		

Note: Ta, Tb, and Tc are programmable

11.3 Memory Timing

11.3.1 Single Data Rate SDRAM/SGRAM

Table 11-3 AC Single Data Rate SDRAM/SGRAM Cycle Timing Values

Parameter	Description	Min (ns)	Max (ns)
tC	Clock period provided	6.0	15.0
tCH	Clock high time provided	2.70	
tCL	Clock low time provided	2.70	
tCMS	Command setup time provided (RAS, CAS, WE, CS, CKE)	2.00	
tCMH	Command hold time provided (RAS, CAS, WE,CS, CKE)	1.00	
tAS	Address setup time provided	2.00	
tAH	Address hold time provided	1.00	
tDQMS	DQM setup time provided	2.00	
tDQMH	DQM hold time provided	1.00	
tWDS	Write data setup time provided	2.00	
tWDH	Write data hold time provided	1.00	
tRDS	Read data setup time required [1]	0.50	
tRDH	Read data hold time required [1]	0.50	

Note 1: Read Data Setup and Hold times relative to Clock at the QS0 & QS4 input pins

11.3.2 Double Data Rate SDRAM/SGRAM

Table 11-4 AC Double Data Rate Cycle Timing Values w/o SDRAM DLL

Parameter	Description	Min	Max
tC	Clock period provided	6.0ns	15.0 ns
tCH	Clock high time provided	2.70ns	
tCL	Clock low time provided	2.70ns	
tCMS	Command setup time provided (RAS, CAS, WE, CS, CKE)	2.00ns	
tCMH	Command hold time provided (RAS, CAS, WE,CS, CKE)	1.00ns	
tAS	Address setup time provided	2.00ns	
tAH	Address hold time provided	1.00ns	
tWQS	Clock to valid write strobe provided	0.25*tC	tC
tWPRE	Write strobe preamble provided	0.25*tC	
tWPST	Write strobe postamble provided	0.25*tC	
tDQMS	DM setup time provided w.r.t. QS	0.12*tC	

Table 11-4 AC Double Data Rate Cycle Timing Values w/o SDRAM DLL (Continued)

Parameter	Description	Min	Max
tDQMH	DM hold time provided w.r.t. QS	0.12*tC	
tWDS	Write data setup time provided w.r.t. QS	0.12*tC	
tWDH	Write data hold time provided w.r.t. QS	0.12*tC	
tRQS	Clock to read data strobe required	2.00	0.9*tC
tRPRE	Read strobe preamble required	tC	
tRPST	Read strobe postamble required	0.5*tC	
tRDQS	Read data to QS variance required		0.50

Table 11-5 AC Double Data Rate Cycle Timing Values w/ SDRAM DLL

Parameter	Description	Min	Max
tC	Clock period provided	6.0ns	15.0ns
tCH	Clock high time provided	2.70ns	
tCL	Clock low time provided	2.70ns	
tCMS	Command setup time provided (RAS, CAS, WE, CS, CKE)	2.00ns	
tCMH	Command hold time provided (RAS, CAS, WE,CS, CKE)	1.00ns	
tAS	Address setup time provided	2.00ns	
tAH	Address hold time provided	1.00ns	
tWQS	Clock to valid write strobe provided	0.75*tC	1.25*tC
tWPRE	Write strobe preamble provided	0.75*tC	
tWPST	Write strobe postamble provided	tC	
tDQMS	DM setup time provided w.r.t. QS	0.12*tC	
tDQMH	DM hold time provided w.r.t. QS	0.12*tC	
tWDS	Write data setup time provided w.r.t. QS	0.12*tC	
tWDH	Write data hold time provided w.r.t. QS	0.12*tC	
tRQS	Clock to read data strobe required	2.00ns	0.9*tC
tRPRE	Read strobe preamble required	tC	
tRPST	Read strobe postamble required	0.5*tC	
tRDQS	Read data to QS variance required		0.50ns

11.3.3 Programming Timing Values

The table below shows the memory timing parameters that may be programmed through the registers EXT_MEM_CNTL(=0x00000844), MEM_ADDR_CONFIG(=0x00000848) and MEM_SDRAM_MODE_REG(=0x00000858). They apply to both Single Data Rate SDRAM/SGRAM and Double Data Rate SDRAM/SGRAM.

Table 11-6 Programming of Timing Values

Description	Symbol	Min (clocks)	Max (clocks)	Register field in EXT_MEM_CNTL
Row cycle time	tRC	5	15	MEM_TRP[1:0] + MEM_TRAS[6:4]
PRE to ACTV minimum delay	tRP	1	4	MEM_TRP[1:0]
ACTV to PRE minimum delay	tRAS	4	11	MEM_TRAS[6:4]
Extra Row Cycle time for Refresh	tRFC	0	3	MEM_TRFC[20:19]
ACTV to Rd CMD minimum delay	tRCD	1	4	MEM_TRCD[3:2]
ACTV to Wr CMD minimum delay	tRCDW	1	4	MEM_TRCDW[22:21]
ACTV to ACTV minimum delay	tRRD	1	4	MEM_TRRD[9:8]
Write recovery time	tWR	0	3	MEM_TWR[13:12]
Different group/rank read to read data minimum turnaround cycles	tR2R	0	3	MEM_TR2R[15:14]
Read to write data minimum turnaround cycles	tR2W	1	4	MEM_TR2W[11:10]
Write to read minimum QS turnaround cycles (DDR only)	tW2R	0	3	MEM_TW2R[18:16]
				Register field in MEM_ADDR_CONFIG
Different group/rank write-to-write minimum QS turnaround cycles (DDR only)	tW2W	0	3	MEM_TW2W[31:30]
				Register field in MEM_SDRAM_MODE_REG
Read Data CAS Latency	CL	2	4	MEM_CAS_LATENCY[22:20][

The tables below show the **recommended** values for the above parameters

Table 11-7 Recommended Timing Values - SDR

Parameter	Description	Register field in EXT_MEM_CNTL
tRC	Row cycle time	9 clocks
tRP	PRE to ACTV minimum delay	MEM_TRP[1:0] = 2 (3 clocks)
tRAS	ACTV to PRE minimum delay	MEM_TRAS[6:4] = 5 (6 clocks)
tRFC	Extra Row Cycle time for Refresh	MEM_TRFC[20:19] = 0 (0 clocks)
tRCD	ACTV to Rd CMD minimum delay	MEM_TRCD[3:2] = 2 (3 clocks)

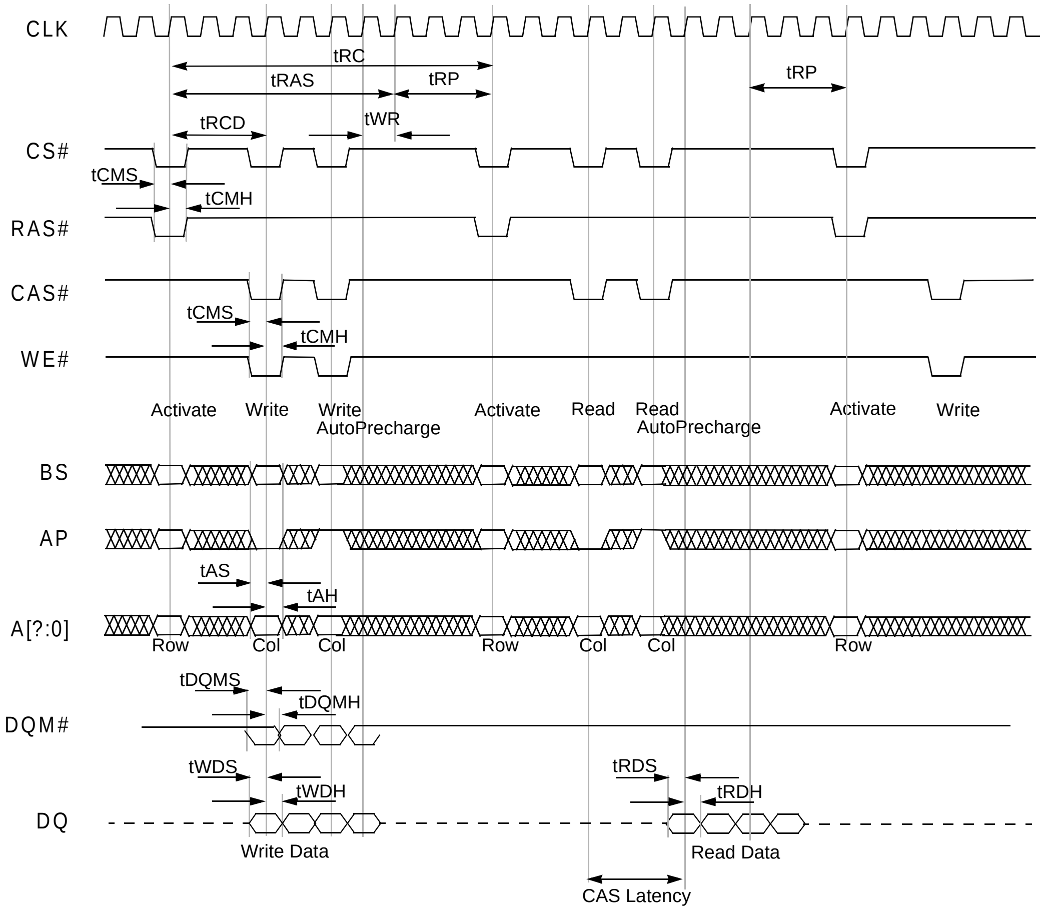
Table 11-7 Recommended Timing Values - SDR (Continued)

Parameter	Description	Register field in EXT_MEM_CNTL
tRCDW	ACTV to Wr CMD minimum delay	MEM_TRCDW[3:2] = 2 (3 clocks)
tRRD	ACTV to ACTV minimum delay	MEM_TRRD[9:8] = 1 (2 clocks)
tWR	Write recovery time	MEM_TWR[13:12] = 1 (2 clocks)
tR2R	Different group/rank read to read data minimum turnaround cycles	MEM_TR2R[17:16] = 1 (1 clocks)
tR2W	Read to write data minimum turnaround cycles	MEM_TR2W[11:10] = 0 (1 clock)
tW2R	Write to read command delay	MEM_TW2R[18:16] = 0 (0 clocks)
		Register field in MEM_ADDR_CONFIG
tW2W	Different group/rank write-to-write minimum QS turnaround cycles	MEM_TW2W[31:30] = 0 (0 clocks)
		Register field in MEM_SDRAM_MODE_REG
CL	Read Data CAS Latency	MEM_CAS_LATENCY[22:20] = 3 (3 clocks)

Table 11-8 Recommended Timing Values - DDR

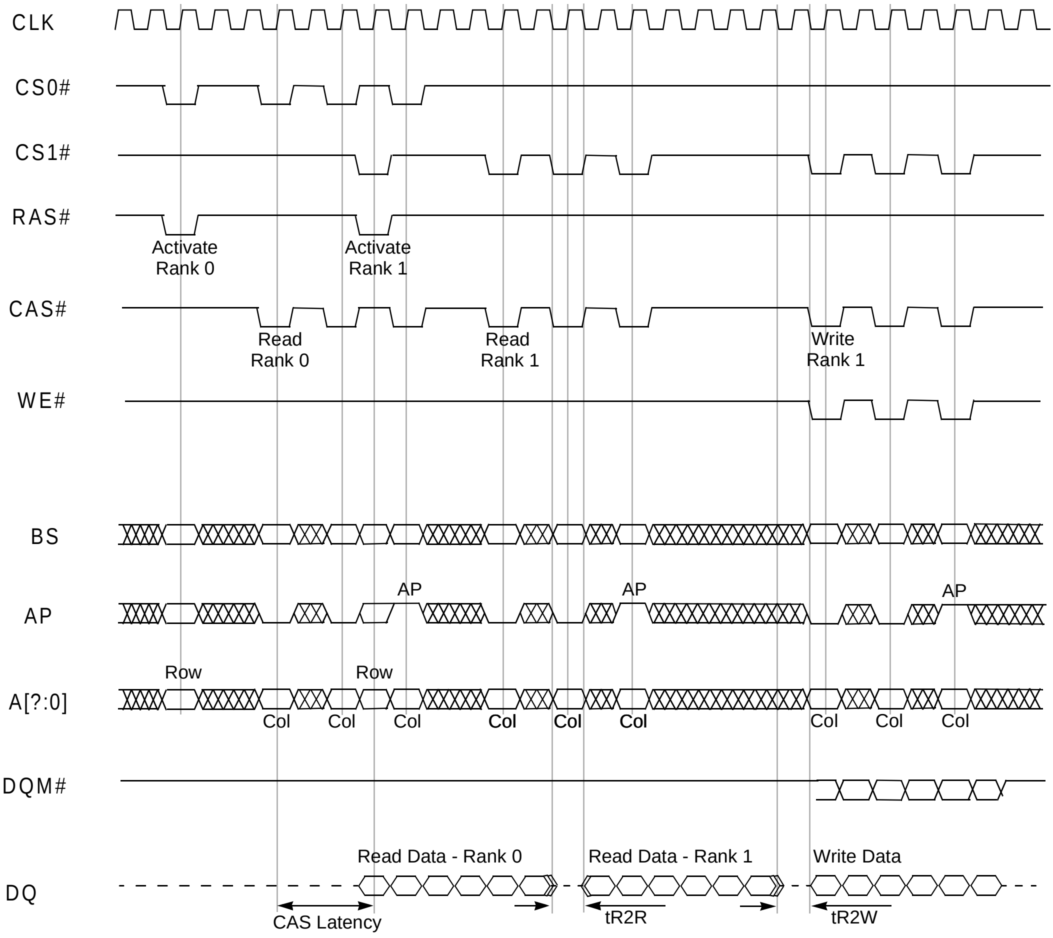
Parameter	Description	Register field in EXT_MEM_CNTL
tRC	Row cycle time	9 clocks
tRP	PRE to ACTV minimum delay	MEM_TRP[1:0] = 2 (3 clocks)
tRAS	ACTV to PRE minimum delay	MEM_TRAS[6:4] = 5 (6 clocks)
tRFC	Extra Row Cycle time for Refresh	MEM_TRFC[20:19] = 2 (2 clocks)
tRCD	ACTV to Rd CMD minimum delay	MEM_TRCD[3:2] = 2 (3 clocks)
tRCDW	ACTV to Wr CMD minimum delay	MEM_TRCDW[3:2] = 2 (3 clocks)
tRRD	ACTV to ACTV minimum delay	MEM_TRRD[9:8] = 1 (2 clocks)
tWR	Write recovery time	MEM_TWR[13:12] = 3 (4 clocks)
tR2R	Different group/rank read to read data minimum turnaround cycles	MEM_TR2R[17:16] = 2 (2 clocks)
tR2W	Read to write data minimum turnaround cycles	MEM_TR2W[11:10] = 0 (1 clock)
tW2R	Write to read command delay	MEM_TW2R[18:16] = 2 (2 clocks)
		Register field in MEM_ADDR_CONFIG
tW2W	Different group/rank write-to-write minimum QS turnaround cycles	MEM_TW2W[31:30] = 1 (1 clock)
		Register field in MEM_SDRAM_MODE_REG
CL	Read Data CAS Latency	MEM_CAS_LATENCY[22:20] = 3 (3 clocks)

11.3.4 Single Data Rate (SDR) SGRAM/SDRAM Timing Diagrams



Basic Read and Write Cycles:
Basic Read and Write Cycles with Auto Precharge:
Burst Length = 2, CAS Latency = 3

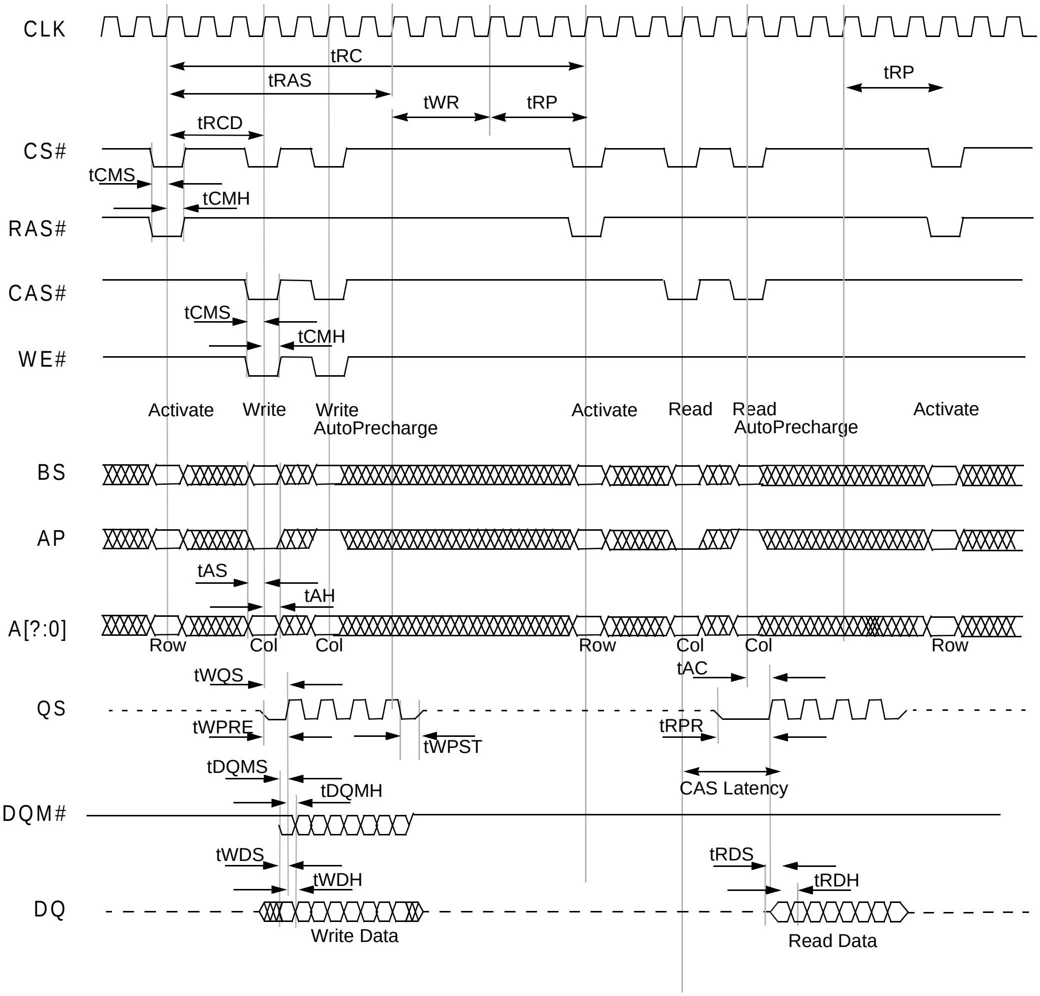
Figure 11-17. SDR SGRAM/SDRAM Basic Read/Write Cycle Timing



Read Turnaround with tR2R =1
Read-to-Write Turnaround with tR2W = 1
Burst Length = 2, CAS Latency = 3

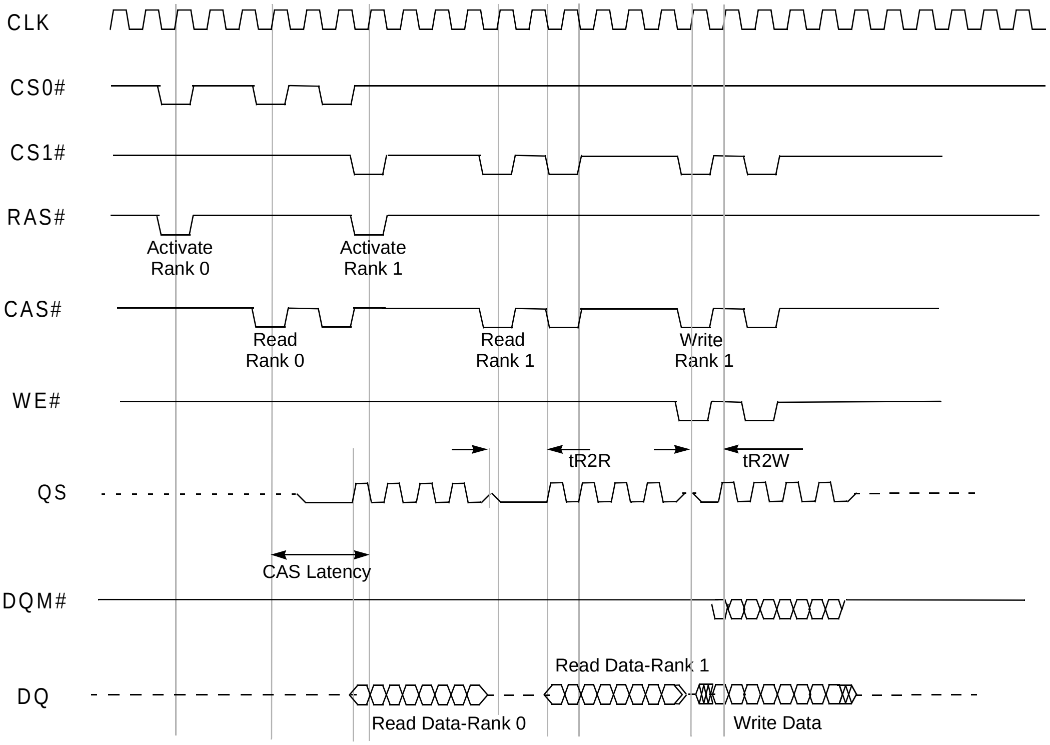
Figure 11-18. SDR Read/Write Data Turnaround Cycles

11.3.5 Double Data Rate (DDR) SGRAM Timing Diagrams



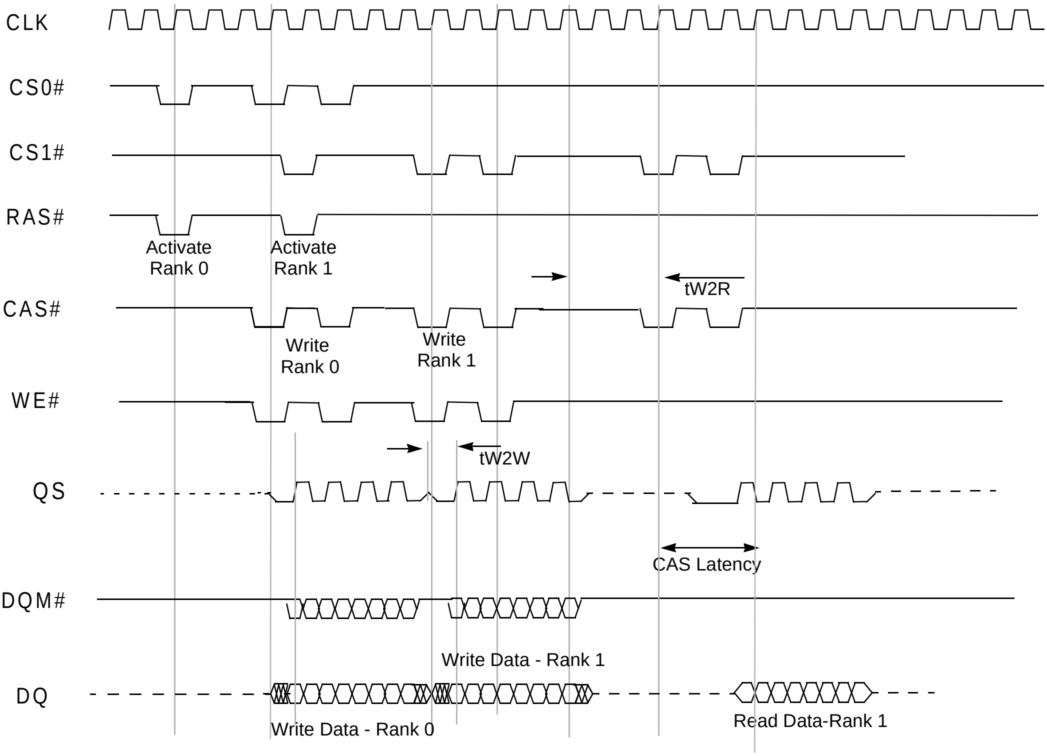
Basic Read and Write Cycles with Auto Precharge:
Burst Length = 4, CAS Latency = 3
SDRAM DLL Disabled

Figure 11-19. DDR SGRAM/SDRAM Basic Read/Write Cycle Timing



Read Turnaround with tR2R =2
Read-to-Write Turnaround with tR2W = 1
Burst Length = 4, CAS Latency = 3
SDRAM DLL Disabled

Figure 11-20. DDR Read/Read & Read/Write Data Turnaround Cycles



Read Turnaround with tW2R =2
Read-to-Write Turnaround with tW2W = 1
Burst Length = 4, CAS Latency = 3
SDRAM DLL Disabled

Figure 11-21. DDR Write/Write & Write/Read Data Turnaround Cycles

11.3.6 Input/Output Timing

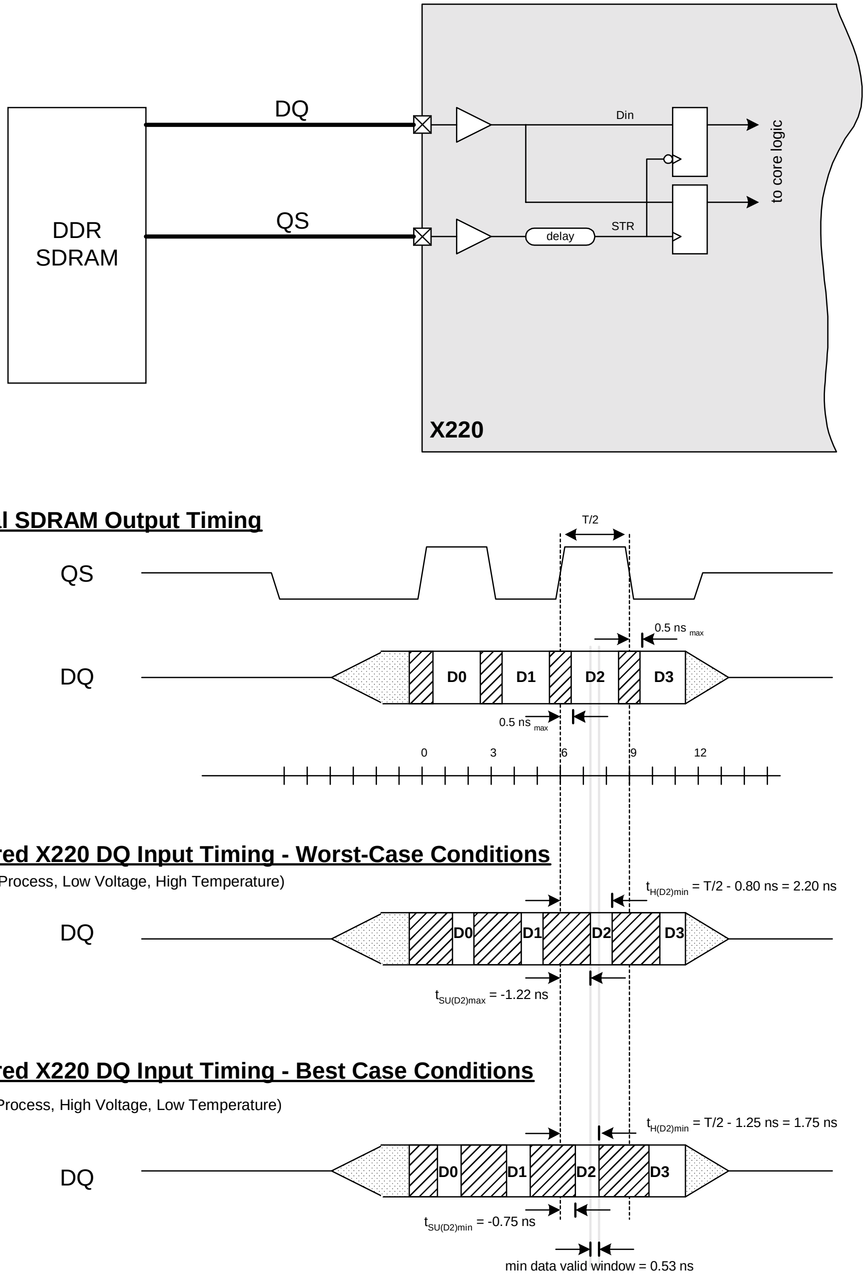

DQ
QS
DDR
SDRAM
Din
STR
delay
to core logic
X220
Typical SDRAM Output Timing
T/2
QS
DQ
0.5 ns max
D0
D1
D2
D3
0.5 ns max
0
3
6
9
12
Required X220 DQ Input Timing - Worst-Case Conditions
(Slow Process, Low Voltage, High Temperature)
tH(D2)min = T/2 - 0.80 ns = 2.20 ns
DQ
D0
D1
D2
D3
tSU(D2)max = -1.22 ns
Required X220 DQ Input Timing - Best Case Conditions
(Fast Process, High Voltage, Low Temperature)
tH(D2)min = T/2 - 1.25 ns = 1.75 ns
DQ
D0
D1
D2
D3
tSU(D2)min = -0.75 ns
min data valid window = 0.53 ns

Figure 11-22. Read Data Input Timing Requirements for the Xilleon 220 Memory Data for DDR SDRAMs

Figure 11-22. above illustrates the read data input timing requirements for the Xilleon 220 Memory Data for Double-Data Rate SDRAMs.

First, the interface is shown schematically as a DQ and QS input from SDRAM to X220. The DQ represents any of the bi-directional Data bus signals. The QS represents any of the bi-directional Data Strobe signals.

The SDRAM transmits the DQ and QS in an "edge-aligned" relationship; that is, the DQ data value changes at nominally the same time as the QS edges. For a typical high-performance DDR SDRAM, the alignment is better than 0.5 ns. A 166 MHz clock rate timing is shown.

Once the DQ and QS are received, the X220 delays the QS signal into a new signal, STR, so that the edges are nominally centered on the data-valid window to achieve reliable data capture. The X220 implementation has a programmable delay element that can be used to optimize this timing, primarily at lower clock rates. This analysis is based on the shortest and simplest internal delay possibility.

Since the STR is delayed inside the X220, the specification of setup and hold timing at the X220 external interface is unusual. The setup time shows the latest that the data may arrive; it is negative, meaning that the data edge may change after the capturing edge of QS. The hold-time shows the earliest time when the transition to the next data value may start; it is unusually long because of the negative setup time.

Two cases are shown in *Figure 11-22.* above: worst case and best case. The worst case represents slow process, low voltage and high temperature, while the best case represents fast process, high voltage, and low temperature. These two corners represent the most pessimistic variation in timing.

As shown in *Figure 11-22.* above, the typical SDRAM output timing provides a valid DQ window that is well within the required capture window of the X220 and has margin for PC Board tolerances.

Table 11-9 Minimum Setup /Hold time for Different Programmable Delay Settings

STR DELAY PATH[1]	DLL DELAY TAP[2,3]	DLL RANGE[4,5]	MIN SETUP TIME REQUIRED[6] (ns)	MIN HOLD TIME REQUIRED (ns)
Pad	n/a	n/a	-1.1	1.8
DLL	Bypass	n/a	-0.43	1.9
	2T/N	0	-0.43 - 0.07T	1.9 + 0.07T
		1	-0.43 - 0.08T	1.9 + 0.08T
		2	-0.43 - 0.11T	1.9 + 0.11T
		3	-0.43 - 0.14T	1.9 + 0.14T
	4T/N	0	-0.43 - 0.14T	1.9 + 0.14T
		1	-0.43 - 0.17T	1.9 + 0.17T
		2	-0.43 - 0.22T	1.9 + 0.22T
		3	-0.43 - 0.29T	1.9 + 0.29T

Notes:

1 Controlled by register fields:
MC.MEM_DLL_CNTL_RDCLK*.PAD_STRSEL*.
Cannot use PAD delay for x32-bit SDRAMs (with only 1 QS)

2 Controlled by register fields:
MC.MEM_DLL_CNTL_RDCLK*.RDCLK*_SINSEL
MC.MEM_DLL_CNTL_RDCLK*.RDCLK*_SOUTSEL

3 When using the frequency-dependent tapped delay line for STR delay; T is the CLK period.

4 Controlled by register fields:
CG.XDLL_RDCLK*.XDLL_RDCLK*_RANGE

5 Use range 0,1 for 60 MHz and up; Range 2,3 for 100 MHz and up

6 Negative value for setup time means that data is not required to be stable until | t_{SU} | after the QS edge that captures it.

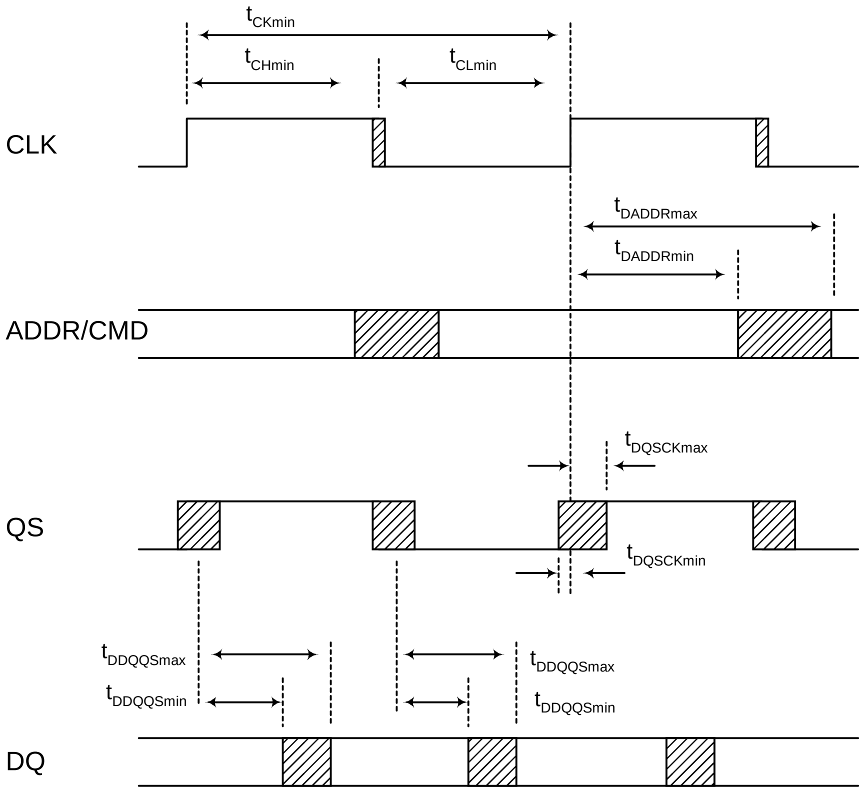


Figure 11-23. X220 DDR Output Timing

Table 11-10 X220 DDR Output Timing Parameters

Parameter	Value (ns)
tCKmin	6
tCHmin	2.4
tCLmin	2.4
tDADDRmin	2.8
tDADDRmax	4.2
tDQSCKmin	-0.1
tDQSCKmax	0.7
tDDQQSmin	0.7
tDDQQSmax	2.4

Note: The values above are for the worst case and include the effects of ground bounce. All outputs are terminated with 10pF.

11.4 Transport Stream Interface Timing

The transport demultiplexer input interface consists of 5 pins in serial mode or 12 pins in parallel mode: TCLK, TDAT, TSTRT, TVLD, TER. This interface can operate from a clock qualified data source (TCLK and TDAT are required) where clock runs only when data is available. It can also take a stream from a data validated source (TCLK, TDAT & TVLD are required) where clock can be synchronous or occasional (runs with data) and each data bit, bytes, group of bytes of a full transport packet is valid when TVLD signal is asserted. Signal TSTRT is optional, and if it exists, it should be byte aligned to first byte of the transport packet (with a bit or byte time duration). Finally, when TER is asserted, no data is being clocked to the input framing logic.

11.4.1 Input Stream (A and B) and Out of Band

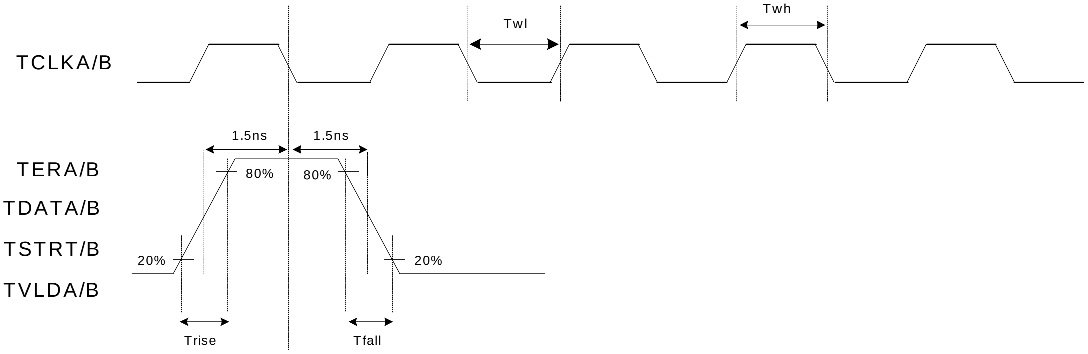


Figure 11-24. Transport Stream Input Timing

Table 11-11 Transport Stream (A and B) and OOB Input Timing Parameters

Parameter	Min (ns)	Max (ns)
TCLKA/B, OOBCLK		10
Twh		5
Twl		5
Trise		1.4
Tfall		1.4
TDATA/B, OOBDAT setup	1.5	
TDATA/B, OOBDAT hold	0	
TSTRTA/B, OOBSTRT setup	1.5	
TSTRTA/B, OOBSTRT hold	0	
TVLDA/B, OOBVLD setup	1.5	
TVLDA/B, OOBVLD hold	0	
TERA/B, OOBER setup	1.5	

Table 11-11 Transport Stream (A and B) and OOB Input Timing Parameters

Parameter	Min (ns)	Max (ns)
TERA/B, OOBER hold	0	

11.4.2 Output Stream C (Normal Mode)

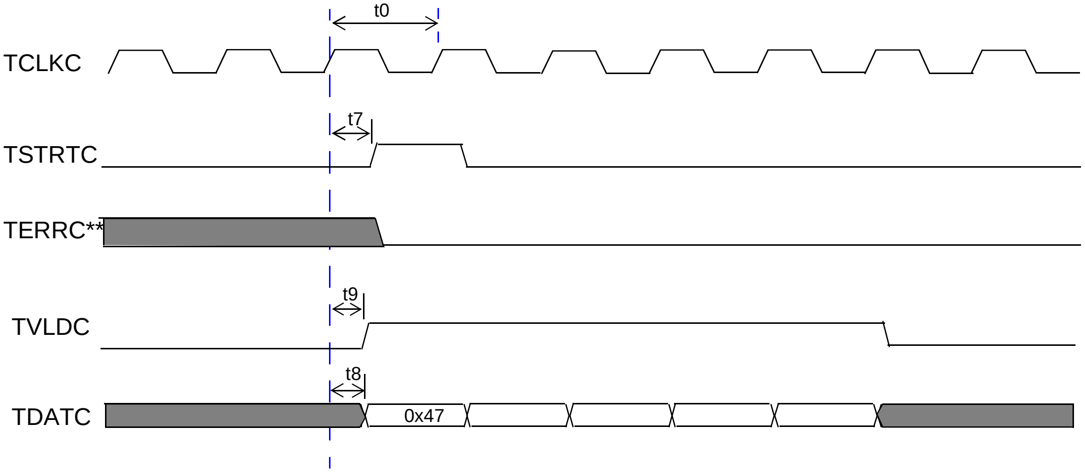


Figure 11-25. Transport Stream C Output Timing

Table 11-12 Transport Stream C Output Timing Parameters (Preliminary)

Parameter	Description	Min (ns)	Max (ns)
t0	TCLKC period (minimal XCLK*2)	12.0	0.0
t7	TCLKC to TSTRTC valid time		8.0
t8	TCLKC to TDATC valid time		8.0
t9	TCLKC to TVLDC valid time		8.0
t10	TCLKC to TERRC valid time		n/a

Notes:

- In bypass mode, timing on Transport Stream C will be the same as on input Transport Stream A or B.

- TERRC signal available only on bypass mode.

11.4.3 NRSS Timing

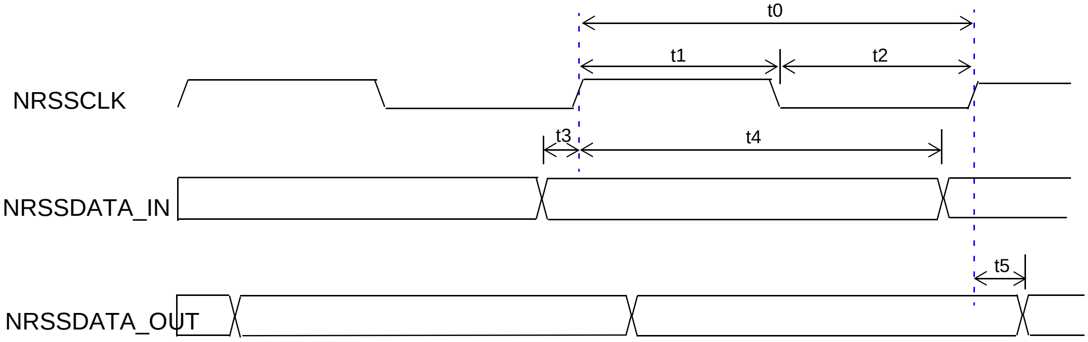


Figure 11-26. NRSS Timing

Table 11-13 NRSS Timing Parameters (Preliminary)

Parameter	Description	Min (ns)	Max (ns)
t0	NRSSCLK period	10	
t1	NRSSCLK high	5	
t2	NRSSCLK low	5	
t3	NRSSDATA_IN to NRSSCLK setup time	2	
t4	NRSSDATA_IN to NRSSCLK hold time	0	
t5	NRSSDATA_OUT valid time		2

11.5 DVI Out Timing

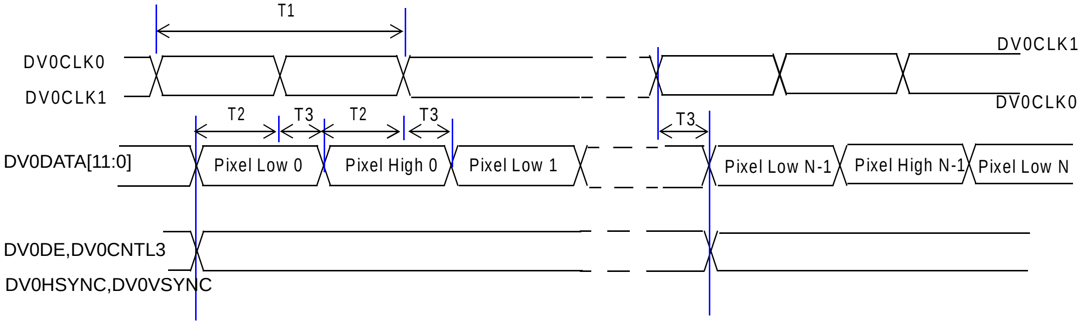


Figure 11-27 DVI Out Timing

Table 11-14 DVI Out Timing Parameters

Parameter	Description	Min (ns)	Max (ns)
T1	clock period	6.73	-
T2	Data, DE, VSYNC, HYSNC, CNTL3 output setup time to DV0CLK0/1 rising/falling edge	1.0	-
T3	Data, DE, VSYNC, HYSNC, CNTL3 output hold time to DV0CLK0/1 rising/falling edge	1.0	-

11.6 ITU-656 Timing

11.6.1 ITU-656 In

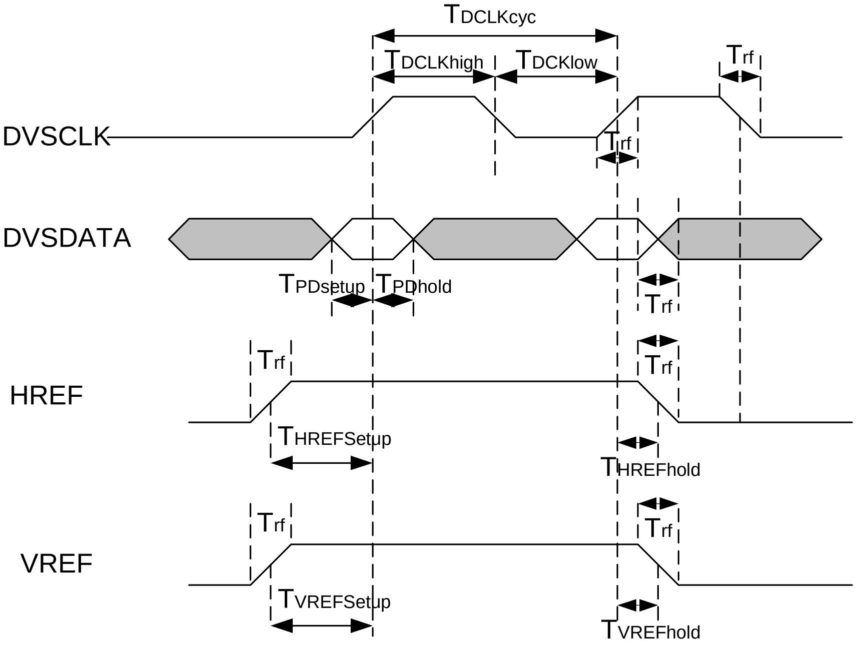


Figure 11-28. ITU-656 In Timing

Table 11-15 ITU-656 InTiming Parameters

Parameter	Description	Min (ns)	Typ (ns)	Max (ns)
TDCLKcyc	DVSCLK cycle time	13.3	37	
TDCLKhigh	DVSCLK high time	6	18	
TTCKlow	DVSCLK low time	6	18	
TPDsetup	DVSDATA setup time before rising edge of DVSCLK	3		
TPDhold	DVSDATA hold time after rising edge of DVSCLK	3		
THREFsetup	HREF setup timing before the rising edge of DVSCLK	6		
THREFhold	HREF hold timing after the rising edge of DVSCLK	3		
THREFsetup	VREF setup timing before the rising edge of DVSCLK	6		
THREFhold	VREF hold timing after the rising edge of DVSCLK	3		
Trf	DVS signals rise and fall time			1.5

11.6.2 ITU-656 Out

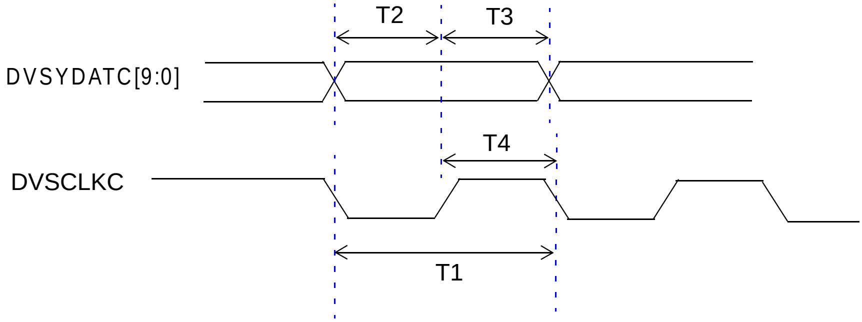


Figure 11-29 ITU-656 Out Timing

Table 11-16 ITU-656 Out Timing Parameters

Parameter	Description	Min.(ns)	Max.(ns)
T1	clock period	-	37
T2	output data setup time to clock[1]	3.37	
T3	output data hold time to clock[1]	3.37	
T4	clock pulse width[1]	3.37	33.7

[1] The clock pulse width and skew to data are adjustable by the clock pattern.

11.7 SPDIF Timing

The diagram below applies to both SPDIFA and SPDIFB.

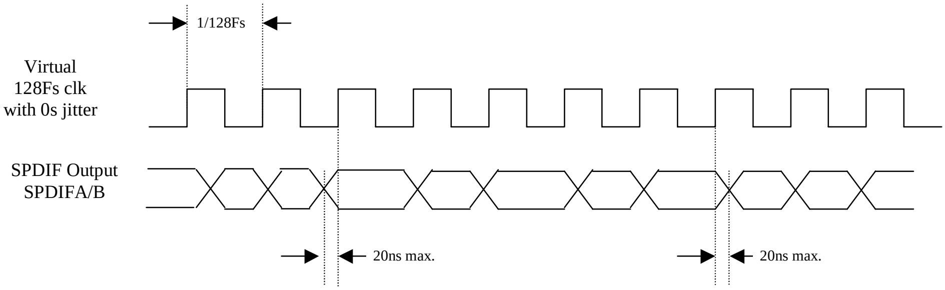


Figure 11-30. SPDIF Timing Diagram

11.8 I2S Bus Timing

11.8.1 I2S Transmitter Timing

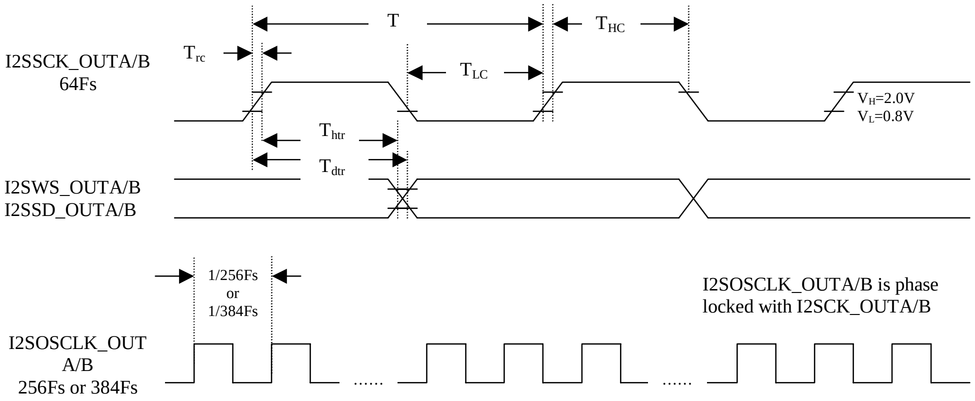


Figure 11-31. I2S Transmitter Interface Timing

Table 11-17 I2S Transmitter Interface Timing Parameters

Parameter	Description	Requirement
T	Typ. Clock period	T = 64Fs, Fs= sampling frequency Min allowed clk period for a transmitter: 0.9T= T_{tr}
T_{HC}	Clock high	Min > 0.35T
T_{LC}	Clock low	Min > 0.35T
T_{dtr}	Delay	Max < 0.8T
T_{htr}	Hold time	Min > 0
T_{rc}	Clock rise time	Max > $0.15T_{tr}$ (only relevant in slave mode)

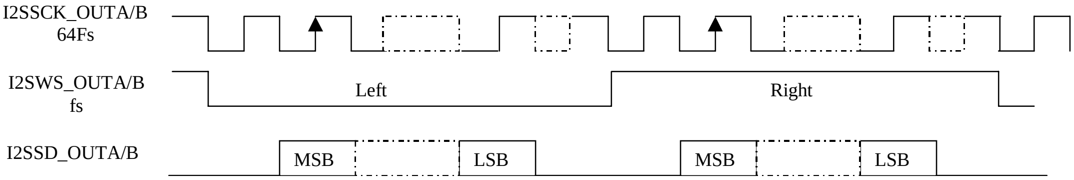


Figure 11-32. I2S Format

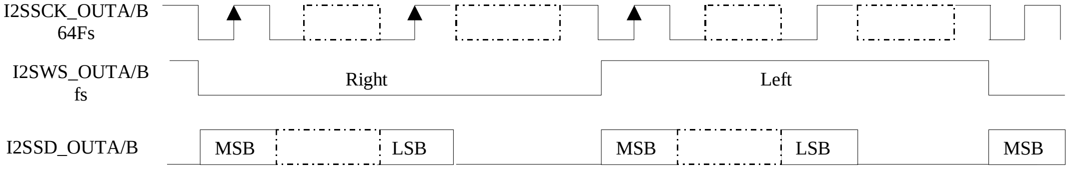


Figure 11-33. Left Justified Format (Rising Edge Valid SCLK)

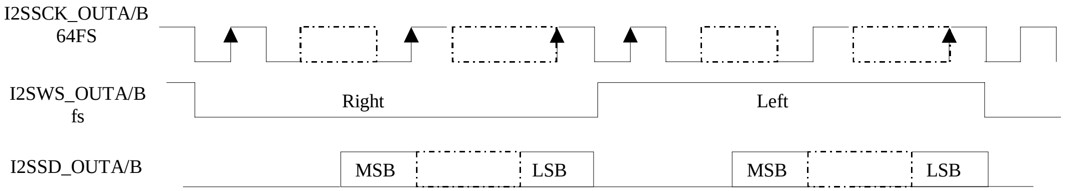


Figure 11-34. Right Justified Format (Rising Edge Valid SCLK)

11.8.2 I2S Receiver Timing

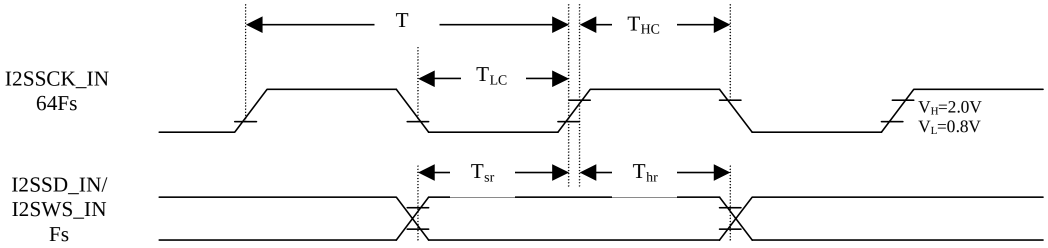


Figure 11-35. I2S Receiver Interface Timing

Table 11-18 I2S Receiver Timing Parameters

Parameter	Description	Requirement
T	Typ. Clock period	T = 1/64Fs, Fs = sampling frequency
T_{HC}	Clock high	Min > 0.35T
T_{LC}	Clock low	Min > 0.35T
T_{sr}	Setup time	Max < 0.2T
T_{htr}	Hold time	Min < 0

11.9 I²C Timing

The diagrams and tables apply to both Channel A and Channel B.

11.9.1 Write Cycle

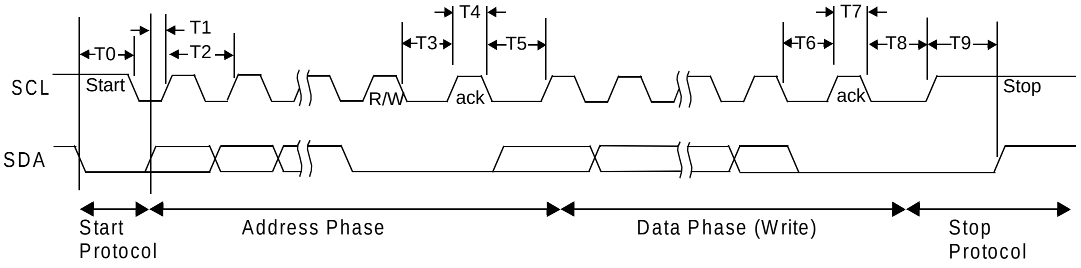


Figure 11-36. I²C Write Cycle

Table 11-19 I²C Write Cycle Timing Parameters

Parameter	Description	Min	Max
T0	Time for the start protocol	$T_{period}/2$	T_{period}
T1	Setup time for outbound address/data	$T_{period}/4$	$T_{period}/4$
T2 (T_{period})	Period of SCL	T_{period}	T_{period}
T3	Time elapse from the R/W bit to ACK	T_{period}	T_{period}
T4	Time for SCL high during ACK	$3T_{period}/4$	$3T_{period}/4$
T5	Time elapse from ACK to the first bit of data	T_{period}	T_{period}
T6	Time elapse from the negative edge of the SCL for the last bit of writing data to the ACK from slave	$7T_{period}/4$	$7T_{period}/4$
T7	Time for SCL high during ACK	$T_{period}/4$	$T_{period}/4$
T8	Time from ACK to STOP protocol	$3T_{period}/4$	$3T_{period}/4$
T9	Setup for stop protocol	$T_{period}/2$	$T_{period}/2$

11.9.2 Read Cycle

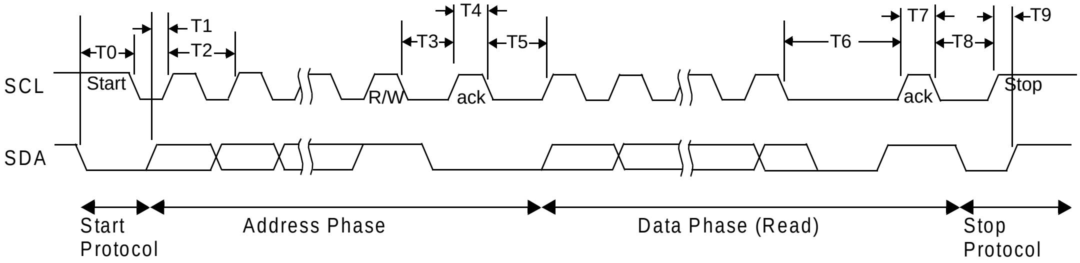


Figure 11-37. I^{2}C Read Cycle

Table 11-20 I^{2}C Read Cycle Timing Parameters

Parameter	Description	Min	Max
T0	Time for the start protocol	$T_{period}/2$	T_{period}
T1	Setup time for outbound address/data	$T_{period}/4$	$T_{period}/4$
T2 (T_{period})	Period of SCL	T_{period}	T_{period}
T3	Time elapse from the R/W bit to ACK	T_{period}	T_{period}
T4	Time for SCL high during ACK	$3T_{period}/4$	$3T_{period}/4$
T5	Time elapse from ACK to the first bit of data	T_{period}	T_{period}
T6	Time elapse from the negative edge of the SCL for the last bit of reading data to ACK from master	$3T_{period}/4$	$3T_{period}/4$
T7	Time for SCL high during ACK	$T_{period}/4$	$T_{period}/4$
T8	Time from ACK to STOP protocol	T_{period}	T_{period}
T9	Setup for stop protocol	$T_{period}/2$	$T_{period}/2$

11.10 AC-Link Timing

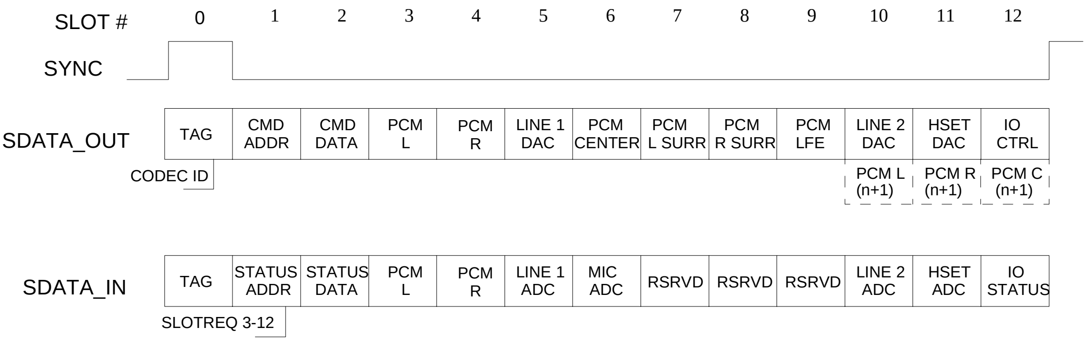


Figure 11-38 AC’97 Standard Bi-directional Audio Frame

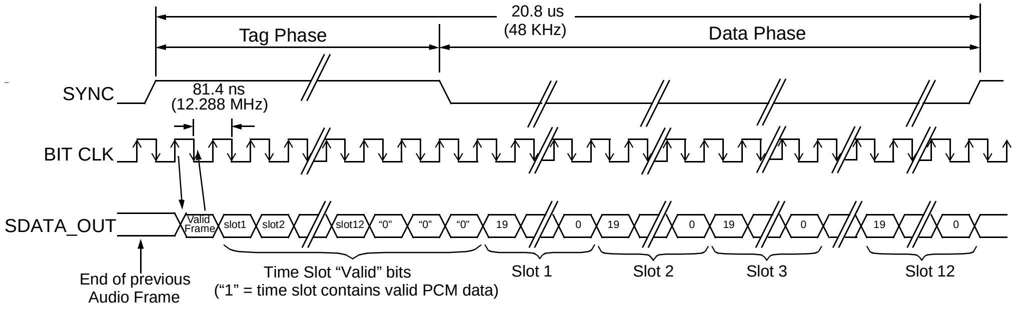


Figure 11-39 AC-Link Audio Output Frame

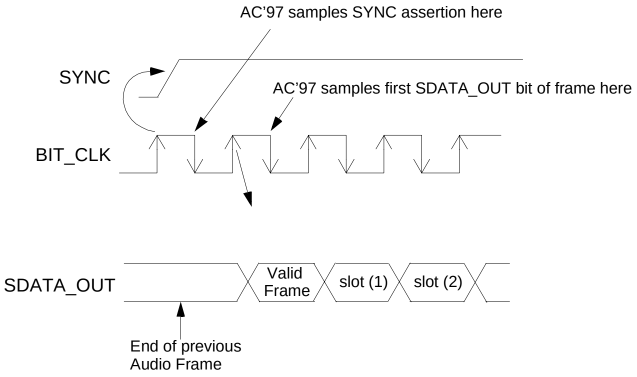

AC'97 samples SYNC assertion here
SYNC
AC'97 samples first SDATA_OUT bit of frame here
BIT_CLK
SDATA_OUT
Valid Frame
slot (1)
slot (2)
End of previous Audio Frame

Figure 11-40 Start of an Audio Output Frame

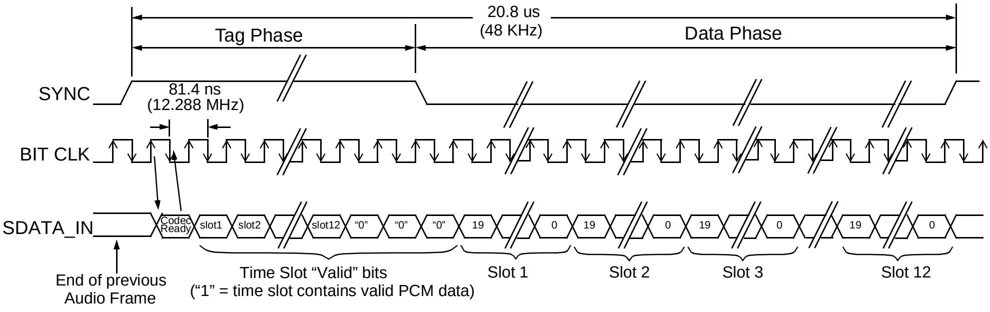


Figure 11-41 AC-Link Audio Input Frame

11.11 Timer Port Timing

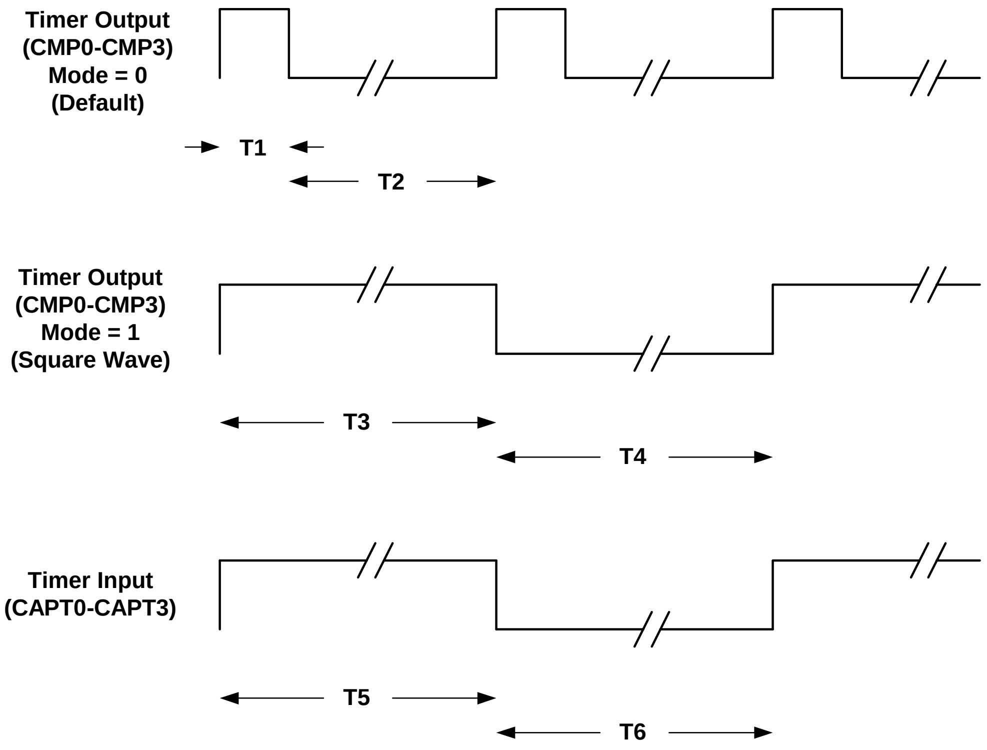


Figure 11-42. Timer Port Timing

Table 11-21 Timer Port Timing Parameters

Parameter	Minimum	Maximum
T1	37 ns	37ns
T2	37 ns	159 seconds
T3	74 ns	159 seconds
T4	74 ns	159 seconds
T5	50 ns	unlimited
T6	50 ns	unlimited

11.12 VIP Host Timing

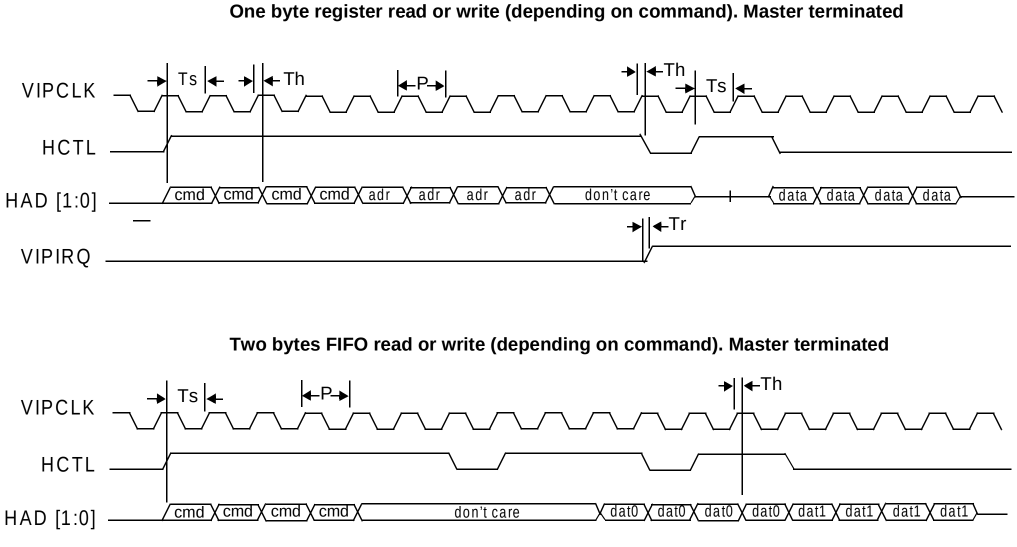

One byte register read or write (depending on command). Master terminated
VIPCLK
HCTL
HAD [1:0]
VIPIRQ
Ts
Th
P
Tr
cmd
adr
don't care
data
Two bytes FIFO read or write (depending on command). Master terminated
dat0
dat1

Figure 11-43. VIP Timing and Protocol

Table 11-22 VIP Timing Parameters

Parameter	Description	Min	Max
P	Clock frequency	-	33MHz
Th	Hold time for hctl and had	0ns	-
Ts	Setup time for hctl and had	5ns	-
Tr	Low to high transition time		5ns

11.13 UART (Serial Port) Timing

The two UARTs run off a base clock rate of 48 MHz (which is generated by the "PCU PLL" inside the clock-generator block of the Xilleon 220). Both UART ports have identical timing characteristics.

All outputs are driven by flip-flops based upon this clock. The serial data output signals (TDA for Port A; TDB for Port B) only switch at the frequency dictated by the programmable baud rate. The MODEM control output signals (DTRAb, RTSAb for Port A; and DTRBb, RTSBb for Port B) may switch on any 48 MHz clock edge, but since they only switch under software control, the actual switching rate is much lower.

The serial data input signals (RDA for Port A; RDB for Port B) are sampled at 16x the programmed baud rate, based upon the 48 MHz clock.

The MODEM control input signals (CTSAb, DCDAb, DSRAb, RIAb for Port A; and CTSBb, DCDBb, DSRBb, RIBb for Port B) are sampled on every-other rising edge of the 48 MHz clock, effectively achieving a 24 MHz sampling rate (for compatibility with legacy UARTs).

11.14 LPC Timing

Table 11-23 LPC Bus Input and Output Timing Specifications

Signal	Min (ns)	Max (ns)
LCLK Cycle Time	30 (33MHz)	
LCLK Clock High Time	11	-
LCLK Clock Low Time	11	-
Input Setup Timing		
Input: LPC signals: Setup Time to LCLK	9.5	-
Input Hold Timing		
Input: LPC signals: Hold Time from LCLK	0	-
Output Timing		
LCLK to Outputs: LPC signals: Valid Delay	-	11
LCLK to Outputs: LPC signals: Float to Active Delay	2	-
LCLK to Outputs: LPC signals: Active to Float Delay	-	28

11.14.1 Memory Read Cycle Timing

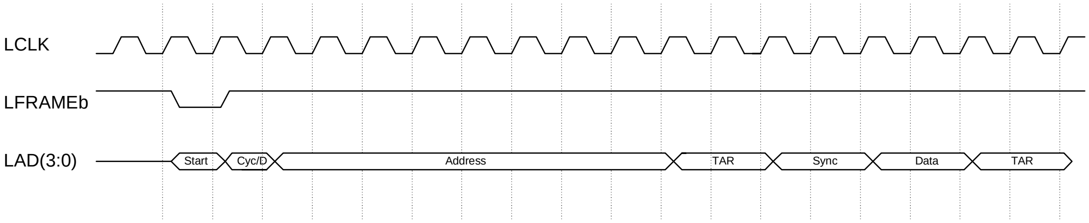


Figure 11-44. LPC Bus Memory Read Cycle

Table 11-24 Memory Read Cycle Timing

Mem Read	Driver	#Clocks	Comment
Start	Host	1	LAD=0000
Cycle type/Dir	Host	1	LAD(3:1)=010
Addr	Host	8	
Turn-around	Host	2	1st TAR clock: Host drives LAD to 1111, 2nd clock: Hi-Z w/ pull-up
Sync	Peripheral	1 -> n	Typically 5 clocks; optionally limited by sync timeout counter
Data	Peripheral	2	

Table 11-24 Memory Read Cycle Timing (Continued)

Mem Read	Driver	#Clocks	Comment
Turn-around	Peripheral	2	1st TAR clock: Peripheral drives LAD to 1111, 2nd clock: Hi-Z w/ pull-up

11.14.2 Memory Write Cycle Timing

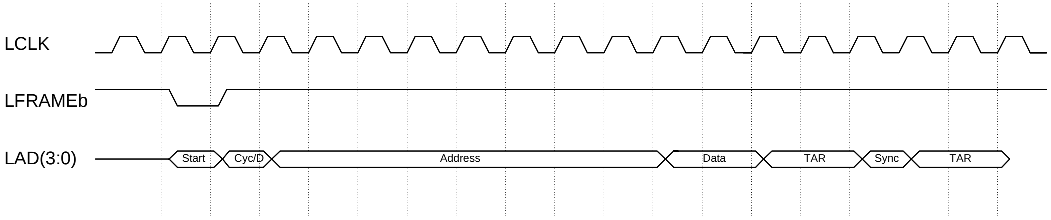


Figure 11-45. LPC Bus Memory Write Cycle

Table 11-25 Memory Write Cycle Timing

Mem Write	Driver	#Clocks	Comment
Start	Host	1	LAD=0000
Cycle type/Dir	Host	1	LAD(3:1)=011
Addr	Host	8	
Data	Host	2	
Turn-around	Host	2	1st TAR clock: Host drives LAD to 1111, 2nd clock: Hi-Z w/ pull-up
Sync	Peripheral	1	
Turn-around	Peripheral	2	1st TAR clock: Peripheral drives LAD to 1111, 2nd clock: Hi-Z w/ pull-up

11.14.3 IO Read Cycle Timing

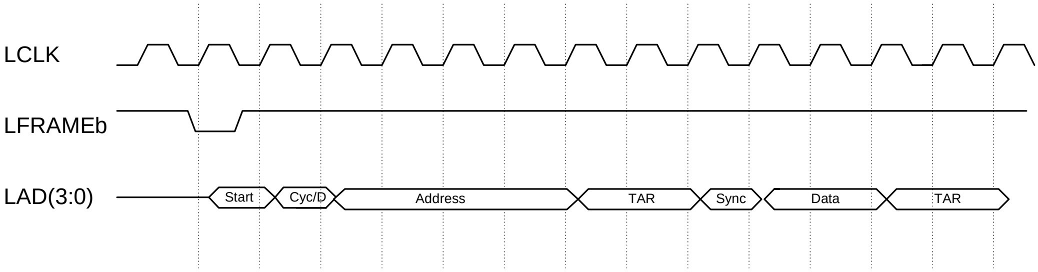


Figure 11-46. LPC Bus IO Read Cycle

Table 11-26 IO Read Cycle Timing

IO Read	Driver	#Clocks	Comment
Start	Host	1	LAD=0000
Cycle type/Dir	Host	1	LAD(3:1)=000
Addr	Host	4	
Turn-around	Host	2	1st TAR clock: Host drives LAD to 1111, 2nd clock: Hi-Z w/ pull-up
Sync	Peripheral	1->n	Optionally controlled by sync timeout counter
Data	Peripheral	2	
Turn-around	Peripheral	2	1st TAR clock: Peripheral drives LAD to 1111, 2nd clock: Hi-Z w/ pull-up

11.14.4 IO Write Cycle Timing

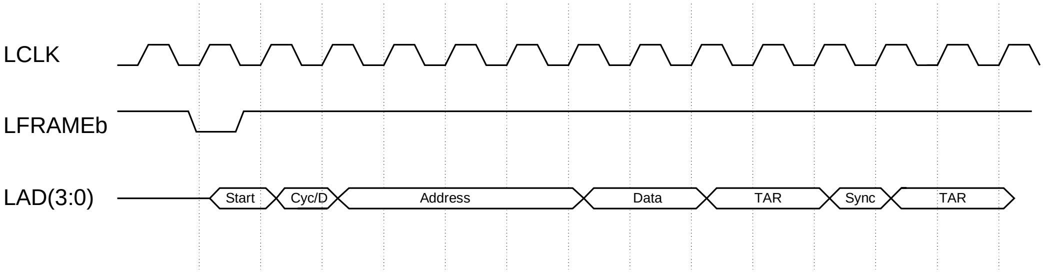


Figure 11-47. LPC Bus IO Write Cycle

Table 11-27 I/O Write Cycle Timing

IO Write	Driver	#Clocks	Comment
Start	Host	1	LAD=0000
Cycle type/Dir	Host	1	LAD(3:1)=001
Addr	Host	4	
Data	Host	2	
Turn-around	Host	2	1st TAR clock: Host drives LAD to 1111, 2nd clock: Hi-Z w/ pull-up
Sync	Peripheral	1	
Turn-around	Peripheral	2	1st TAR clock: Peripheral drives LAD to 1111, 2nd clock: Hi-Z w/ pull-up

11.15 FlexBus Timing

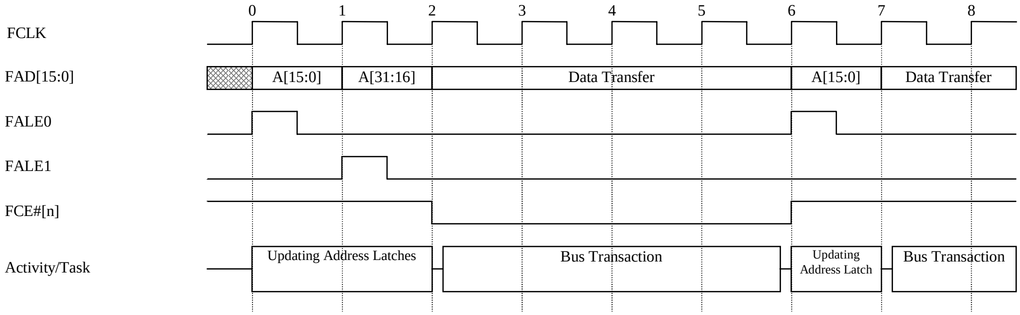


Figure 11-48. FlexBus Operation

The FlexBus Controller (FBC) manages the driving of the bidirectional FAD[] bus, to ensure that there is always one clock of no driving when turning around the direction of the bus. For example, if the first bus transaction shown in the figure above were a read cycle, the FBC would insert an additional cycle after clock edge 6, which would have the bus tri-state in order to turnaround the FAD bus from an input to an output. Then the address-latch updating would occur during the following clock period.

Some transactions on the FlexBus do not use the FRDY signal to control the length of the transaction on the bus, i.e., they take a fixed number of clocks, and the slave device has no mechanism to throttle the action of the master.

If there are several devices on the FlexBus which support the use of FRDY, then the FRDY signal must be an active-low open-collector (or open-drain) signal with an external pull-up. When an agent on the FlexBus determines that it is the slave device for a given transaction, it can drive the FRDY signal, otherwise FRDY is not driven by any device, and it is pulled-high by the pull-up resistor.

If only one FlexBus device has a Ready output, then it can be either active-high or active-low.

The FlexBus can directly support 8-bit devices as follows: Data is transferred on bits 7:0 of the data bus. The FBC must steer the data to/from the FlexBus to accommodate this case.

Table 11-28 FlexBus AC Specifications

	MIN	MAX	UNITS	NOTES1
Clock				
FCLK cycle time	15	100	ns	2
FCLK frequency	10	66	MHz	2
FCLK high time	6	40	ns	2
FCLK low time	6	40	ns	2

Table 11-28 FlexBus AC Specifications (Continued)

	MIN	MAX	UNITS	NOTES1
Output Times				
FCLK-to-FAD Delay	2.5	6.0	ns	3
FCLK-to-FBE Delay	2.5	6.0	ns	3
FCLK-to-FALE Delay	2.5	6.0	ns	3
FCLK-to-FSTB Delay	2.5	6.0	ns	3
FCLK-to-FRD Delay	2.5	6.0	ns	3
FCLK-to-FWE Delay	2.5	6.0	ns	3
FCLK-to-FCE# Delay	2.5	6.0	ns	3
FCLK-to-FGNT# Delay	2.5	6.0	ns	3
Setup Times				
FAD-to-FCLK Setup Time	3.0		ns	4
FRDY#-to-FCLK Setup Time	5.0		ns	4
FREQ#-to-FCLK Setup Time	3.0		ns	4
Hold Times				
FAD-to-FCLK Hold Time	1.0		ns	4
FRDY#-to-FCLK Hold Time	1.0		ns	4
FREQ#-to-FCLK Hold Time	1.0		ns	4

Notes:

1) All timing specs are referenced at the pins of the Xilleon 220 package. These numbers are very preliminary.

2) Clock output has maximum loading of 50pF.

3) Signal output has maximum loading of 50pF.

4) Signal input has maximum loading of 10pF.

11.15.1 Bus Request/Grant Protocol

The FBC supports external masters on the FlexBus using a simple request/grant protocol, similar to the HOLD/HLDA protocol that Intel defined for their microprocessor buses. The FBC provides an arbiter that receives FREQ# as an input and asserts FGNT# as an output. The FBC must tri-state the FlexBus signals before asserting FGNT#. While FGNT# and FREQ# are asserted, an external master may drive the FlexBus for transactions.

The figure below shows an example of the arbitration protocol. An external device asserts FREQ# when he wishes to gain ownership of the bus, and the arbiter responds by asserting FGNT#. The arbiter may take an indefinite (but finite) time to respond, depending upon the activity demands of the internal devices. As long as the external master holds FREQ# asserted he owns the bus. It is the responsibility of the external master to eventually surrender ownership by deasserting FREQ#, and tri-stating the bus at the same time. In response to FREQ# going away, the arbiter will deassert FGNT#, and the host will again own the bus and drive the tri-state signals.

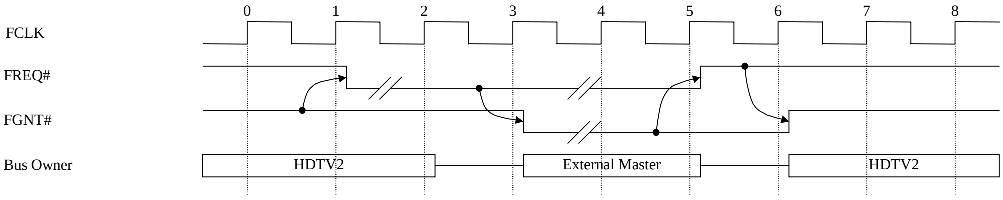


Figure 11-49. Master-initiated Surrender of Bus Ownership

The following signals that must be tri-stated by the relinquishing master:

FAD[15:0], FBE[1:0], FSTB#, FRD#, FWE#, and FCE#[5:0].

It may be necessary to add an external pull-up resistor on certain signals to keep them from floating to an intermediate state while no device is actively driving them, especially FSTB#, FRD#, FWE#, and FCE#[].

11.15.2 Address Bus Connection

8-bit and 16-bit devices must be wired to the address bus of the FlexBus as described below:

- 8-bit devices must have their two least-significant address pins wired to the FBE[1:0] pins of Xilleon 220; note that they should **not** be wired to the latch output corresponding to bit 0 of the FAD bus, i.e., A[0] of the device connects to FBE[0], and A[1] of the device connects to FBE[1].
- 16-bit devices must have their byte enable pins connected to FBE[1:0], and their A[1] input pin connected to the output of the latch on the FAD[1] signal, i.e., A[1] of the device connects to AQ[1], BE[0] of the device connects to FBE[0], and BE[1] of the device connects to FBE[1].

The reason that 8-bit devices cannot be connected to the output of an address latch is because, for 8-bit devices, the FlexBus Controller will **not** update the address latch (for higher performance) if only bits 1 or 0 of the address bus have changed.

The figure below shows address bus connection for 8-bit and 16-bit devices. Note that the FlexBus supports connection to both 8-bit and 16-bit devices in the same system.

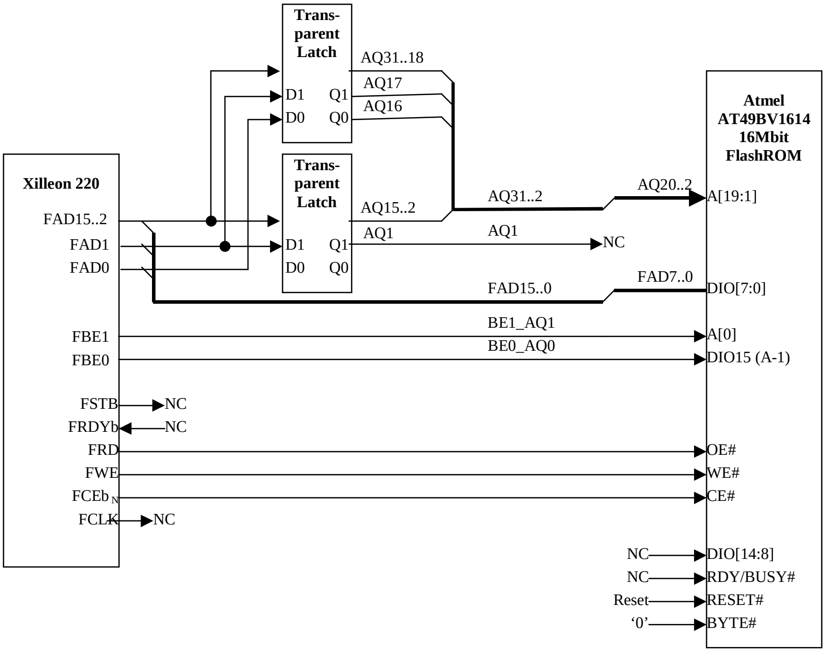


Figure 11-50. Address Bus Connection for 8-bit and 16-bit devices

11.15.3 Example Connection to an 8-bit Flash ROM Device

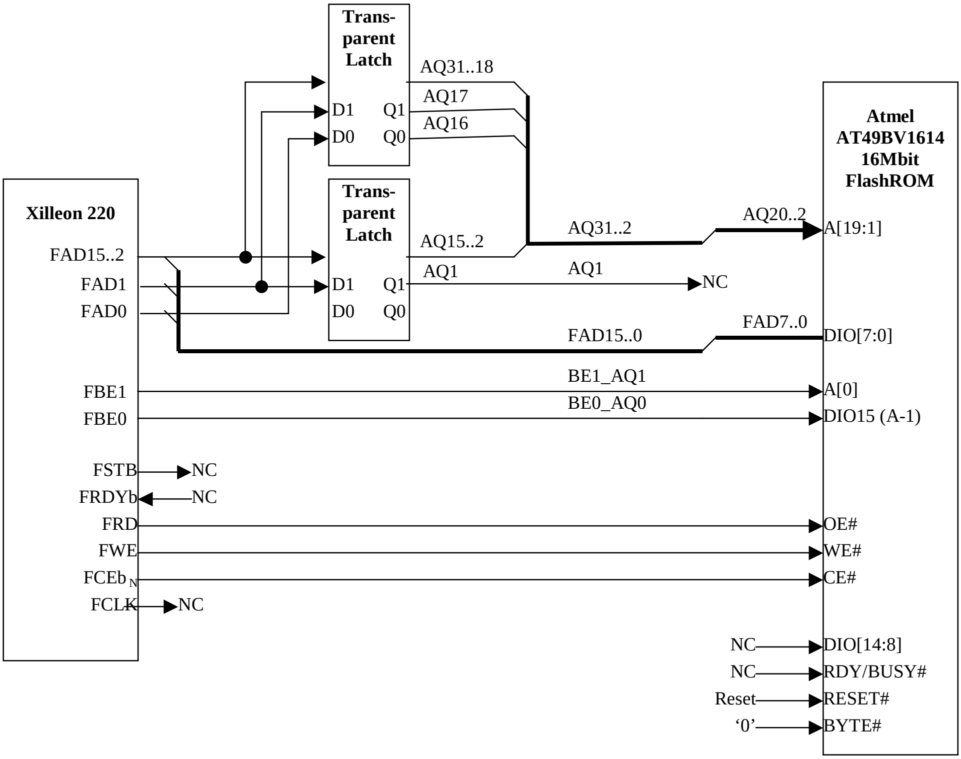


Figure 11-51. Example Connection to an 8-bit Flash ROM Device

Table 11-29 Microcode for Read Transaction

Instr. No	Opcode (binary)	Command	RDS	CE	STB	WE	RD	Notes
N	001_1110_0110	wait(8)	0	0	1	1	0	Assert RD, CE for 8 clocks
N+1	001_1111_0110	wait(8)	1	0	1	1	0	Assert RD, CE for 8 more clocks, capture data (RDS) on last clock
N+2	011_0010_1111	waith(2)	0	1	1	1	1	Deassert all signals for 2 clocks and halt

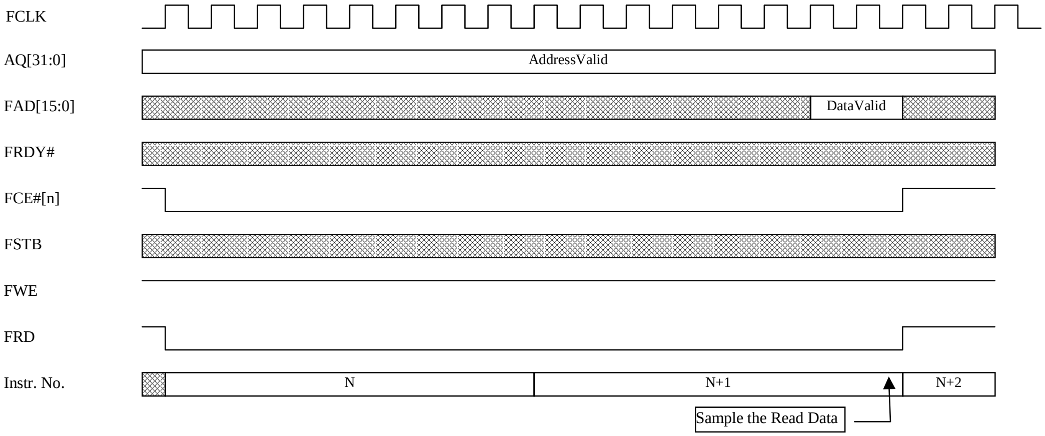


Figure 11-52. Timing Diagram for Read Transaction

Table 11-30 Microcode for Write Transaction

Instr. No	Opcode (binary)	Command	RDS	CE	STB	WE	RD	Notes
N	001_0000_1101	wait(1)	0	1	1	0	1	Assert WE for 1 clock
N+1	001_1100_0101	wait(7)	0	0	1	0	1	Assert WE, CE for 7 more clocks
N+2	001_0000_1101	wait(1)	0	1	1	0	1	Assert WE for 1 clock
N+3	011_0010_1111	waith(2)	0	1	1	1	1	Deassert all signals for 2 clocks and halt

The Flash ROMs typically have some setup-time from Address_Valid-to-CE_Asserted that is greater than or equal to zero. To be conservative, allow an entire clock cycle for this setup time.

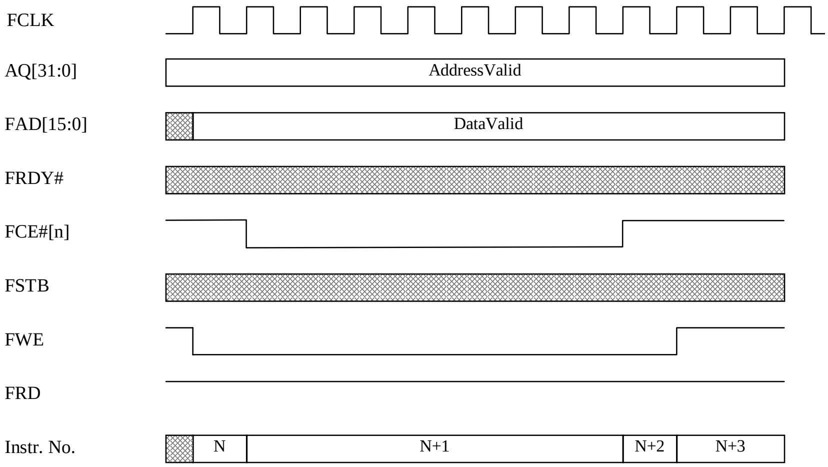


Figure 11-53. Timing Diagram for Write Transaction

11.15.4 Example Connection to a DOCSIS Media Access Control (MAC) Device

The LiBit 4230 device has a 16-bit data bus, and uses a Transmit Strobe/Transmit Acknowledge handshake to throttle transactions between the Xilleon 220 and itself.

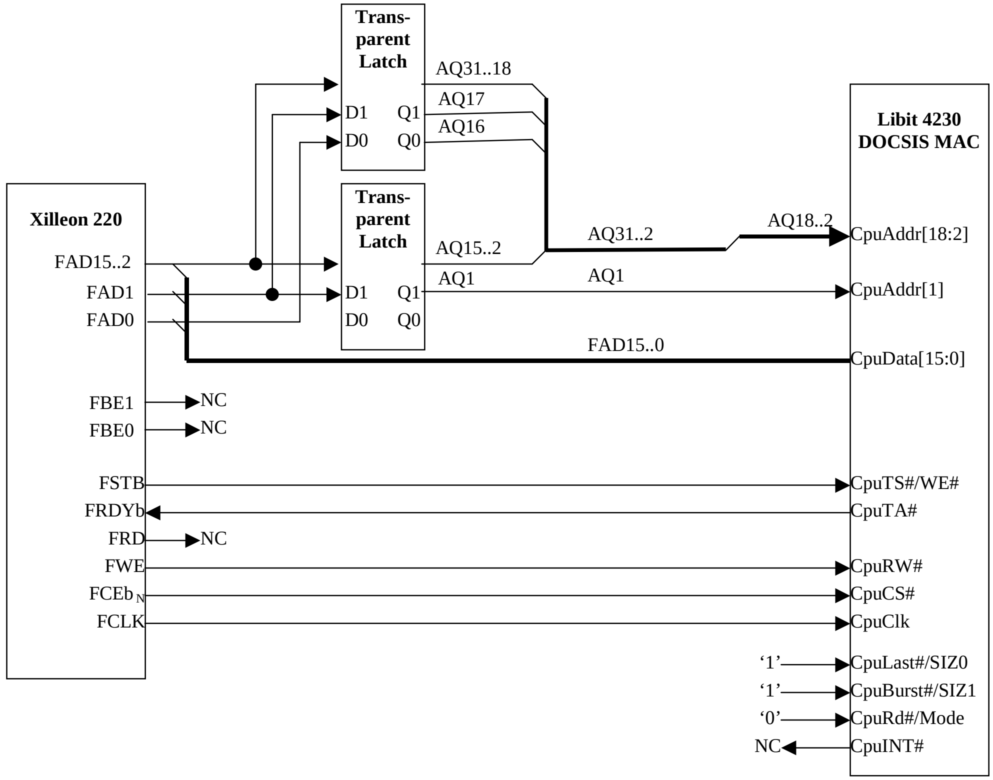


Figure 11-54. Example Connection to a DOCSIS Media Access Control Device

Table 11-31 Microcode for Read Transaction

Instr. No.	Opcode(binary)	Command	RDS	CE	STB	WE	RD	Notes
N	001_0000_0011	wait(1)	0	0	0	1	1	Assert CE, STB for 1 clock
N+1	010_0111_0111	wait_nfrdyh	1	0	1	1	1	Keep asserting CE, and wait until FRDY goes low. Capture data when FRDY goes low. Then halt.

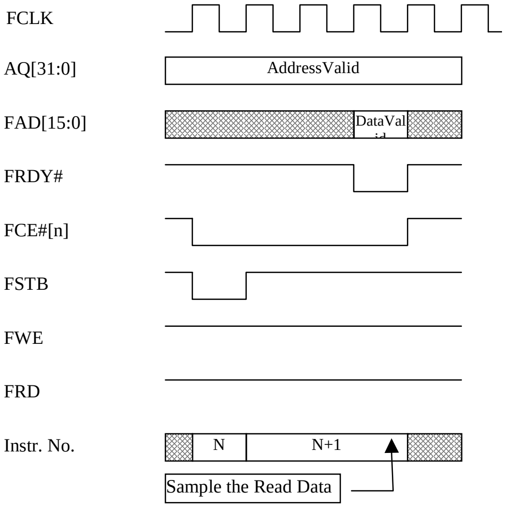


Figure 11-55. Timing Diagram for Read Transaction

Table 11-32 Microcode for Write Transaction

Instr. No.	Opcode (binary)	Command	RDS	CE	STB	WE	RD	Notes
N	001_0000_0001	wait(1)	0	0	0	0	1	Assert CE, STB, WE for 1 clock
N+1	000_0110_0101	wait_nfrdy	0	0	1	0	1	Keep asserting CE and WE, and wait until FRDY goes low.
N+2	011_0000_1111	waith(1)	0	1	1	1	1	De-assert CE and WE, and halt.

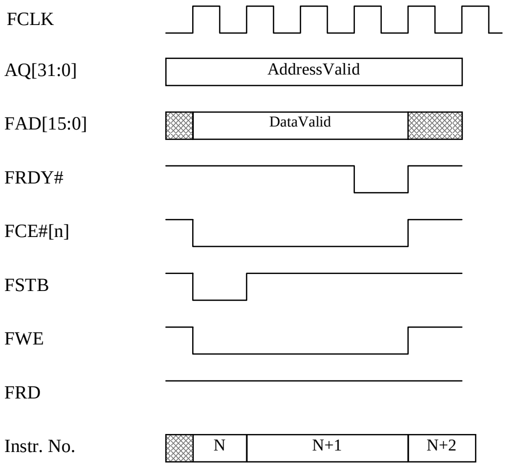


Figure 11-56. Timing Diagram for Write Transaction

11.16 USB Timing

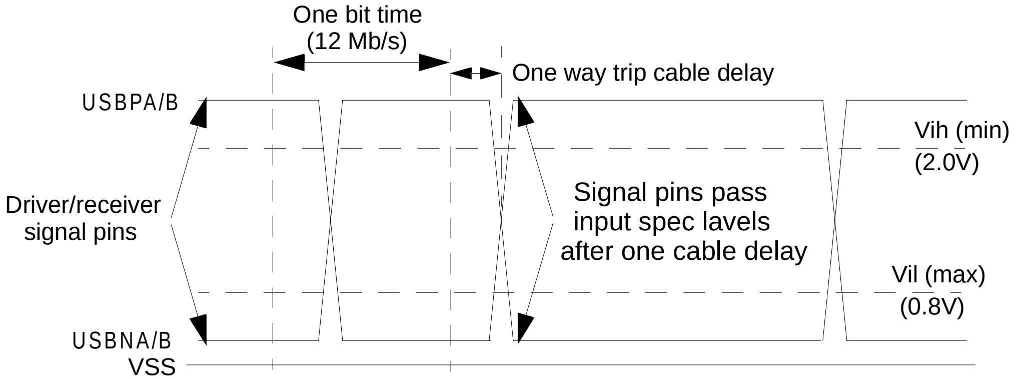


Figure 11-57. Full Speed Signal Waveform

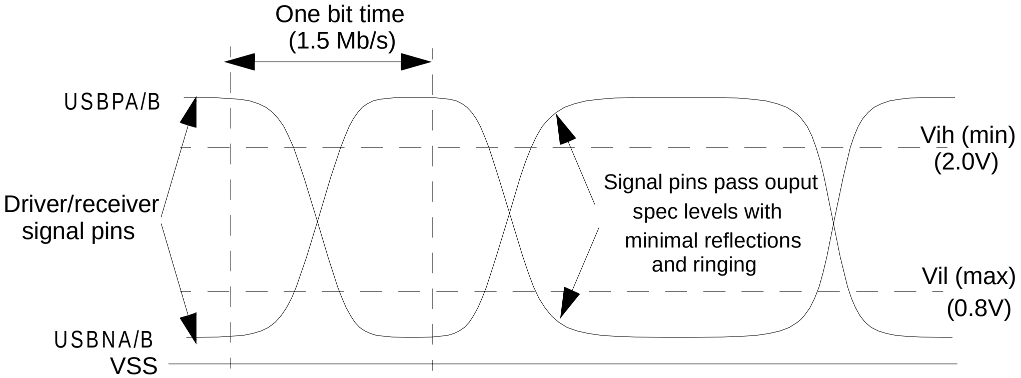


Figure 11-58. Low Speed Signal Waveform

11.17 IEEE 1394 LINK Layer Chip Timing

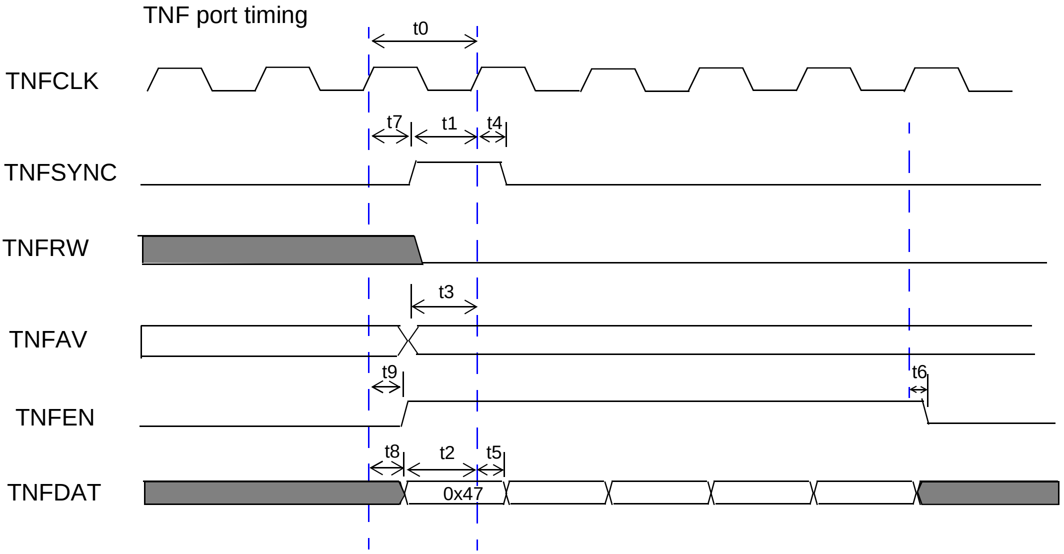


Figure 11-59. IEEE 1394 LINK Layer ChipTiming

Table 11-33 IEEE 1394 LINK Layer Chip Timing Parameters

Parameter	Description	min (ns)	max (ns)
t0	TNFCLK period (minimal XCLK*2)	12.0	1536.1
Input mode			
t1	TNFSYNC to TNFCLK setup time	0.8	
t2	TNFDATA to TNFCLK setup time	0.8	
t3	TNFAV to TNFCLK setup time	0.8	
t4	TNFSYNC to TNFCLK hold time	0.0	
t5	TNFDATA to TNFCLK hold time	0.0	
t6	TNFAV to TNFCLK hold time	0.0	
Output mode			
t7	TNFCLK to TNFSYNC valid time		8.0
t8	TNFCLK to TNFDATA valid time		8.0
t9	TNFCLK to TNFEN valid time		8.0

11.18 UIRT Timing

The UIRT runs off a base clock rate of 48 MHz (which is generated by the "PCU PLL" inside the clock-generator block of the Xilleon 220). The IRTXD output is driven by a flip-flop based upon this clock; and the IRRXD input is sampled at a programmable rate based upon this clock.

11.19 IDE Timing

11.19.1 PIO Mode Timing

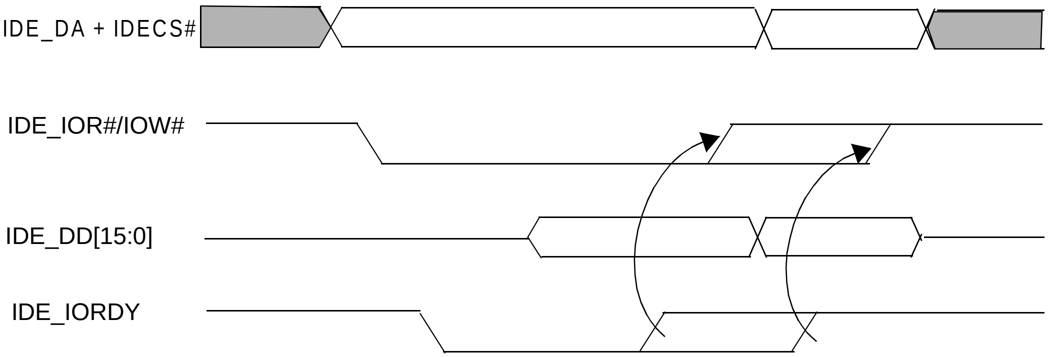


Figure 11-60. PIO Mode Cycle

Table 11-34 PIO Mode Timing Values

Mode	Timing (ns)
8 bit	960
0	600
1	383
2	240
3	180
4	120

11.19.2 DMA Mode Timing

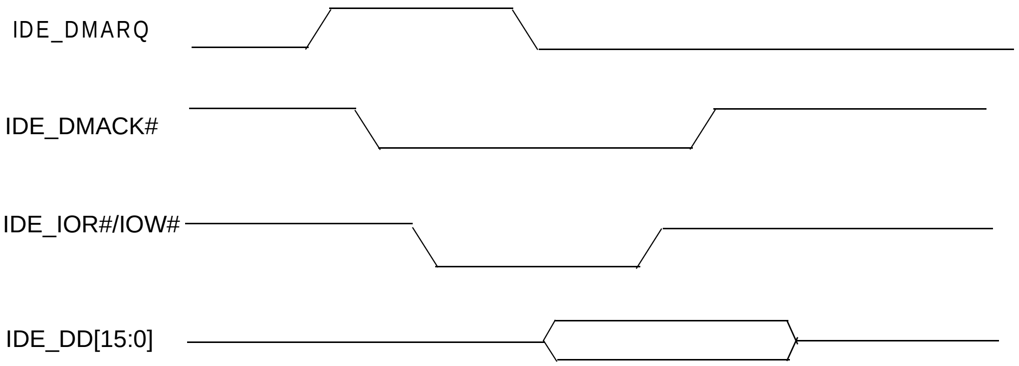
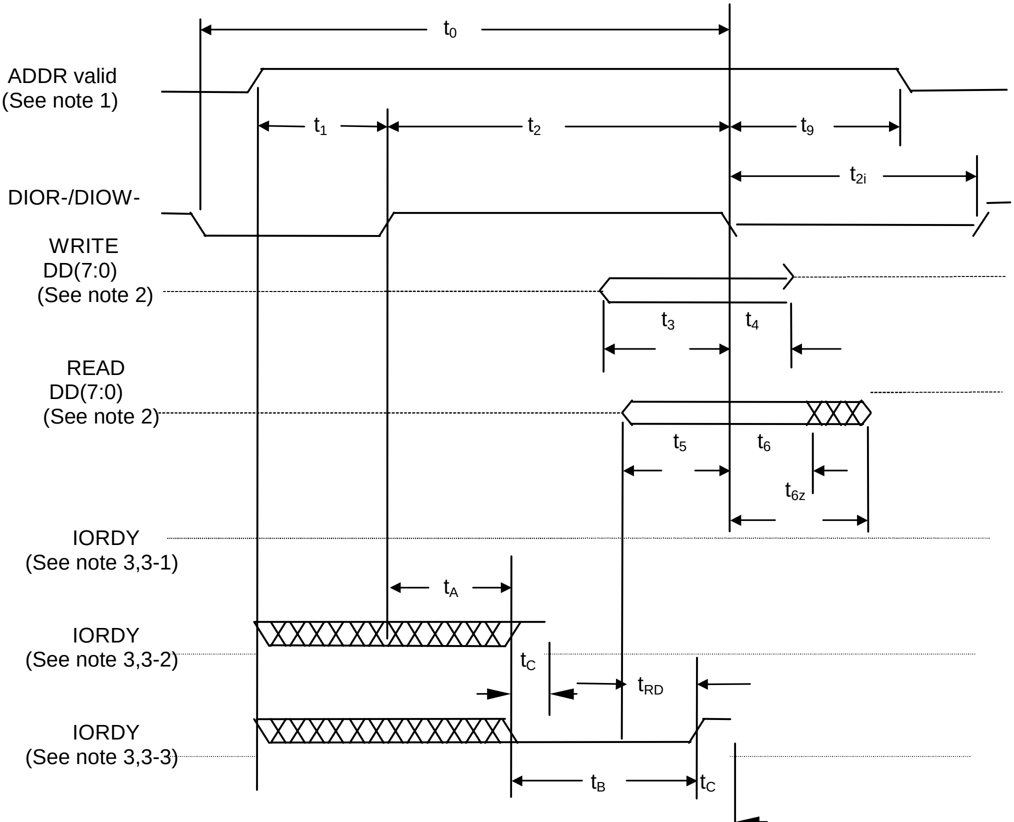


Figure 11-61. DMA Cycle

Table 11-35 DMA Mode Timing Values

Mode	Timing (ns)
Single Word 0	960
Single Word 1, Multi-word 0	480
Single Word 2	240
Multi-word 1	150
Multi-word 2	120

11.19.3 Register Transfer to/from Device



NOTES –

1 Device address consists of signals CS0-, CS1- and DA(2:0)

2 Data consists of DD(7:0).

3 The negation of IORDY by the device is used to extend the PIO cycle. The determination of whether the cycle is to be extended is made by the host after t_A from the assertion of DIOR- or DIOW-. The assertion and negation of IORDY are described in the following three cases:

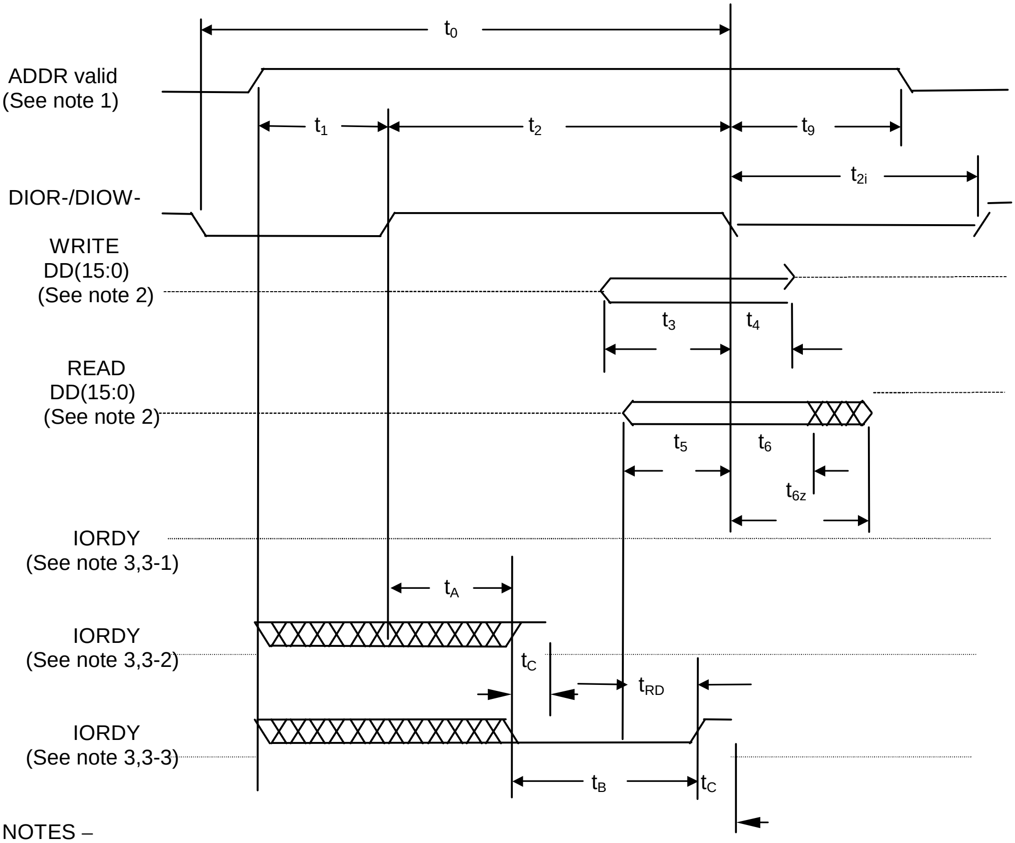
3-1 Device never negates IORDY, devices keeps IORDY released: no wait is generated.

3-2 Device negates IORDY before t_A, but causes IORDY to be asserted before t_A. IORDY is released prior to negation and may be asserted for no more than 5 ns before release: no wait generated.

3-3 Device negates IORDY before t_A. IORDY is released prior to negation and may be asserted for no more than 5 ns before release: wait generated. The cycle completes after IORDY is reasserted. For cycles where a wait is generated and DIOR- is asserted, the device shall place read data on DD(7:0) for t_{RD} before asserting IORDY.

Figure 11-62. Register Transfer to/from Device

11.19.4 PIO Data Transfer to/from Device



NOTES –

1 Device address consists of signals CS0-, CS1- and DA(2:0)

2 Data consists of DD(15:0).

3 The negation of IORDY by the device is used to extend the PIO cycle. The determination of whether the cycle is to be extended is made by the host after t_A from the assertion of DIOR- or DIOW-. The assertion and negation of IORDY are described in the following three cases:

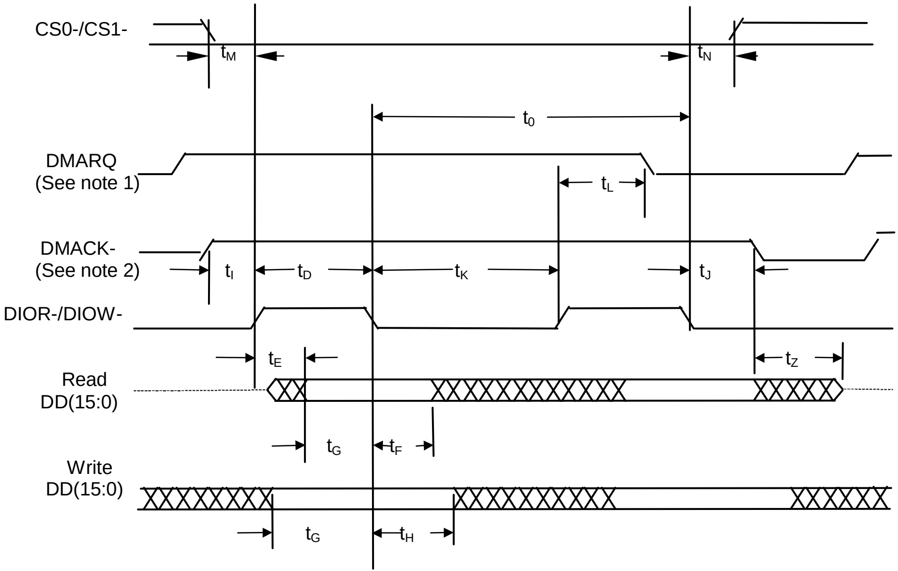
3-1 Device never negates IORDY, devices keeps IORDY released: no wait is generated.

3-2 Device negates IORDY before t_A, but causes IORDY to be asserted before t_A. IORDY is released prior to negation and may be asserted for no more than 5 ns before release: no wait generated.

3-3 Device negates IORDY before t_A. IORDY is released prior to negation and may be asserted for no more than 5 ns before release: wait generated. The cycle completes after IORDY is reasserted. For cycles where a wait is generated and DIOR- is asserted, the device shall place read data on DD(15:0) for t_{RD} before asserting IORDY.

Figure 11-63. PIO Data Transfer to/from Device

11.19.5 Multiword DMA Data Transfer



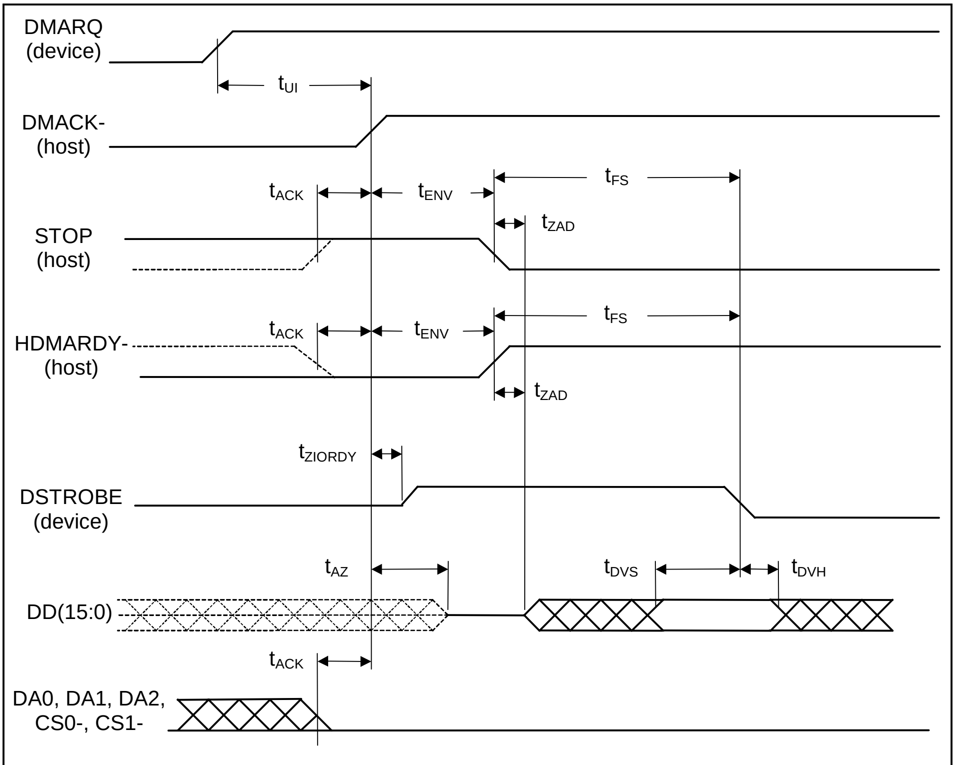
NOTE –

1 For Multi-Word DMA transfers, the Device may negate DMARQ within the t_L specified time once DMACK- is asserted and reassert it again at a later time to resume the DMA operation. Alternatively, if the device is able to continue the transfer of data, the device may leave DMARQ asserted and wait for the host to reassert DMACK-.

2 This signal may be negated by the Host to suspend the DMA transfer in process.

Figure 11-64. Multiword DMA Data Transfer

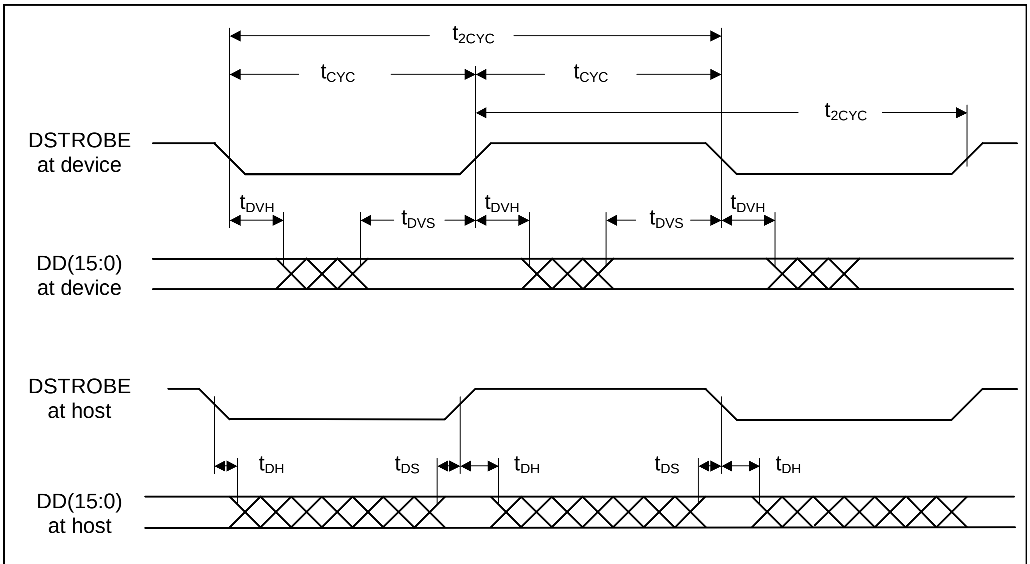
11.19.6 Initiating an Ultra DMA Data-in Burst



NOTE –
The definitions for the STOP, HDMARDY and DSTROBE signal lines are not in effect until DMARQ and DMACK are asserted.

Figure 11-65. Initiating an Ultra DMA Data-in Burst

11.19.7 Sustained Ultra DMA Data-in Burst



NOTE –
DD(15:0) and DSTROBE signals are shown at both the host and the device to emphasize that cable settling time as well as cable propagation delay shall not allow the data signals to be considered stable at the host until some time after they are driven by the device.

Figure 11-66. Sustained Ultra DMA Data-in Burst

11.20 EJTAG Port Timing

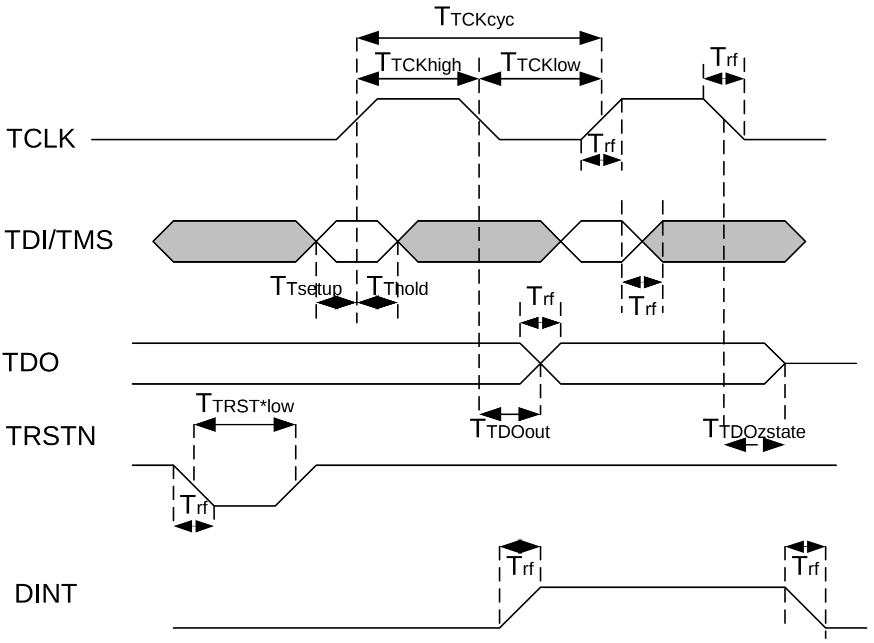


Figure 11-67. EJTAG Port Timing

Table 11-36 EJTAG Port Timing Parameters

Parameter	Description	Min (ns)	Max (ns)
TTCKcyc	TCK cycle time	25	
TTCKhigh	TCK high time	10	
TTCKlow	TCK low time	10	
TTsetup	TDI/TMS setup time before rising edge of TCK	5	
TThold	TDI/TMS hold time after rising edge of TCK	3	
TTDOout	TDO output delay time after falling edge of TCK		5
TTDOzstate	TDO 3 state delay time after falling edge of TCK		5
TTRSTlow	TRST low time	25	
Trf	EJTAG signals rise and fall time		3

Chapter 12
EXOR Tree

12.1 Brief Description of an EXOR Tree

A sample of a generic EXOR tree is shown in the figure below.

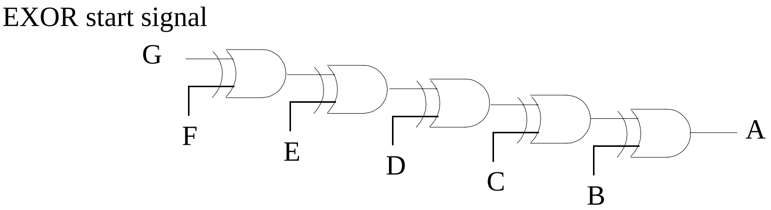


Figure 12-1. Sample of an EXOR Tree

Pin A is assigned to the output direction, and pins B through F are assigned to the input direction. It can be seen that after all pins B to F are assigned a logic '0' or '1', a logic change in any one of these pins will toggle the output pin A.

Table 12-1 EXOR Tree Truth Table

G	F	E	D	C	B	A
0	0	0	0	0	0	0
1	0	0	0	0	0	1
1	1	0	0	0	0	0
1	1	1	0	0	0	1
1	1	1	1	0	0	0
1	1	1	1	1	0	1
1	1	1	1	1	1	0

12.2 Description of the EXOR Tree for Xilleon 220

For Xilleon 220, one EXOR tree is implemented. The following gives a description of the connections of the EXOR tree and its activation.

12.2.1 Connections

The EXOR start signal (ref Fig 23-1) comes from GPIOB(16) pin which is TDI pin of the JTAG circuitry.

Several pins are excluded from the EXOR tree. For obvious reason, all the power and ground pins, and analog pins are not part of any EXOR tree. The following table shows the remaining pins not included in any EXOR tree.

Table 12-2 Pins Excluded from the EXOR Tree

Pin Name	Reason
XTALIN	PLL might be active during Reset
XTALOUT	Connected to XTALIN
GPIOB(19:13) , TESTEN	JTAG Pins
GPIOB(7)	Clock pin in test mode
INTROUTb	Interrupt out
RESETb	Reset
PCICLK	PCI clock in
IDSEL	ID select
VSBPA	VSB pin
VSBNA	VSB pin
VSBPB	VSB pin
VSBNB	VSB pin

12.2.2 EXOR Tree Activation

The Xilleon 220 chip enters the EXOR tree testmode by means of JTAG. First, the 8-bit instruction register of the JTAG is loaded with the EXOR instruction ("00001000"). This instruction will assign the input direction to all the pins except pin GPIOB(15), which is assigned the output direction to serve as the output of the EXOR tree. After loading, the JTAG is taken to the Run-Test state for completion of the EXOR tree initialization.

12.2.3 Unused Pins

If any pins hooked up in the EXOR tree are unused, they must be internally pulled up or down before the EXOR tree is activated. This is carried out by loading the instruction register with the pull-up instruction ("00110000") or the pull-down instruction ("00110011"). The EXOR tree is then activated as described earlier.

Appendix A

Pinout Listing

This appendix contains pin listings for Xilleon 220 sorted in two ways. To go to the listing of interest, use the following linked cross references:

“Pins Sorted by Signal Name” on page A-2

“Pins Sorted by Ball Reference” on page A-10

Table A-1 Pins Sorted by Signal Name

Signal Name	Ball Ref
A2VDD	AG22
A2VDD	AG23
A2VDDQ	AE21
A2VSSN	AE23
A2VSSN	AF22
A2VSSQ	AE22
AA(0)	B27
AA(1)	A27
AA(10)	D23
AA(11)	C23
AA(12)	B23
AA(13)	A23
AA(14)	D22
AA(2)	C26
AA(3)	B26
AA(4)	A26
AA(5)	D25
AA(6)	C25
AA(7)	B25
AA(8)	A25
AA(9)	D24
AB(0)	A9
AB(1)	B9
AB(10)	B6
AB(11)	C6
AB(12)	D6
AB(13)	A5
AB(14)	B5
AB(2)	C9
AB(3)	D9
AB(4)	A8
AB(5)	B8
AB(6)	C8
AB(7)	D8
AB(8)	A7
AB(9)	B7

Table A-1 Pins Sorted by Signal Name (Cont’d)

Signal Name	Ball Ref
ACBIT_CLK	AJ11
ACDATA_IN(0)	AH12
ACDATA_IN(1)	AG12
ACDATA_OUT	AF12
ACRESETb	AJ12
ACSYNC	AH11
ACXTAL_OUT	AF13
AD(0)	G27
AD(1)	G28
AD(10)	K26
AD(11)	K27
AD(12)	K28
AD(13)	K29
AD(14)	L26
AD(15)	L27
AD(16)	P26
AD(17)	P27
AD(18)	P28
AD(19)	P29
AD(2)	G29
AD(20)	R29
AD(21)	R28
AD(22)	R27
AD(23)	T28
AD(24)	T27
AD(25)	T26
AD(26)	U29
AD(27)	U28
AD(28)	U26
AD(29)	U27
AD(3)	H26
AD(30)	V29
AD(31)	V28
AD(4)	H27
AD(5)	H28
AD(6)	H29
AD(7)	J26

Table A-1 Pins Sorted by Signal Name (Cont'd)

Signal Name	Ball Ref
AD(8)	J28
AD(9)	J29
APVDD	AF4
APVSS	AG4
AVDD	AF24
AVDD	AF25
AVDDQ	AG24
AVSSN	AD25
AVSSN	AE25
AVSSQ	AF23
CASAb	C21
CASBb	C4
CBEb(0)	J27
CBEb(1)	L28
CBEb(2)	N29
CBEb(3)	R26
CKEA	C22
CKEB	C5
CLKA	C24
CLKAb	B24
CLKB	C7
CLKBb	D7
CLKFBA	A24
CLKFBB	A6
CSAb(0)	B22
CSAb(1)	A22
CSBb(0)	D5
CSBb(1)	A4
DAC0	AH24
DAC1	AJ24
DAC2	AH25
DAC3	AG25
DAC4	AH23
DAC5	AH22
DAC6	AJ22
DEVSELb	M29
DQA(0)	G26

Table A-1 Pins Sorted by Signal Name (Cont'd)

Signal Name	Ball Ref
DQA(1)	F29
DQA(10)	D26
DQA(11)	C29
DQA(12)	B29
DQA(13)	B28
DQA(14)	A29
DQA(15)	A28
DQA(16)	A21
DQA(17)	D20
DQA(18)	C20
DQA(19)	B20
DQA(2)	F28
DQA(20)	C19
DQA(21)	B19
DQA(22)	A19
DQA(23)	D18
DQA(24)	C18
DQA(25)	B18
DQA(26)	A18
DQA(27)	D17
DQA(28)	A17
DQA(29)	D16
DQA(3)	F27
DQA(30)	C16
DQA(31)	B16
DQA(4)	E28
DQA(5)	E27
DQA(6)	E26
DQA(7)	D29
DQA(8)	D28
DQA(9)	D27
DQB(0)	A14
DQB(1)	B14
DQB(10)	A11
DQB(11)	B11
DQB(12)	A10
DQB(13)	B10

Table A-1 Pins Sorted by Signal Name (Cont’d)

Signal Name	Ball Ref
DQB(14)	C10
DQB(15)	D10
DQB(16)	A2
DQB(17)	A1
DQB(18)	B3
DQB(19)	B2
DQB(2)	C14
DQB(20)	D4
DQB(21)	D3
DQB(22)	C2
DQB(23)	C1
DQB(24)	D2
DQB(25)	D1
DQB(26)	E4
DQB(27)	E3
DQB(28)	F3
DQB(29)	F4
DQB(3)	D14
DQB(30)	F2
DQB(31)	F1
DQB(4)	C13
DQB(5)	D13
DQB(6)	A12
DQB(7)	B12
DQB(8)	C12
DQB(9)	D12
DQMAb(0)	F26
DQMAb(1)	C27
DQMAb(2)	A20
DQMAb(3)	C17
DQMBb(0)	A13
DQMBb(1)	C11
DQMBb(2)	B1
DQMBb(3)	E2
FAD(0)	T1
FAD(1)	R3
FAD(10)	N4

Table A-1 Pins Sorted by Signal Name (Cont’d)

Signal Name	Ball Ref
FAD(11)	N3
FAD(12)	M1
FAD(13)	M2
FAD(14)	M3
FAD(15)	L1
FAD(2)	R2
FAD(3)	R1
FAD(4)	P1
FAD(5)	P2
FAD(6)	P3
FAD(7)	P4
FAD(8)	N1
FAD(9)	N2
FALE(0)	T4
FALE(1)	T3
FBE(0)	T2
FBE(1)	R4
FCEb(0)	W2
FCEb(1)	V4
FCEb(2)	W1
FCEb(3)	V3
FCEb(4)	V2
FCEb(5)	V1
FCLK	M4
FGNTb	L2
FRAMEb	N28
FRD	U4
FRDYb	U2
FREQb	L3
FSTB	U1
FWE	U3
GNTb(0)	W28
GNTb(1)	W27
GNTb(2)	Y26
GPIOA(0)	AF10
GPIOA(1)	AJ9
GPIOA(10)	AF8

Table A-1 Pins Sorted by Signal Name (Cont'd)

Signal Name	Ball Ref
GPIOA(11)	AG7
GPIOA(12)	AF7
GPIOA(13)	AJ6
GPIOA(2)	AH9
GPIOA(3)	AG9
GPIOA(4)	AF9
GPIOA(5)	AJ8
GPIOA(6)	AH8
GPIOA(7)	AG8
GPIOA(8)	AJ7
GPIOA(9)	AH7
GPIOB(0)	AE26
GPIOB(1)	AE28
GPIOB(10)	AH28
GPIOB(11)	AJ28
GPIOB(12)	AJ29
GPIOB(13)	AG27
GPIOB(14)	AH27
GPIOB(15)	AJ27
GPIOB(16)	AF29
GPIOB(17)	AF28
GPIOB(18)	AH26
GPIOB(19)	AJ26
GPIOB(2)	AE27
GPIOB(3)	AE29
GPIOB(4)	AG26
GPIOB(5)	AF26
GPIOB(6)	AG29
GPIOB(7)	AG28
GPIOB(8)	AH29
GPIOB(9)	AF27
I2CCLKA	AJ5
I2CCLKB	AG6
I2CDATAA	AF6
I2CDATAB	AH6
I2SSCK_IN	AJ1
I2SSCK_OUTA	AJ4

Table A-1 Pins Sorted by Signal Name (Cont'd)

Signal Name	Ball Ref
I2SSD_IN	AJ3
I2SSD_OUTA	AG5
I2SSOSCK_OUTA	AH5
I2SWS_IN	AJ2
I2SWS_OUTA	AF5
IDE_CSb(0)	AG15
IDE_CSb(1)	AH15
IDE_DA(0)	AJ15
IDE_DA(1)	AG14
IDE_DA(2)	AH14
IDE_DD(0)	AH19
IDE_DD(1)	AE18
IDE_DD(10)	AJ17
IDE_DD(11)	AF16
IDE_DD(12)	AG16
IDE_DD(13)	AF15
IDE_DD(14)	AJ16
IDE_DD(15)	AH16
IDE_DD(2)	AJ19
IDE_DD(3)	AF18
IDE_DD(4)	AG18
IDE_DD(5)	AH18
IDE_DD(6)	AJ18
IDE_DD(7)	AF17
IDE_DD(8)	AG17
IDE_DD(9)	AH17
IDE_DMACKb	AE19
IDE_DMARQ	AH20
IDE_INTRQ	AG20
IDE_IORb	AG19
IDE_IORDY	AJ20
IDE_IOWb	AF19
IDSEL	T29
INTRb(0)	AB27
INTRb(1)	AB29
INTRb(2)	AB28
INTROUTb	AA26

Table A-1 Pins Sorted by Signal Name (Cont'd)

Signal Name	Ball Ref
IRDYb	N27
IRRXD	K1
IRTXD	L4
LAD(0)	AD29
LAD(1)	AD28
LAD(2)	AD27
LAD(3)	AD26
LCLK	AC27
LDRQb(0)	AC29
LDRQb(1)	AC28
LFRAMEb	AC26
MEMTEST	C15
MEMVMODE	D15
NRSSCLKA	W3
NRSSCLKB	AB4
NRSSDATA_INA	W4
NRSSDATA_INB	AC3
NRSSDATA_OUTA	Y1
NRSSDATA_OUTB	AC4
PAR	L29
PCICLK	AA29
PCICLKOUT(0)	Y29
PCICLKOUT(1)	Y28
PCICLKOUT(2)	Y27
PCICLKOUT(3)	AA28
PERRb	M27
PMIVDD	AH21
PMIVSS	AG21
PWMA	AG3
PWMB	AH4
QSA(0)	E29
QSA(1)	C28
QSA(2)	D19
QSA(3)	B17
QSB(0)	B13
QSB(1)	D11
QSB(2)	C3

Table A-1 Pins Sorted by Signal Name (Cont'd)

Signal Name	Ball Ref
QSB(3)	E1
R2SET	AJ23
RASAb	D21
RASBb	B4
RDB	AF20
REQb(0)	V27
REQb(1)	V26
REQb(2)	W29
RESETb	AA27
RESETOUTb	W26
RSET	AJ25
SERIRQb	AB26
SERRb	M26
SMCLKA	G4
SMCLKB	H3
SMCLKS	J2
SMDATAA	G1
SMDATAB	J3
SMDATAS	K2
SMDETECTA	G2
SMDETECTB	J4
SMDETECTS	K3
SMRSTA	G3
SMRSTB	H2
SMRSTS	J1
SMVCCA	H4
SMVCCB	H1
SMVCCS	K4
STOPb	M28
TCLKA	Y2
TCLKB	AD1
TDATA(0)	AB2
TDATA(1)	AB1
TDATA(2)	AA4
TDATA(3)	AA3
TDATA(4)	AA2
TDATA(5)	AA1

Table A-1 Pins Sorted by Signal Name (Cont'd)

Signal Name	Ball Ref
TDATA(6)	Y4
TDATA(7)	Y3
TDATB(0)	AF1
TDATB(1)	AF2
TDATB(2)	AE3
TDATB(3)	AE2
TDATB(4)	AE1
TDATB(5)	AD4
TDATB(6)	AD3
TDATB(7)	AD2
TDB	AJ21
TERA	AC2
TERB	AG1
TESTEN	AG10
TRDYb	N26
TSTRTA	AB3
TSTRTB	AE4
TVLDA	AC1
TVLDB	AF3
USB_OVRCUR	AJ14
USBNA	AH13
USBNB	AF14
USBPA	AJ13
USBPB	AG13
VCXO_INA	AG2
VCXO_INB	AH1
VDD2DI	AF21
VDDC	E9
VDDC	E10
VDDC	E13
VDDC	E14
VDDC	E16
VDDC	E17
VDDC	E20
VDDC	E21
VDDC	K5
VDDC	K25

Table A-1 Pins Sorted by Signal Name (Cont'd)

Signal Name	Ball Ref
VDDC	L5
VDDC	L25
VDDC	R5
VDDC	R25
VDDC	T5
VDDC	T25
VDDC	U5
VDDC	U25
VDDC	Y5
VDDC	Y25
VDDC	AA5
VDDC	AA25
VDDC	AD16
VDDC	AD20
VDDC	AD21
VDDC	AE9
VDDC	AE10
VDDC	AE13
VDDC	AE14
VDDC	AE16
VDDC18_BOTTOM	P25
VDDC18_LEFT	AE15
VDDC18_RIGHT	E15
VDDC18_TOP	P5
VDDDI	AE24
VDDP	H25
VDDP	J25
VDDP	M25
VDDP	N25
VDDP	V25
VDDP	W25
VDDP	AB25
VDDP	AC24
VDDP	AC25
VDDR1	E5
VDDR1	E6
VDDR1	E7

Table A-1 Pins Sorted by Signal Name (Cont’d)

Signal Name	Ball Ref
VDDR1	E8
VDDR1	E11
VDDR1	E12
VDDR1	E18
VDDR1	E19
VDDR1	E22
VDDR1	E23
VDDR1	E24
VDDR1	E25
VDDR1	F5
VDDR1	F25
VDDR1	G5
VDDR1	G25
VDDR3	H5
VDDR3	J5
VDDR3	M5
VDDR3	N5
VDDR3	V5
VDDR3	W5
VDDR3	AB5
VDDR3	AC5
VDDR3	AD5
VDDR3	AD17
VDDR3	AD18
VDDR3	AD19
VDDR3	AD22
VDDR3	AD23
VDDR3	AE5
VDDR3	AE6
VDDR3	AE7
VDDR3	AE8
VDDR3	AE11
VDDR3	AE12
VDDR3	AE17
VIPCLK	AF11
VIPHAD(0)	AJ10
VIPHAD(1)	AH10

Table A-1 Pins Sorted by Signal Name (Cont’d)

Signal Name	Ball Ref
VIPHCTL	AG11
VREF	B15
VSS	L11
VSS	L12
VSS	L13
VSS	L14
VSS	L15
VSS	L16
VSS	L17
VSS	L18
VSS	L19
VSS	M11
VSS	M12
VSS	M13
VSS	M14
VSS	M15
VSS	M16
VSS	M17
VSS	M18
VSS	M19
VSS	N11
VSS	N12
VSS	N13
VSS	N14
VSS	N15
VSS	N16
VSS	N17
VSS	N18
VSS	N19
VSS	P11
VSS	P12
VSS	P13
VSS	P14
VSS	P15
VSS	P16
VSS	P17
VSS	P18

Table A-1 Pins Sorted by Signal Name (Cont'd)

Signal Name	Ball Ref
VSS	P19
VSS	R11
VSS	R12
VSS	R13
VSS	R14
VSS	R15
VSS	R16
VSS	R17
VSS	R18
VSS	R19
VSS	T11
VSS	T12
VSS	T13
VSS	T14
VSS	T15
VSS	T16
VSS	T17
VSS	T18
VSS	T19
VSS	U11
VSS	U12
VSS	U13
VSS	U14
VSS	U15
VSS	U16
VSS	U17
VSS	U18
VSS	U19
VSS	V11
VSS	V12
VSS	V13
VSS	V14
VSS	V15
VSS	V16
VSS	V17
VSS	V18
VSS	V19

Table A-1 Pins Sorted by Signal Name (Cont'd)

Signal Name	Ball Ref
VSS	W11
VSS	W12
VSS	W13
VSS	W14
VSS	W15
VSS	W16
VSS	W17
VSS	W18
VSS	W19
VSS2DI	AE20
VSSDI	AD24
WEAb	B21
WEBb	A3
XMVDD	A15
XMVSS	A16
XTALIN	AH2
XTALOUT	AH3

Table A-2 Pins Sorted by Ball Reference

Ball Ref	Signal Name
A1	DQB(17)
A10	DQB(12)
A11	DQB(10)
A12	DQB(6)
A13	DQMBb(0)
A14	DQB(0)
A15	XMVDD
A16	XMVSS
A17	DQA(28)
A18	DQA(26)
A19	DQA(22)
A2	DQB(16)
A20	DQMAb(2)
A21	DQA(16)
A22	CSAb(1)
A23	AA(13)
A24	CLKFBA
A25	AA(8)
A26	AA(4)
A27	AA(1)
A28	DQA(15)
A29	DQA(14)
A3	WEBb
A4	CSBb(1)
A5	AB(13)
A6	CLKFBB
A7	AB(8)
A8	AB(4)
A9	AB(0)
AA1	TDATA(5)
AA2	TDATA(4)
AA25	VDDC
AA26	INTROUTb
AA27	RESETb
AA28	PCICLKOUT(3)
AA29	PCICLK

Table A-2 Pins Sorted by Ball Reference (Cont'd)

Ball Ref	Signal Name
AA3	TDATA(3)
AA4	TDATA(2)
AA5	VDDC
AB1	TDATA(1)
AB2	TDATA(0)
AB25	VDDP
AB26	SERIRQb
AB27	INTRb(0)
AB28	INTRb(2)
AB29	INTRb(1)
AB3	TSTRTA
AB4	NRSSCLKB
AB5	VDDR3
AC1	TVLDA
AC2	TERA
AC24	VDDP
AC25	VDDP
AC26	LFRAMEb
AC27	LCLK
AC28	LDRQb(1)
AC29	LDRQb(0)
AC3	NRSSDATA_INB
AC4	NRSSDATA_OUTB
AC5	VDDR3
AD1	TCLKB
AD16	VDDC
AD17	VDDR3
AD18	VDDR3
AD19	VDDR3
AD2	TDATB(7)
AD20	VDDC
AD21	VDDC
AD22	VDDR3
AD23	VDDR3
AD24	VSSDI
AD25	AVSSN
AD26	LAD(3)

Table A-2 Pins Sorted by Ball Reference (Cont’d)

Ball Ref	Signal Name
AD27	LAD(2)
AD28	LAD(1)
AD29	LAD(0)
AD3	TDATB(6)
AD4	TDATB(5)
AD5	VDDR3
AE1	TDATB(4)
AE10	VDDC
AE11	VDDR3
AE12	VDDR3
AE13	VDDC
AE14	VDDC
AE15	VDDC18_LEFT
AE16	VDDC
AE17	VDDR3
AE18	IDE_DD(1)
AE19	IDE_DMACKb
AE2	TDATB(3)
AE20	VSS2DI
AE21	A2VDDQ
AE22	A2VSSQ
AE23	A2VSSN
AE24	VDDDI
AE25	AVSSN
AE26	GPIOB(0)
AE27	GPIOB(2)
AE28	GPIOB(1)
AE29	GPIOB(3)
AE3	TDATB(2)
AE4	TSTRTB
AE5	VDDR3
AE6	VDDR3
AE7	VDDR3
AE8	VDDR3
AE9	VDDC
AF1	TDATB(0)
AF10	GPIOA(0)

Table A-2 Pins Sorted by Ball Reference (Cont’d)

Ball Ref	Signal Name
AF11	VIPCLK
AF12	ACDATA_OUT
AF13	ACXTAL_OUT
AF14	USBNB
AF15	IDE_DD(13)
AF16	IDE_DD(11)
AF17	IDE_DD(7)
AF18	IDE_DD(3)
AF19	IDE_IOWb
AF2	TDATB(1)
AF20	RDB
AF21	VDD2DI
AF22	A2VSSN
AF23	AVSSQ
AF24	AVDD
AF25	AVDD
AF26	GPIOB(5)
AF27	GPIOB(9)
AF28	GPIOB(17)
AF29	GPIOB(16)
AF3	TVLDB
AF4	APVDD
AF5	I2SWS_OUTA
AF6	I2CDATAA
AF7	GPIOA(12)
AF8	GPIOA(10)
AF9	GPIOA(4)
AG1	TERB
AG10	TESTEN
AG11	VIPHCTL
AG12	ACDATA_IN(1)
AG13	USBPB
AG14	IDE_DA(1)
AG15	IDE_CSb(0)
AG16	IDE_DD(12)
AG17	IDE_DD(8)
AG18	IDE_DD(4)

Table A-2 Pins Sorted by Ball Reference (Cont'd)

Ball Ref	Signal Name
AG19	IDE_IORb
AG2	VCXO_INA
AG20	IDE_INTRQ
AG21	PMIVSS
AG22	A2VDD
AG23	A2VDD
AG24	AVDDQ
AG25	DAC3
AG26	GPIOB(4)
AG27	GPIOB(13)
AG28	GPIOB(7)
AG29	GPIOB(6)
AG3	PWMA
AG4	APVSS
AG5	I2SSD_OUTA
AG6	I2CCLKB
AG7	GPIOA(11)
AG8	GPIOA(7)
AG9	GPIOA(3)
AH1	VCXO_INB
AH10	VIPHAD(1)
AH11	ACSYNC
AH12	ACDATA_IN(0)
AH13	USBNA
AH14	IDE_DA(2)
AH15	IDE_CSb(1)
AH16	IDE_DD(15)
AH17	IDE_DD(9)
AH18	IDE_DD(5)
AH19	IDE_DD(0)
AH2	XTALIN
AH20	IDE_DMARQ
AH21	PMIVDD
AH22	DAC5
AH23	DAC4
AH24	DAC0
AH25	DAC2

Table A-2 Pins Sorted by Ball Reference (Cont'd)

Ball Ref	Signal Name
AH26	GPIOB(18)
AH27	GPIOB(14)
AH28	GPIOB(10)
AH29	GPIOB(8)
AH3	XTALOUT
AH4	PWMB
AH5	I2SSOSCK_OUTA
AH6	I2CDATAB
AH7	GPIOA(9)
AH8	GPIOA(6)
AH9	GPIOA(2)
AJ1	I2SSCK_IN
AJ10	VIPHAD(0)
AJ11	ACBIT_CLK
AJ12	ACRESETb
AJ13	USBPA
AJ14	USB_OVRCUR
AJ15	IDE_DA(0)
AJ16	IDE_DD(14)
AJ17	IDE_DD(10)
AJ18	IDE_DD(6)
AJ19	IDE_DD(2)
AJ2	I2SWS_IN
AJ20	IDE_IORDY
AJ21	TDB
AJ22	DAC6
AJ23	R2SET
AJ24	DAC1
AJ25	RSET
AJ26	GPIOB(19)
AJ27	GPIOB(15)
AJ28	GPIOB(11)
AJ29	GPIOB(12)
AJ3	I2SSD_IN
AJ4	I2SSCK_OUTA
AJ5	I2CCLKA
AJ6	GPIOA(13)

Table A-2 Pins Sorted by Ball Reference (Cont'd)

Ball Ref	Signal Name
AJ7	GPIOA(8)
AJ8	GPIOA(5)
AJ9	GPIOA(1)
B1	DQMBb(2)
B10	DQB(13)
B11	DQB(11)
B12	DQB(7)
B13	QSB(0)
B14	DQB(1)
B15	VREF
B16	DQA(31)
B17	QSA(3)
B18	DQA(25)
B19	DQA(21)
B2	DQB(19)
B20	DQA(19)
B21	WEAb
B22	CSAb(0)
B23	AA(12)
B24	CLKAb
B25	AA(7)
B26	AA(3)
B27	AA(0)
B28	DQA(13)
B29	DQA(12)
B3	DQB(18)
B4	RASBb
B5	AB(14)
B6	AB(10)
B7	AB(9)
B8	AB(5)
B9	AB(1)
C1	DQB(23)
C10	DQB(14)
C11	DQMBb(1)
C12	DQB(8)
C13	DQB(4)

Table A-2 Pins Sorted by Ball Reference (Cont'd)

Ball Ref	Signal Name
C14	DQB(2)
C15	MEMTEST
C16	DQA(30)
C17	DQMAb(3)
C18	DQA(24)
C19	DQA(20)
C2	DQB(22)
C20	DQA(18)
C21	CASAb
C22	CKEA
C23	AA(11)
C24	CLKA
C25	AA(6)
C26	AA(2)
C27	DQMAb(1)
C28	QSA(1)
C29	DQA(11)
C3	QSB(2)
C4	CASBb
C5	CKEB
C6	AB(11)
C7	CLKB
C8	AB(6)
C9	AB(2)
D1	DQB(25)
D10	DQB(15)
D11	QSB(1)
D12	DQB(9)
D13	DQB(5)
D14	DQB(3)
D15	MEMVMODE
D16	DQA(29)
D17	DQA(27)
D18	DQA(23)
D19	QSA(2)
D2	DQB(24)
D20	DQA(17)

Table A-2 Pins Sorted by Ball Reference (Cont'd)

Ball Ref	Signal Name
D21	RASAb
D22	AA(14)
D23	AA(10)
D24	AA(9)
D25	AA(5)
D26	DQA(10)
D27	DQA(9)
D28	DQA(8)
D29	DQA(7)
D3	DQB(21)
D4	DQB(20)
D5	CSBb(0)
D6	AB(12)
D7	CLKBb
D8	AB(7)
D9	AB(3)
E1	QSB(3)
E10	VDDC
E11	VDDR1
E12	VDDR1
E13	VDDC
E14	VDDC
E15	VDDC18_RIGHT
E16	VDDC
E17	VDDC
E18	VDDR1
E19	VDDR1
E2	DQMBb(3)
E20	VDDC
E21	VDDC
E22	VDDR1
E23	VDDR1
E24	VDDR1
E25	VDDR1
E26	DQA(6)
E27	DQA(5)
E28	DQA(4)

Table A-2 Pins Sorted by Ball Reference (Cont'd)

Ball Ref	Signal Name
E29	QSA(0)
E3	DQB(27)
E4	DQB(26)
E5	VDDR1
E6	VDDR1
E7	VDDR1
E8	VDDR1
E9	VDDC
F1	DQB(31)
F2	DQB(30)
F25	VDDR1
F26	DQMAb(0)
F27	DQA(3)
F28	DQA(2)
F29	DQA(1)
F3	DQB(28)
F4	DQB(29)
F5	VDDR1
G1	SMDATAA
G2	SMDETECTA
G25	VDDR1
G26	DQA(0)
G27	AD(0)
G28	AD(1)
G29	AD(2)
G3	SMRSTA
G4	SMCLKA
G5	VDDR1
H1	SMVCCB
H2	SMRSTB
H25	VDDP
H26	AD(3)
H27	AD(4)
H28	AD(5)
H29	AD(6)
H3	SMCLKB
H4	SMVCCA

Table A-2 Pins Sorted by Ball Reference (Cont’d)

Ball Ref	Signal Name
H5	VDDR3
J1	SMRSTS
J2	SMCLKS
J25	VDDP
J26	AD(7)
J27	CBEb(0)
J28	AD(8)
J29	AD(9)
J3	SMDATAB
J4	SMDETECTB
J5	VDDR3
K1	IRRXD
K2	SMDATAS
K25	VDDC
K26	AD(10)
K27	AD(11)
K28	AD(12)
K29	AD(13)
K3	SMDETECTS
K4	SMVCCS
K5	VDDC
L1	FAD(15)
L11	VSS
L12	VSS
L13	VSS
L14	VSS
L15	VSS
L16	VSS
L17	VSS
L18	VSS
L19	VSS
L2	FGNTb
L25	VDDC
L26	AD(14)
L27	AD(15)
L28	CBEb(1)
L29	PAR

Table A-2 Pins Sorted by Ball Reference (Cont’d)

Ball Ref	Signal Name
L3	FREQb
L4	IRTXD
L5	VDDC
M1	FAD(12)
M11	VSS
M12	VSS
M13	VSS
M14	VSS
M15	VSS
M16	VSS
M17	VSS
M18	VSS
M19	VSS
M2	FAD(13)
M25	VDDP
M26	SERRb
M27	PERRb
M28	STOPb
M29	DEVSELb
M3	FAD(14)
M4	FCLK
M5	VDDR3
N1	FAD(8)
N11	VSS
N12	VSS
N13	VSS
N14	VSS
N15	VSS
N16	VSS
N17	VSS
N18	VSS
N19	VSS
N2	FAD(9)
N25	VDDP
N26	TRDYb
N27	IRDYb
N28	FRAMEb

Table A-2 Pins Sorted by Ball Reference (Cont'd)

Ball Ref	Signal Name
N29	CBEb(2)
N3	FAD(11)
N4	FAD(10)
N5	VDDR3
P1	FAD(4)
P11	VSS
P12	VSS
P13	VSS
P14	VSS
P15	VSS
P16	VSS
P17	VSS
P18	VSS
P19	VSS
P2	FAD(5)
P25	VDDC18_BOTTOM
P26	AD(16)
P27	AD(17)
P28	AD(18)
P29	AD(19)
P3	FAD(6)
P4	FAD(7)
P5	VDDC18_TOP
R1	FAD(3)
R11	VSS
R12	VSS
R13	VSS
R14	VSS
R15	VSS
R16	VSS
R17	VSS
R18	VSS
R19	VSS
R2	FAD(2)
R25	VDDC
R26	CBEb(3)
R27	AD(22)

Table A-2 Pins Sorted by Ball Reference (Cont'd)

Ball Ref	Signal Name
R28	AD(21)
R29	AD(20)
R3	FAD(1)
R4	FBE(1)
R5	VDDC
T1	FAD(0)
T11	VSS
T12	VSS
T13	VSS
T14	VSS
T15	VSS
T16	VSS
T17	VSS
T18	VSS
T19	VSS
T2	FBE(0)
T25	VDDC
T26	AD(25)
T27	AD(24)
T28	AD(23)
T29	IDSEL
T3	FALE(1)
T4	FALE(0)
T5	VDDC
U1	FSTB
U11	VSS
U12	VSS
U13	VSS
U14	VSS
U15	VSS
U16	VSS
U17	VSS
U18	VSS
U19	VSS
U2	FRDYb
U25	VDDC
U26	AD(28)

Table A-2 Pins Sorted by Ball Reference (Cont'd)

Ball Ref	Signal Name
U27	AD(29)
U28	AD(27)
U29	AD(26)
U3	FWE
U4	FRD
U5	VDDC
V1	FCEb(5)
V11	VSS
V12	VSS
V13	VSS
V14	VSS
V15	VSS
V16	VSS
V17	VSS
V18	VSS
V19	VSS
V2	FCEb(4)
V25	VDDP
V26	REQb(1)
V27	REQb(0)
V28	AD(31)
V29	AD(30)
V3	FCEb(3)
V4	FCEb(1)
V5	VDDR3
W1	FCEb(2)
W11	VSS
W12	VSS
W13	VSS
W14	VSS
W15	VSS
W16	VSS
W17	VSS
W18	VSS
W19	VSS
W2	FCEb(0)
W25	VDDP

Table A-2 Pins Sorted by Ball Reference (Cont'd)

Ball Ref	Signal Name
W26	RESETOUTb
W27	GNTb(1)
W28	GNTb(0)
W29	REQb(2)
W3	NRSSCLKA
W4	NRSSDATA_INA
W5	VDDR3
Y1	NRSSDATA_OUTA
Y2	TCLKA
Y25	VDDC
Y26	GNTb(2)
Y27	PCICLKOUT(2)
Y28	PCICLKOUT(1)
Y29	PCICLKOUT(0)
Y3	TDATA(7)
Y4	TDATA(6)
Y5	VDDC

This page intentionally left blank.

Appendix B

Revision History

Rev 1.7 (March 2001)

- Preliminary release

Rev 1.8 (May 2001)

(Based on eng spec 3.27)

- General edits
- Updated Chapter 3: Pinout and Strap Descriptions
- Added Chapter 20 on Time Shifting
- Added Chapter 21: Electrical and Physical Characteristics
- Updated extensively Chapter 9 and 15

Rev 1.9 (May 2001)

(Based on eng spec 3.28)

- New ball out diagram and changes in signal names in Chapter 3
- New pin listing in Appendix A.
- New mechanical drawing in Chapter 21

Rev 2.0 (June 2001)

(Based on padout spec 1.7)

- New ball out diagram in Chapter 3 with the following ball assignment swaps:

Signal Name	New Ball Assignment	Previous Ball Assignment
INTRb(0)	AB27	AA26
INTROUTb	AA26	AB27
AD(23)	T28	R26
CBEb (3)	R26	T28
AD(29)	U27	U26
AD(28)	U26	U27
RESETOUTb	W26	AA28
GNTb(2)	Y26	W26
PCICLKOUT(3)	AA28	Y26
TVLDA	AC1	AB4
TERA	AC2	AC1
NRSSCLKB	AB4	AC2
TDATB(1)	AF2	AE4
TSTRTB	AE4	AF2

Signal Name	New Ball Assignment	Previous Ball Assignment
TERB	AG1	AF4
VCX0_INA	AG2	AG1
PWMA	AG3	AG2
APVDD	AF4	AG3
DQb(21)	D3	C1
DQb(23)	C1	D3
DQb(28)	F3	F4
DQb(29)	F4	F3
DQb(22)	C2	D4
DQb(20)	D4	C2
SMDETECTA	G2	G1
SMVCCA	H4	G2
SMDATAA	G1	H4
FAD(10)	N4	N3
FAD(11)	N3	N4
FRD	U4	U3
FWE	U3	U4
FCEb(2)	W1	V4
FCEb(1)	V4	W1
IDE_DA(1)	AG14	AJ14
USBNB	AF14	AG14
USBPB	AG13	AF14
USBPA	AJ13	AG13
USB_OVRCUR	AJ14	AJ13
IDE_DD(2)	AJ19	AE18
IDE_DD(1)	AE18	AJ19
IDE_IORDY	AJ20	AE19
IDE_DMARQ	AH20	AJ20
IDE_DMACKb	AE19	AH20
GPIOB(5)	AF26	AG26
GPIOB(4)	AG26	AF26
IDE_DD(13)	AF15	AH16
IDE_DD(15)	AH16	AF15
GPIOA(10)	AF8	AH7
GPIOA(8)	AJ7	AF8
GPIOA(9)	AH7	AJ7
VIPHAD(0)	AJ10	AF11
VIPHAD(1)	AH10	AJ10
VIPCLK	AF11	AH10
GPIOB(12)	AJ29	AF27
GPIOB(9)	AF27	AJ29
VDD2DI	AF21	AE21
A2VDDQ	AE21	AF21
COMP	AG25	AH25
B	AH25	AG25
Y_G	AH22	AJ22
COMP_B	AJ22	AH22
MEMTEST	C15	B15
VREF	B15	C15

- New pin listing in Appendix A.

Rev 2.1 (June 2001)

(Based on padout spec 1.8)

- 4 ball assignment changes:

Signal Name	New Ball Assignment	Previous Ball Assignment
FCLK	M4	L1
FAD(15)	L1	M4
FAD(0)	T1	R4
FBE(1)	R4	T1

Rev 2.2 (July 2001)

- Added new standards in the VBI Block section 9.4.2.

Rev 2.3 (July 2001)

- Added Chapter 22: Timing Specifications

Rev 2.4 (August 2001)

- Updated Chapter 22: Timing Specifications
- Updated Chapter 5 and 6.
- Added Chapter 23: EXOR Tree.

Rev 3.0 (September 2001)

- Major content re-organization and updates.

Rev 3.1 (October 2001)

- Updated UIRT section.
- Added DMA section.
- Added I2C and VIP information

Rev 3.2 (October 2001)

- Updated the I^2C and VIP sections.
- Updated the FlexBus Controller section.
- Updated the Timing chapter.
- Updated the Memory Controller chapter.
- Added LPC Interface Controller section.
- Updated the EIDE section.
- Updated the USB section.

Rev 3.3 (November 2001)

- Updated MIPS chapter.

- General edits.

Rev 3.4 (November 2001)

- Updated Transport Demultiplexer section.
- General edits

Rev 3.5 (December 2001)

- Updated Transport Demultiplexer section.
- General edits based on engineering document rev 3.32

Rev 3.6 (January 2002)

- Updated Transport Demultiplexer section

Rev 3.7 (Feb 2002)

- General edits.
- Added internal Pullup/down resistor mapping table in Appendix A.

Rev 3.8 (Feb 2002)

- Changed PWMA/AUDIO_CLKOUTA and PWMB/AUDIO_CLKOUTB pins from type I/O to type O only.

Rev 3.9 (Mar 2002)

- Updated Section 3.3 with many new sub-sections.
- Combined sections 9.3.7 and 9.3.8 and added DAC Configuration tables.
- Modified Straps Table (Section 9.4).
- Moved Default Internal Pull-up/down Table to Chapter 9 and fixed 3 errors (Section 9.5).
- Added Soldering/Reflow Profile information (Chapter 10).
- Added AC/DC information (Chapter 10)
- Added Power Sequence section (Chapter 10).
- Added Power Dissipation section (Chapter 10).
- Added Input/Output Timing section (Section 11.3.6).
- General edits.