GT–64240A

System Controller for MIPS Processors

FEATURES

Integrated system controller with PCI interface and communication ports for high-performance embedded control applications.

Supports 64-bit bus MIPS CPUs:

- PMC-Sierra RM5260, RM5270, RM7000A and RM7000C
- IDT RC5000 and RC64575
- NEC Rv5000 and Rv5464
- LSI Viper

CPU interface features:

- Multiplexed 64-bit address/data bus (36-bit address, 64-bit data).
- Up to 133MHz CPU bus frequency.
- 2.5V or 3.3V CPU bus interface.
- 256 byte write posting buffer that accepts up to six CPU cache line writes with zero wait-states
- 64 byte CPU read buffer that accepts up to two cache line CPU reads.
- Supports RM700A split read transactions (two outstanding reads) with out-of-order completion.
- Supports R4000 and pipeline write modes (also available in multiple GT–64240A configuration).
- Supports R5000/R7000 CPU caches.

CPU address remapping to PCI.

Supports access, write, and caching protection to configurable address ranges.

Supports up to four multiple GT–64240A devices on the same CPU bus.

Supports both Little and Big Endian modes.

Synchronization barrier support between the CPU and the PCI.

SDRAM controller:

- 64-bit wide (+ 8-bit ECC) SDRAM interface.
- Up to 133MHz SDRAM frequency.
- 3.3V SDRAM interface.
- Supports SDRAM and registered SDRAM.
- Four DRAM banks.
- 1MB-1GB bank address space.
- Up to 4GB DRAM address space.
- Supports 2-way & 4-way SDRAM bank interleaving.
- Supports 16/64/128/256/512 Mbit SDRAM.
- Supports up to 16 pages open.

Supports the Unified Memory Architecture Standard.

- Allows for external masters access to SDRAM directly.
- Allows glueless multiple GT–64240A devices share the same SDRAM.

Device controller:

- A dedicated 32-bit multiplexed address/data bus (separated from SDRAM bus).
- Up to 133MHz bus frequency.
- 3.3V device interface.
- Five chip selects.
- 1MB-512MB bank address space.
- Up to 2.5GB Device address space.
- Programmable timing for each chip select.
- Supports many types of standard memory and I/O devices.
- Optional external wait-state support.
- 8-,16-,32-bit width device support.
- Support for boot ROMs.

Eight channels DMA controller:

- Chaining via linked-lists of records.
- Byte address boundary for source and destination.
- Moves data between the PCI, SDRAM, Devices, and CPU buses.
- Two 2Kbyte internal FIFOs allowing transfers to take place concurrently.
- Alignment of source and destination addresses.
- Increment or hold of source and destination addresses.
- DMAs can be initiated by the CPU, external DMAReq* signal, or an internal timer/counter.
- Termination of DMA transfer on each channel.
- Descriptor ownership transfer to CPU.
- Supports unlimited burst DMA transfers between the SDRAM and the PCI.

FEATURES (CONTINUED)

Two high-performance PCI 2.2 compliant interfaces.

- P2P memory, I/O, and configuration transactions between the two PCI interfaces.
- Separate reset for each PCI interface.
- PCI bus speed of up to 66MHz with zero wait states.
- The two PCI interfaces can run in asynchronous clocks to each other.
- Operates either synchronous or asynchronous to CPU clock, at slower, equal, or faster clock frequency.
- 32/64-bit PCI master and target operations.
- Supports flexible byte swapping through the PCI interface.
- 3.3V PCI buffers (configurable 3.3/5V).
- Configurable PCI arbiter for up to six external masters, plus the internal master, on each PCI interface.

Master specific features:

- 512 bytes posted write buffer and 512 bytes read buffer for unlimited DMA bursts between SDRAM and the PCI.
- Host to PCI bridge - translates CPU cycles to PCI I/O or Memory cycles.
- Supports 64-bit addressing through Dual Address cycles.
- Supports configuration, interrupt acknowledge, and special cycles on the PCI bus.

Target specific features:

- PCI to main memory bridge.
- 512 bytes posted write buffer and 1Kbyte read prefetch buffer for unlimited bursts between the PCI and SDRAM.
- Up to eight delayed reads.
- Read prefetch of up to 1Kbyte.
- Supports fast back-to-back transactions.
- Supports memory and I/O transactions to internal configuration registers.
- Supports 64-bit addressing through dual address cycles.
- Synchronization barrier support between the PCI and the CPU.
- PCI address remapping to resources.
- Supports access and write protect to configurable address ranges.

PCI Hot-Plug and CompactPCI Hot-Swap ready compliant.

Messaging Unit:

- Efficient messaging interface between the PCI and the CPU, or between the two PCI interfaces.
- Doorbell and message interrupts between the CPU and the PCI.
- I_2O support.

Plug and Play Support:

- Plug and Play compatible configuration registers.
- PCI configuration registers can be accessed from the CPU or PCI side.
- Expansion ROM support.
- VPD support.
- PCI Power Management compliant.
- Message signal interrupt support.
- BIST support.

Three 10/100Mbps Fast Ethernet MAC controllers:

- MII or RMII interface.
- Full duplex and flow-control support.
- Programmable perfect filtering of 8K MAC addresses (both physical and multicast).
- Priority queueing based on MAC address or 802.1q tag (four queues for receive, two for transmit).
- IGMP/BPDU packet trapping.

Two Multi-Protocol Serial Controllers (MPSCs):

- Each channel supports HDLC, BISYNC, UART, and Transparent protocols.
- Bit rate of up to 55Mbit/s on multiple channels, simultaneously.
- Dedicated DPLL for clock recovery and data encoding/decoding.
- Supports NRZ, NRZI, FM0, FM1, Manchester, and Differential Manchester.
- Hardware support for HDLC over asynchronous channel in UART mode.

Ten Serial DMA channels (SDMA) supporting the MPSCs and Ethernet controllers.

- Moves data between communications controllers and the SDRAM, Device, or PCI buses.
- Chaining via a linked list of descriptors.

Two baud rate generators with multiple clock sources.

32 multi purpose pins (MPP) dedicated for peripheral functions and general purpose I/Os (GPP).

- Each pin can be configured independently.
- GPP inputs can generate a maskable interrupt.

FEATURES (CONTINUED)

Data integrity support between the CPU, PCI, and DRAM interfaces:

- ECC support on SDRAM interface.
- Parity support on the CPU and PCI busses.
- Propagation of parity and ECC errors between the three interfaces.
- Full error report, including error counter.
- Support corruption of ECC bank for debug.

Interrupt controller:

- Maskable interrupts to CPU and PCI.
- Drive up to seven interrupt pins.

Eight 32-bit wide timer/counters initiated by the CPU or externally through the MPP pin.

I^2C interface that supports master and slave operations.

Serial ROM initialization through I^2C interface.

Advanced 0.18 micron process.

665 PBGA package

Document Conventions

The following name and usage conventions are used in this document:

Signal Range	A signal name followed by a range enclosed in brackets represents a range of logically related signals. The first number in the range indicates the most significant bit (MSb) and the last number indicates the least significant bit (LSb). Example: GTXD[7:0]
Active Low Signals *	A * symbol at the end of a signal name indicates that the signal's active state occurs when voltage is low. Example: INT*

Document Status

Advanced Information	This datasheet contains design specifications for initial product development. Specifications may change without notice. Contact Marvell Field Application Engineers for more information.
Preliminary Information	This datasheet contains preliminary data, and a revision of this document will be published at a later date. Specifications may change without notice. Contact Marvell Field Application Engineers for more information.
Final Information	This datasheet contains specifications on a product that is in final release. Specifications may change without notice. Contact Marvell Field Application Engineers for more information.

Galileo / Marvell
2350 Zanker Road
San Jose, CA 95131
Phone: 1 408 367-1400
Fax: 1 (408) 367-1401
www.galileoT.com

www.marvell.com

Document Status: Preliminary

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

List of Tables

List of Tables (Continued)

List of Tables (Continued)

List of Tables (Continued)

List of Tables (Continued)

List of Figures

List of Figures (Continued)

List of Figures (Continued)

List of Figures (Continued)

1. OVERVIEW

The GT–64240A provides a single-chip solution for designers building systems for a MIPS 64-bit bus CPU. The GT–64240A architecture supports several system implementations for different applications.

The GT–64240A has a five bus architecture:

- A 64-bit interface to the CPU bus.
- A 64-bit interface to SDRAM.
- A 32-bit interface to Devices.
- Two 64-bit PCI interfaces.

The five buses are de-coupled from each other in most accesses, enabling concurrent operation of the CPU bus, PCI devices, and accesses to memory. For example, the CPU bus can write to the on-chip write buffer, a DMA engine can move data from SDRAM to its own buffers, and a PCI device can write into an on-chip FIFO, all simultaneously.

In addition, the GT–64240A integrates three 10/100Mbps ethernet ports and two MPSC controllers. Each MPSC can be programmed to process either HDLC, UART, BISYNC, or Transparent protocols.

The GT–64240A offers 10 SDMA channels to support the two MPSCs and three Fast Ethernet controllers. The SDMA channels are used to transfer data from the various serial ports to/from the SDRAM, Device, or PCI buses. The SDMA uses linked chain of descriptors and buffers to reduce CPU overhead.

1.1 CPU Bus Interface

The GT–64240A supports MIPS bus protocol. With a maximum frequency of 133MHz, the CPU can transfer in excess of 1 Gbytes/sec.

NOTE: The PMC-Sierra RM7000C CPU is now supported in TTL mode, only.

The GT–64240A supports up to two pipelined transactions on the CPU bus. For example, if the CPU initiates a data read from the PCI interface and starts a code read from SDRAM, the two cycles are pipelined. The CPU interface reads from the PCI interface and from SDRAM in parallel.

By the time read data is returned from the PCI interface, read data from SDRAM is already available – since an SDRAM access is faster than a PCI access. The GT–64240A drives the data of the SDRAM read immediately after a PCI read data with zero wait states. In case of a RM7000 CPU, that supports out of order read completion, the GT–64240A drives the SDRAM read data first and then the PCI read data that arrives later.

The CPU can connect with up to four GT–64240A or any other 60x compatible slave devices. This increases the flexibility of system design significantly.

NOTE: The increased loading has a small effect on the system's maximum operating frequency.

The GT–64240A supports CPU address remapping to the PCI interface. It also supports access, write, and caching protection, per user specified address ranges.

The GT–64240A CPU interface supports both Little and Big Endian modes.

NOTE: For additional information about the CPU bus interface, see Section 4. "CPU Interface" on page 56.

1.2 SDRAM Interface

The GT–64240A SDRAM controller supports SDRAM and registered SDRAM. It supports 16/64/128/256/512 Mbit SDRAMs.

The GT–64240A works at frequencies up to 133MHz, and can address up to 4GBytes.

Up to four banks of SDRAM may be connected to The GT–64240A.

The controller supports two bank interleaving for 16 Mbit SDRAMs and four bank interleaving for 64/128/256/512 Mbit SDRAMs.

The GT–64240A also supports page mode, which minimizes SDRAM cycles on multiple transactions to the same SDRAM page, and can be configured to support up to 16 simultaneously opened pages.

The GT–64240A supports the Unified Memory Architecture (UMA) protocol that enables external masters to arbitrate for direct access to SDRAM. This feature enhances system performance and gives flexibility when designing shared memory systems.

NOTE: For additional information about the SDRAM interface, see Section 5. “SDRAM Controller” on page 102.

1.3 Device Interface

The GT–64240A device controller supports different types of memory and I/O devices.

It has the control signals and the timing programmability to support devices such as SynBurst SRAM, Flash, EPROMs, FIFOs, and I/O controllers. Device widths of 8-, 16-, and 32-bits are supported.

The GT–64240A has a dedicated 32-bit Device bus. It supports bursts of up to 32 bytes to a 32-bit wide device and can run SDRAM and Device transactions simultaneously, so SDRAM access performance is not affected by access to slow memory devices.

NOTE: For additional information about the Device interface, see Section 7. “Device Controller” on page 140.

1.4 PCI Interface

The GT–64240A interfaces directly with two 64-bit PCI busses, operating at a maximum frequency of 66MHz. Each PCI interface can act as a master initiating a PCI bus transaction or as a target responding to a PCI bus transaction.

The GT–64240A becomes the PCI bus master when the CPU, DMA, or Comm port initiates a bus cycle to a PCI device. It’s internal buffers allow unlimited DMA bursts between PCI and memory. It supports all PCI commands including 64-bit addressing using DAC cycles.

The GT–64240A acts as a target when a PCI device initiates a memory access (or an I/O access in the case of internal registers or a P2P transaction). It responds to all memory read/write accesses, including DAC, and to all configuration and I/O cycles, in the case of internal registers. It’s internal buffers allow unlimited burst reads and writes. It supports up to eight pending delayed reads.

The GT–64240A can also perform basic P2P transfers. Each PCI interface can directly transfer memory and I/O transactions to the other PCI interface. Also, type 1 configuration transactions can be transferred to the other PCI interface as type 1 or type 0 configuration cycle.

Each PCI interface can run at different clock speeds and there are no restrictions between the PCI and CPU clock ratios. It is possible for the PCI clock speed to be slower, equal, or faster than the CPU clock. It is also optional to synchronize the PCI clock to the CPU clock.

It is possible to program the PCI slave to retry all PCI transactions targeted to the GT–64240A, during CPU initialization.

The PCI slave performs PCI address remapping to SDRAM and Devices. It also supports configurable read prefetch, access and write protect, and byte swapping per user specified address ranges.

The GT–64240A PCI interface is fully PCI rev. 2.2 compliant. It contains all the required PCI configuration registers. All internal registers, including the PCI configuration registers, are accessible from the CPU bus or the PCI bus.

The GT–64240A configuration register set is PC Plug and Play compatible. It supports PCI spec rev. 2.2 features such as VPD, message signal interrupt, and power management.

The GT–64240A also supports PCI Hot-Plug and CompactPCI Hot-Swap ready.

The GT–64240A also includes a messaging unit to support industry standard I_2O messaging. This includes:

- Two doorbell registers.
- Two message registers.
- Four messages queues located in SDRAM.

NOTE: For additional information about the PCI interface, see Section 8. “PCI Interface” on page 155.

1.5 DMA Engines

The GT–64240A incorporates eight high performance DMA engines. Each DMA engine has the capability to transfer data between PCI devices, SDRAM, or devices.

The DMA uses two internal 2Kbyte FIFOs for temporary DMA data storage. Two FIFOs allows two DMA channels to work concurrently since each channel utilizes a FIFO. For example, channel0 transfers data from SDRAM to PCI_0 using one FIFO, while channel2 transfers data from PCI_1 to device using the other FIFO.

Source and destination addresses can be non-aligned on any byte address boundary. The DMA channels are programmable by the CPU, or by PCI masters, or without CPU bus intervention via a linked list of descriptors. This linked list is loaded by the DMA controller into the channel’s working set when a DMA transaction ends. The DMA supports increment/hold on source and destination addresses independently, and alignment of addresses towards source and destination. In addition, the GT–64240A provides an override capability of source/destination/descriptor address mapping to force access to PCI_0 or PCI_1 bus.

It is possible to initiate a DMA transfer by the software writing to a register, an external request via a DMAReq* pin, or an internal timer/counter. Eight End of Transfer pins act as inputs to the GT–64240A and allow ending a DMA transfer on a certain channel. In cases of chained mode with the transfer completed, it is possible to transfer the descriptor to CPU ownership. The CPU can calculate the number of remaining bytes in the buffer associated with the closed descriptor.

NOTE: For additional information about the DMA engines, see Section 10. “IDMA Controller” on page 276.

1.6 Data Integrity

The GT–64240A supports full data integrity on its different interfaces.

The GT–64240A supports ECC on SDRAM. It supports detection and correction of one error, detection of two errors, and detection of three and four errors, if they are in the same nibble. It supports SDRAM read-modify-write for partial writes. It has full error report, including ECC error counter. It also supports corruption of ECC bank for debug.

The GT–64240A supports parity checking and generation on the PCI bus through PAR and PERR* signals. It also supports configured SERR* assertion for different errors. In cases of error detection, address and data are latched for debug.

The GT–64240A also supports data parity checking and generation on the CPU bus. In case of error detection, an interrupt is asserted. As with error detection on the PCI bus, address and data are latched for debug.

ECC and parity errors are optionally propagated between the interfaces. For example, in case of a PCI read from SDRAM that results in detection of uncorrectable ECC error, the GT–64240A may drive the wrong PAR value with the read data on the PCI bus.

NOTE: For additional information about data integrity features, see Section 6. “Address and Data Integrity” on page 133.

1.7 Communication Ports

The GT–64240A integrates a high-performance communication unit. This unit includes two multi-protocol serial controllers (MPSCs) and three perfect filtering 10/100 Ethernet controllers, and 10 SDMA engines.

The GT–64240A can directly support many WAN interfaces including Basic Rate ISDN, frame relay, non channelized T1/E1/T3, xDSL (HDSL, ADSL, VDSL etc.), HSSI and more.

The two MPSCs integrated on the GT–64240A support UART, HDLC, BISYNC, and transparent protocols. The MPSCs are implemented in hardware. This implementation allows for superior performance versus microcoded implementations.

In HDLC mode, the MPSCs perform all framing operations, such as; bit stuffing/stripping, flag generation, and part of the data link operations (e.g. address recognition functions). The MPSCs directly support common HDLC protocols including those used by ISDN and frame relay.

There are three 10/100Mbps full duplex Ethernet ports in the GT–64240A. Each port is fully compliant with the IEEE 802.3 and 802.3u standards and integrates MAC function and a dual speed MII interface.

The Ethernet ports can be configured to MII or RMII (three Ethernet ports configuration is available only with RMII). The port’s speed (10 or 100Mb/s) as well as the duplex mode (half or full duplex) is auto negotiated through the PHY and does not require user intervention.

The ports’ logic also supports 802.3x flow-control mode for full-duplex and back-pressure mode for half-duplex.

The GT–64240A's Ethernet ports include Galileo Technology's advanced address filtering capability and can be programmed to accept or reject packets based on MAC addresses, thus providing hardware acceleration to complicated tasks such as bridging, routing, and firewall. Up to 8K individual MAC addresses can be filtered.

NOTE: For additional information about the communication ports, see Section 12. "Communication Unit" on page 321.

2. PIN INFORMATION

Figure 1 shows the GT–64240A interfaces.

Figure 1: GT–64240A Interfaces

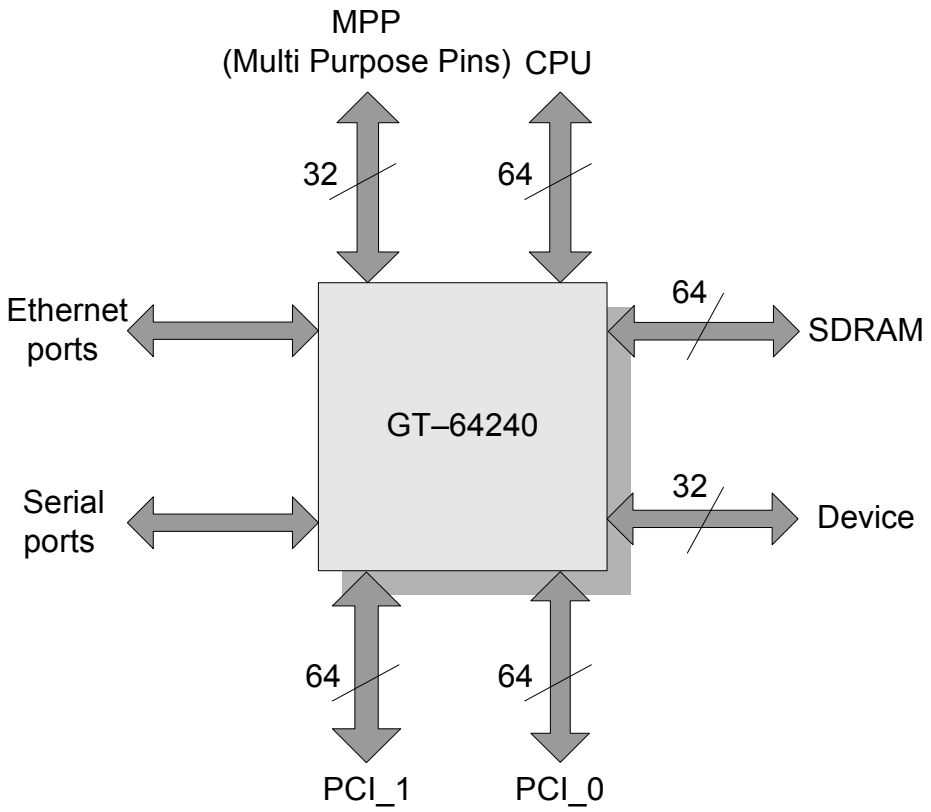


Table 1 lists the conventions that apply to I/O or O type pins described in the Pin Assignment tables:

Table 1: Pin Assignment Table Conventions

Abbreviation	Description
t/s	Tri-State pin.
s/t/s	Sustained Tri-State pin. Driven to its inactive value for one cycle before float. **NOTE:** A pull-up is required to sustain the inactive value.
o/d	Open Drain pin. Allows multiple drivers simultaneously (wire-OR connection). **NOTE:** A pull-up is required to sustain the inactive value.

Table 2: Core Clock Pin Assignments

Pin Name/ Ball #	Type	Full Name	Description
TClk F24	I	Internal Clock	The GT–64240A units internal clock (up to 133MHz). Used as input clock to the internal PLL.
AVCC H25	I	PLL Vcc	Quiet power supply to the internal PLL. **NOTE:** For information on the PLL, see Section 26.4 "PLL Power Filter Circuit" on page 535.
AGND G25	I	PLL Vss	Quiet ground supply to the internal PLL.
Core Clock Pin Count: 3			

Table 3: CPU Interface Pin Assignments

Pin Name/ Ball #	Type	Full Name	Description
SysClk E25	I	System Clock	CPU interface clock (up to 100 MHz). Can run at any frequency less than or equal to the TClk frequency asynchronously. The CPU interface can be configured to run with TClk instead of SysClk. **NOTE:** In this configuration, SysClk is not used and must be tied to GND.
SysRst* D25	I	System Reset	Main reset signal of the GT–64240A. Resets all units to their initial state. **NOTE:** When in the reset state, all output pins, except for SDRAM address and control signals, are put into tristate.
SysAD[63:0]	t/s I/O	System Address/Data Bus	64-bit multiplexed CPU address/data bus. Driven by the CPU during address phase and write data phase. Driven by GT–64240A during read response data phase. **NOTE:** These pins utilize integrated pullups.
[63:54] AC27, AC28, U27, AC29, V30, V29, L29, L30, Y31, P30 [53:44] AD26, P29, AC30, R30, AD27, AA30, AA31, V31, AA28, U28 [43:34] W27, R31, W28, AD28, W29, AD29, P31, AB27, AB28, AD30 [33:24] W30, T26, M26, R27, N28, N27, T27, L31, U30, M31 [23:14] Y27, M30, P28, Y29, V27, P27, T30, AC26, V26, AA27 [13:0] AC31, T31, V28, Y30, AA29, U31, R29, AB31, Y28, N31, T28, AB29, W31, AD31			

Table 3: CPU Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
SysADC[7:0] L28, R28, M28, AB30, M29, N30, U29, T29	t/s I/O	System Address/Data Parity Bus	8-bit parity for the SysAD bus. Driven by CPU during write data phase. Driven by GT–64240A during read response data phase. **NOTE:** – SysADC is valid on data cycles only. – These pins utilize integrated pullups.
SysCmd[8:0] G26, G27, G29, F28, F29, F30, F31, F26, E31	t/s I/O	System Command/Data Identifier Bus	9-bit multiplexed CPU command/data identifier bus. System Command driven by the CPU during address phase. Data identifier driven by the CPU during write data phase and by the GT–64240A during read response data phase.
ValidOut* D31	I	Valid Output	The CPU signals that it is driving valid address/data on the SysAD bus and valid command/data identifier on the SysCmd bus.
Release* E27	I	Release	The CPU signals that it has released the SysAD and the SysCmd buses after completion of a read request.
ValidIn* C31	t/s O	Valid Input	The GT–64240A signals that it is driving valid read data on the SysAD bus and a valid data identifier on the SysCmd bus. **NOTE:** In multi-GT mode, acts as s/t/s pin.
SysRdyOut* A29	t/s O	Read/Write Ready Output	The GT–64240A signals that it can accept a CPU read or write request. **NOTE:** Must be connected to both RdRdy* and WrRdy* CPU input pins. In multi-GT–64240A configurations, SysRdyOut*, of the boot GT–64240A, is connected to the CPU RdRdy* and WrRdy* inputs. The SysRdyOut* outputs, of the other GT–64240A devices, are connected to SysRdyIn[2:0] of the boot GT–64240A device.
SysRdyIn[2:0]* H27, H30, H31	I	Read/Write Ready Input	**NOTE:** Relevant only in multi-GT–64240A configurations. In a single GT configuration, connect to GND. SysRdyIn of the boot GT–64240A device is connected to SysRdyOut of all other GT–64240A devices. • SysRdyIn[0]* of all the GT–64240A devices, except for the boot device, are connected to the boot GT–64240A device's SysRdyOut* signal, which is also the CPU RdRdy* and WrRdy* input. • SysRdyIn[2:1]* of all the GT–64240A devices, except of the boot device, are not used.

Table 3: CPU Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
PReq* A26	I	Processor Request	CPU requests from the GT–64240A for SysAD bus mastership. **NOTE:** If interfacing with a CPU that does not support Preq*, pull up is required on this pin.
PAck* C25	t/s O	Processor Acknowledge	The GT–64240A signals that it releases the SysAD bus in response to PReq*. **NOTE:** In multi-GT mode, acts as s/t/s pin.
RspSwap* E26	t/s O	Read Response Swap	The GT–64240A signals that it is returning read data to the CPU out of order. **NOTE:** In a multi-GT configuration, RspSwap* is NC.
CPUInt* C25	t/s O	Interrupt	Level sensitive interrupt driven by the GT–64240A to the CPU. There are four more optional CPU interrupt pins multiplexed on the GPP pins.
TcTCE* G30	I	Ternary Cache Tag RAM Chip Enable	The CPU L3 cache controller signals that it is accessing L3 cache. **NOTE:** If interfacing with a CPU that does not support-TcTCE*, pull up is required on this pin.
TcDOE* H26	t/s O	Ternary Cache Data RAM Output Enable	In case of a cache hit, the GT–64240A enables L3 data RAM drive read data on SysAD. **NOTE:** In a multi-GT configuration, acts as s/t/s pin.
TcWord[1:0] E28, E29	t/s O	Ternary Cache Word Index	Determines correct L3 double-word index. Driven by the GT–64240A in case of a CPU block read miss (driven by CPU L3 cache controller in case of L3 hit). **NOTE:** A pull-down is required .
TcMatch C29	I	Ternary Cache Tag Match	Asserted by tag RAM on L3 cache tag match. **NOTE:** If there is no L3 cache, or when working in SimpleCache mode, connect the TcMatch input to GND.
CPU Interface Pin Count: 97			

Table 4: PCI Bus 0 Interface Pin Assignments

Pin Name/ Ball #	Type	Full Name	Description
Clk0 AH16	I	PCI_0 Clock	The PCI clock range is between 0 and 66MHz. PClk0 is completely independent of PClk1, SysClk and TClk.
Rst0* AJ16	I	PCI_0 Reset	Dedicated reset signal for PCI_0 interface. **NOTE:** Does not affect PCI_1 interface or any other GT–64240A unit. When in the reset state, all PCI_0 output pins are put into tristate and all open drain signals are floated.
VREF0 AF17	I	PCI_0 Voltage Reference	This pin must be connected directly to the 3.3V or the 5V power plane depending on which voltage level PCI_0 supports. VREF0 and VREF1 can be completely independent voltage levels.
PAD0[63:0]	t/s I/O	PCI_0 Address/Data	64-bit PCI_0 multiplexed address/data bus. Driven by the transaction master during address phase and write data phase. Driven by the target device during read data phase. **NOTE:** If configured with a 64-bit PCI, PAD0[63:32] requires a pull-up. When configured as a 32-bit bus, the GT–64240A drives these pins; a pull-up is not required.
[63:54] AL27, AK27, AJ27, AH27, AG27, AL28, AK28, AJ28, AH28, AL29 [53:44] AK29, AJ29, AJ30, AJ31, AH31, AH30, AH29, AG31, AG30, AG29 [43:32] AG28, AF31, AF30, AF29, AF28, AF27, AE31, AE30, AE29, AE28, AE27, AE26 [31:24] AL17, AK17, AJ17, AH17, AG17, AL18, AK18, AJ18 [23:14] AL19, AK19, AJ19, AH19, AG19, AL20, AK20, AJ20, AJ22, AH22 [13:0] AG22, AF22, AL23, AK23, AJ23, AH23, AF23, AL24, AK24, AJ24, AH24, AG24, AF24, AL25			
CBE0[7:0]* AL26, AK26, AJ26, AG26, AH18, AH20, AK22, AG23	t/s I/O	PCI_0 Command/Byte Enable	8-bit multiplexed command/byte-enable bus, driven by transaction master. Contains command during the address phase and byte-enable during data phase. **NOTE:** If configured with a 64-bit PCI, CBE0[7:4] requires a pull-up. When configured as a 32-bit bus, the GT–64240A drives these pins; a pull-up is not required.
PAR0 AL22	t/s I/O	PCI_0 Parity (low)	Even parity calculated for PAD0[31:0] and CBE0[3:0]. Driven by transaction master for address phase and write data phase. Driven by target for read data phase.

Table 4: PCI Bus 0 Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
FRAME0* AG20	s/t/s I/O	PCI_0 Frame	Asserted by the transaction master to indicate the beginning of a transaction. The master de-asserts FRAME0* to indicate that the next data phase is the final data phase transaction.
IRDY0* AF20	s/t/s I/O	PCI_0 Initiator Ready	Asserted by the transaction master to indicate it is ready to complete the current data phase of the transaction. A data phase is completed when TRDY0* and IRDY0* are asserted.
DEVSEL0* AK21	s/t/s I/O	PCI_0 Device Select	Asserted by the target of the current access. As a master, the GT–64240A expects the target to assert DEVSEL0* within five bus cycles. If the target does not assert DEVSEL0* within the required bus cycles, the GT–64240A aborts the cycle. As a target, the GT–64240A asserts DEVSEL0* in a medium speed; two cycles after the assertion of FRAME0*.
TRDY0* AL21	s/t/s I/O	PCI_0 Target Ready	Asserted by the target to indicate it is ready to complete the current data phase of the transaction. A data phase is completed when when TRDY0* and IRDY0* are asserted.
STOP0* AJ21	s/t/s I/O	PCI_0 Stop	Asserted by target to indicate transaction termination. Used by a target device to generate a Retry, Disconnect, or Target Abort termination signal.
IDSEL0 AG18	I	PCI_0 Initialization Device Select	Asserted to act as a target device chip select during PCI configuration transactions.
REQ640* AJ25	s/t/s I/O	PCI_0 Request 64-bit Transfer	Asserted by the transaction master to indicate a request of a 64-bit bus width transaction. REQ64* timing is the same as FRAME0* timing. **NOTE:** A 64-bit transaction occurs when REQ640* and ACK640* are asserted.
ACK640* AK25	s/t/s I/O	PCI_0 Acknowledge 64-bit Transfer	Asserted by the target in response to REQ64* to indicate it accepts a 64-bit bus width transaction. ACK640* timing is the same as DEVSEL* timing. **NOTE:** A 64-bit transaction occurs when REQ640* and ACK640* are asserted.

Table 4: PCI Bus 0 Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
PAR640 AF26	t/s I/O	PCI_0 Parity (high)	In cases of a 64-bit PCI transaction, even parity is calculated for PAD0[63:32] and CBE0[7:4]. Driven by the transaction master for address phase and write data phase. Driven by the target for read data phase. **NOTE:** A pull-up is required. When configured as a 32-bit bus, the GT–64240A drives this pin; a pull-up is not required.
REQ0* AF16	t/s O	PCI_0 Bus Request	If using an external PCI arbiter, asserted by the GT–64240A PCI master to indicate it requires PCI bus mastership to initiate a new transaction. If using the internal PCI arbiter, leave unconnected.
GNT0* AG16	I	PCI_0 Bus Grant	If using an external PCI arbiter, asserted to indicates to the GT–64240A PCI master that bus mastership is granted. **NOTES:** – The PCI master drives the bus only when it's GNT0* signal is asserted and the bus is in idle state. – If using the GT–64240A internal PCI arbiter, a pull-up is required.
PERR0* AH21	s/t/s I/O	PCI_0 Parity Error	Asserted when a data parity error is detected. Asserted by a target device in response to bad address or write data parity, or by master device in response to bad read data parity.
SERR0* AG21	o/d O	PCI_0 System Error	Asserted when a serious system error (not necessarily a PCI error) is detected.
INT0* AK16	o/d O	PCI_0 Interrupt Request	Asserted by the GT–64240A when one of the unmasked internal interrupt sources is asserted.
ENUM* AH25	o/d O	Compact PCI Hot Swap ENUM* interrupt	If ENUM is enabled, asserted by the GT–64240A during hot swap insertion or removal.
LED AG25	t/s O	Compact PCI Hot Swap LED	Driven by the GT–64240A to turn the LED on/off.
HS AF25	I	Compact PCI Hot Swap Handle Switch	Sampled handle switch status to identify board insertion/removal. **NOTE:** If not using CompactPCI Hot Swap, must be tied to VCC or GND.

Table 4: PCI Bus 0 Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
P64EN AH26	I	Compact PCI Hot Swap 64-bit PCI Enable	The GT–64240A samples the P64EN* pin on reset de-assertion, rather then REQ640*, to determine whether it is connected to a 64-bit PCI bus.
PCI Bus 0 Interface Pin Count: 94			

Table 5: PCI Bus 1 Interface Pin Assignments

Pin Name/ Ball #	Type	Full Name	Description
Clk1 AE04	I	PCI_1 Clock	The PCI clock range is between 0 and 66MHz. PClk1 is completely independent of PClk0, SysClk, and TClk.
Rst1* AE05	I	PCI_1 Reset	Dedicated reset signal for PCI_1 interface. **NOTE:** It has no affect on PCI_0 interface nor any other GT–64240A unit. When in the reset state, all PCI output pins are put into tristate and all open drain signals are floated.
VREF1 AE08	I	PCI_1 Voltage Reference	This pin must be connected directly to the 3.3V or the 5V power plane depending on which voltage level PCI_1 supports. VREF0 and VREF1 can be completely independent voltage levels.
PAD1[63:0]	t/s I/O	PCI_1 Address/Data	64-bit PCI_1 multiplexed address/data bus. Driven by the transaction master during address phase and write data phase. Driven by the target device during read data phase. **NOTE:** If configured with a 64-bit PCI, PAD1[61:32] requires a pull-up. When configured as a 32-bit bus, the GT–64240A drives these pins; a pull-up is not required.
[63:54] AK10, AJ10, AH10, AG10, AF10, AL11, AK11, AJ11, AH11, AG11, [53:44] AL12, AK12, AJ12, AH12, AG12, AL13, AK13, AJ13, AH13, AG13 [43:32] AL14, AK14, AJ14, AH14, AG14, AF14, AL15, AK15, AJ15, AH15, AG15, AL16 [31:24] AE01, AF06, AF05, AF04, AF03, AF02, AF01, AG05 [23:14] AG02, AG01, AH04, AH03, AH02, AH01, AJ03, AJ02, AL06, AK06 [13:0] AJ06, AH06, AG06, AL07, AK07, AJ07, AG07, AF07, AL08, AK08, AJ08, AH08, AG08, AF08			

Table 5: PCI Bus 1 Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
CBE1[7:0]* AJ09, AH09, AG09, AF09, AG04, AJ01, AH05, AH07	t/s I/O	PCI_1 Command/Byte Enable	8-bit multiplexed command/byte-enable bus, driven by the transaction master. Contains command during the address phase and byte-enable during data phase. **NOTE:** If configured with a 64-bit PCI, CBE0[7:4] requires a pull-up. When configured as a 32-bit bus, the GT–64240A drives these pins; a pull-up is not required.
PAR1 AJ05	t/s I/O	PCI_1 Parity (low)	Even parity calculated for PAD1[31:0] and CBE1[3:0]. Driven by the transaction master for the address phase and write data phase. Driven by the target for the read data phase.
FRAME1* AL03	s/t/s I/O	PCI_1 Frame	Asserted by the transaction master to indicate the beginning of a transaction. The master de-asserts FRAME1* to indicate that the next data phase is the final data phase transaction.
IRDY1* AK03	s/t/s I/O	PCI_1 Initiator Ready	Asserted by the transaction master to indicate that it is ready to complete the current data phase of the transaction. **NOTE:** A data phase is completed when both TRDY1* and IRDY1* are asserted.
DEVSEL1* AK04	s/t/s I/O	PCI_1 Device Select	Asserted by the target of the current access. As a master, the GT–64240A expects the target to assert DEVSEL* within five bus cycles. If the target does not assert DEVSEL* within the required bus cycles, the GT–64240A aborts the cycle. As a target, the GT–64240A asserts DEVSEL1* in a medium speed, two cycles after the assertion of FRAME1*.
TRDY1* AL04	s/t/s I/O	PCI_1 Target Ready	Asserted by the target to indicate it is ready to complete the current data phase of the transaction. **NOTE:** A data phase is completed when both TRDY1* and IRDY1* are asserted.
STOP1* AJ04	s/t/s I/O	PCI_1 Stop	Asserted by the target to indicate transaction termination. STOP1* is used by a target device to generate a Retry, Disconnect, or Target Abort termination.
IDSEL1 AG03	I	PCI_1 Initialization Device Select	Asserted to act as a target device chip select during the PCI configuration transactions.

Table 5: PCI Bus 1 Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
REQ641* AK09	s/t/s I/O	PCI_1 Request 64-bit Transfer	Asserted by the transaction master to indicate a request of a 64-bit bus width transaction. REQ641* timing is the same as FRAME1* timing. **NOTE:** A 64-bit transaction occurs if the REQ641* and ACK641* signals are asserted. To determine whether it is connected to a 64-bit PCI bus, the GT–64240A samples the REQ641* pin on reset de-assertion.
ACK641* AL09	s/t/s I/O	PCI_1 Acknowledge 64-bit Transfer	Asserted by the target in response to REQ641* to indicate that it accepts a 64-bit bus width transaction. ACK641* timing is the same as Frame1* timing. **NOTE:** A 64-bit transaction occurs when REQ641* and ACK641* are asserted.
PAR641 AL10	t/s I/O	PCI_1 Parity (high)	In cases of 64-bit PCI transactions, even parity is calculated for PAD1[63:32] and CBE1[7:4]. Driven by the transaction master for address phase and write data phase. Driven by the target for read data phase. **NOTE:** A pull-up is required. When configured as a 32-bit bus, the GT–64240A drives this pin; a pull-up is not required.
REQ1* AE02	t/s O	PCI_1 Bus Request	In a case using an external PCI arbiter, asserted by the GT–64240A PCI master to indicate it requires the PCI bus mastership to initiate a new transaction. If using the internal PCI arbiter, leave unconnected.
GNT1* AE03	I	PCI_1 Bus Grant	In a case using an external PCI arbiter, asserted to indicate to the GT–64240A PCI master that bus mastership is granted. **NOTES:** – The PCI master drives the bus only when it's GNT* signal is asserted and the bus is in idle state. – If using the GT–64240A internal PCI arbiter, a pull-up is required.
PERR1* AL05	s/t/s I/O	PCI_1 Parity Error	Asserted when a data parity error is detected. Asserted by a target device in response to bad address or write data parity, or by a master device in response to bad read data parity.
SERR1* AK05	o/d O	PCI_1 System Error	Asserted when a serious system error (not necessarily a PCI error) is detected.
INT1* AE06	o/d O	PCI_1 Interrupt Request	Asserted by the GT–64240A when one of the unmasked internal interrupt sources is asserted.

Table 5: PCI Bus 1 Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
PCI Bus 1 Interface Pin Count: 90			

Table 6: SDRAM Interface Pin Assignments

Pin Name/ Ball #	Type	Full Name	Description
SDClkOut/SDClkIn A15	I/O	SDRAM Clock Output	Optional output clock to drive the SDRAM. **NOTE:** Under certain board topologies and multiple DRAM loads, the SDRAM clock may need to be driven from SDClkOut, using a zero delay clock buffer. If SDClkOut is used, the output control sinals have an improved output delay. For further information, see Section 5.13 “SDRAM Clocking” on page 118.
		SDRAM Clock Input	Optional input clock to sample read data from the SDRAM. **NOTE:** Under certain board topologies and multiple DRAM loads, the read data received from the SDRAM may need to be sampled with SDClkIn. For further information, see Section 5.13 “SDRAM Clocking” on page 118.
SRAS* E13	t/s O	SDRAM Row Address Select	Asserted by the GT–64240A to indicate an active ROW address driven on the DAdr lines. **NOTE:** If UMA enabled, acts as s/t/s pin.
SCAS* A11	t/s O	SDRAM Column Address Select	Asserted by the GT–64240A to indicate an active column address driven on the DAdr lines. **NOTE:** If UMA enabled, acts as s/t/s pin.
DWr* B11	t/s O	SDRAM Write	Asserted by the GT–64240A to indicate a write to SDRAM. **NOTE:** If UMA enabled, acts as s/t/s pin.
DAdr[12:0]	t/s O	SDRAM Address	Driven by the GT–64240A during SRAS* and SCAS* cycles to generate a 26-bit SDRAM address.
E16, B15, E15, F15, A14, B14, C14, D14, E14, A13, B13, C13, D13			
BankSel[1:0] C15, D15	t/s O	SDRAM Bank Select	Driven by the GT–64240A during SRAS* and SCAS* cycles to select one of the DRAM virtual banks.
SCS[3:0]* C16, D16, A12, B12	t/s O	SDRAM Chip Selects	Asserted by the GT–64240A to select a specific SDRAM physical bank. **NOTE:** If UMA enabled, acts as s/t/s pin.

Table 6: SDRAM Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
SDQM[7:0]* E17, A16, C12, E12, F17, B16, D12, F12	t/s O	SDRAM Data Mask	Asserted by the GT–64240A to select the specific bytes of the 64-bit SData bus to be written to the SDRAM. **NOTE:** If UMA enabled, acts as s/t/s pin.
SData[63:0]	t/s I/O	SDRAM Data Bus	Driven by the GT–64240A during write to SDRAM. Driven by SDRAM during reads.
[63:54] A24, B23, D23, F23, B22, D22, A21, C21, E21, B20 [53:44] D20, A19, C19, E19, B18, D18, B10, D10, F10, B09 [43:34] D09, F09, B08, D08, F08, B07, D07, F07, B06, D06 [33:24] F06, B05, A23, C23, E23, A22, C22, E22, B21, D21 [23:14] A20, C20, E20, B19, D19, A18, C18, E18, C10, E10 [13:0] A09, C09, E09, A08, C08, E08, A07, C07, E07, A06, C06, E06, A05, C05			
ECC[7:0] A17, C17, C11, E11, B17, D17, D11, A10	t/s I/O	SDRAM ECC byte	Driven by the GT–64240A during write to SDRAM. Driven by SDRAM during reads. **NOTE:** If not using ECC[7:0], a pull-up is required.
SDRAM Interface Pin Count: 103			

Table 7: Device Interface Pin Assignments

Pin Name/ Ball #	Type	Full Name	Description
BAdr[2:0] C03, B03, A03	t/s O	Device Burst Address	Driven by the GT–64240A during burst read/write transactions to a device. **NOTE:** The GT–64240A increments the burst address with each data transfer.
Wr[3:0]* D04, C04, B04, A04	t/s O	Device Write Byte Enables	Asserted by the GT–64240A to select the specific bytes out of the 32-bit AD bus to be written to the device.
AD[0]/BootCS* C01	t/s I/O	Boot Chip Select	Used as boot device chip select during the address phase.
		Data [0]	Used as data bit 0 during the data phase.
AD[1]/DevRW* D03	t/s I/O	Device Read-Write	Used as device read (‘1’) or write (‘0’) indication during the address phase.
		Data [1]	Used as data bit 1 during the data phase.

Table 7: Device Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
AD[27:2]	t/s I/O	Device Address	Used as device address during the address phase.
		Data[27:2]	Used as device data bus during the data phase.
[27:17] J04, J05, J06, H01, H02, H03, H04, H05, H06, G01, G02 [16:2] G03, G04, G05, G06, F01, F02, F03, F04, F05, E01, E02, E03, E04, D01, D02			
AD[31:28]/CS[3:0]* K05, J01, J02, J03	t/s I/O	Chip Select [3:0]	Used as device chip select during the address phase.
		Data [31:28]	Used as data bits [31:28] during the data phase.
CSTiming* E05	t/s O	Device Chip Select Timing	Active for the entire device transaction. Used to qualify DevRW*, CS[3:0]*and BootCS signals.
ALE C02	t/s O	Device Address Latch Enable	Used to latch the Address, BootCS*, CS[3:0]*, and DevRW* signals from the AD bus.
Ready* D05	I	Device Ready:	Used as cycle extender when interfacing a slow device. When inactive during a device access, access is extended until Ready* assertion. **NOTE:** If not using Ready*, tie to GND.
Device Interface Pin Count: 42			

Table 8: Ethernet Interface Pin Assignments

NOTE: Use E0 and E1 ports as interfaces between the three GT–64240A Fast Ethernet controllers and the external PHYs. The exact routing of E0 and E1 pins is determined via the Serial Ports Multiplex register, see Section 19. “Pins Multiplexing” on page 480 for more information. Table 13 on page 40 summarizes the functionality of the Fast Ethernet pins. For pull-up/pull-down requirements, see Section 19.3 “Serial Port Configuration” on page 484.

Pin Name/ Ball #	Type	Full Name	Description
E0[14:0]	I/O	Ethernet Port 0	MII or RMII interface.
[14:7] N05, M01, M02, M03, M04, M05, L01, L02 [6:0] L03, L04, L05, K01, K02, K03, K04			
E1[14:0]	I/O	Ethernet Port 1	MII or RMII interface.
[14:7] T05, T06, R01, R02, R03, R04, R05, P01 [6:0] P02, P03, P04, P05, P06, N01, N02			

Table 8: Ethernet Interface Pin Assignments (Continued)

NOTE: Use E0 and E1 ports as interfaces between the three GT–64240A Fast Ethernet controllers and the external PHYs. The exact routing of E0 and E1 pins is determined via the Serial Ports Multiplex register, see Section 19. "Pins Multiplexing" on page 480 for more information. Table 13 on page 40 summarizes the functionality of the Fast Ethernet pins. For pull-up/pull-down requirements, see Section 19.3 "Serial Port Configuration" on page 484.

Pin Name/ Ball #	Type	Full Name	Description
MDC N04	O	Management Data Clock	MDC is the Clk input divided by 64. Provides the timing reference for the transfer of the MDIO signal.
MDIO N03	I/O	Management Data In/Out	Used to transfer control information and status between PHY devices and GT–64240A.
Ethernet Interface Pin Count: 32			

Table 9: Serial Interface Pin Assignments

NOTE: Use E0 and E1 ports as interfaces between the three GT–64240A Fast Ethernet controllers and the external PHYs. The exact routing of E0 and E1 pins is determined via the Serial Ports Multiplex register, see Section 19. "Pins Multiplexing" on page 480 for more information. Table 13 on page 40 summarizes the functionality of the Fast Ethernet pins.

Pin Name/ Ball #	Type	Full Name	Description
S0[6:0] U03, U04, U05, T01, T02, T03, T04	I/O	Serial Port 0	MPSC port.
S1[6:0] V02, V03, V04, V05, V06, U01, U02	I/O	Serial Port 1	MPSC port.
Serial Interface Pin Count: 14			

Table 10: MPP Interface Pin Assignment

Pin Name/ Ball #	Type	Full Name	Description
MPP[31:0]	I/O	Multi Purpose Pins	
[31:22] AD01, AD02, AD03, AD04, AD05, AD06, AC01, AC02, AC03, AC04 [21:12] AC05, AC06, AB01, AB02, AB03, AB04, AB05, AB06, AA01, AA02 [11:0] AA03, AA04, AA05, Y01, Y02, Y03, Y04, Y05, W01, W02, W03, W04			
Core Clock Pin Count: 32			

Table 11: I^2C Interface Pin Assignments

Pin Name/ Ball #	Type	Full Name	Description
I2CSCK V01	o/d I/O	I^2C Clock	I^2C serial clock. Serves as output when the GT–64240A acts as an I^2C master. Serves as input when the GT–64240A acts as an I^2C slave.
I2CSDA W05	o/d I/O	I^2C Serial Data	Address or write data driven by the I^2C master or read response data driven by the I^2C slave.
I^2C Interface Pin Count: 2			

Table 12: JTAG Interface Pin Assignments

Pin Name/ Ball #	Type	Full Name	Description
TCK E24	I	JTAG Clock	Clock input for the GT–64240A JTAG controller. **NOTE:** A pull-down is required.
TRST D24	I	JTAG Reset	When asserted, resets the GT–64240A JTAG controller. **NOTE:** A pull-down is required.
TMS C24	I	JTAG Mode Select	Controls the GT–64240A JTAG controller state. Sampled with the rising edge of JTCLK. **NOTE:** A pull-up is required.
JTDO F25	O	JTAG Data Out	JTAG serial data output. Driven by the GT–64240A on falling edge of JTCLK.
TDI B24	I	JTAG Data In	JTAG serial data input. Sampled with JTCLK rising edge. **NOTE:** A pull-down is required.
JTAG Interface Pin Count: 5			

Use E0 and E1 ports as interfaces between the three GT–64240A Fast Ethernet controllers and the external PHYs. The exact routing of E0 and E1 pins is determined via the Serial Ports Multiplex register, see Section 19. “Pins Multiplexing” on page 480 for more information.

Table 13 summarizes the functionality of the Fast Ethernet pins.

Table 13: Fast Ethernet Pin Functionality

Pin Name	Type	Functionality	Description
MII Interface			
MTxEN0/1	O	Transmit Enable	Indicates that a packet is being transmitted to the PHY. MTxEN0/1/2 is synchronous to MTxCLK0/1/2, respectively.
MTxCLK0/1	I	Transmit Clock	Provides the timing reference for the transfer of the MTxEN0/1/2, MTxD0/1/2 signals respectively. It operates at either 25 MHz (100Mbps) or 2.5 MHz (10Mbps).
MTxD0/1[3:0]	O	Transmit Data	Data nibble output to the external PHY device. Synchronous to MTxCLK0/1/2, respectively.
MCOL0/1	I	Collision Detect	Indicates that a collision is detected on the wire. This input is ignored in full-duplex mode, and in half duplex mode when MTxEn0/1/2 is LOW, respectively. MCOL0/1/2 is asynchronous.
MRxD0/1[3:0]	I	Receive Data	Data nibble input from external PHY. Synchronous to MRxCLK0/1/2, respectively.
MRxER0/1	I	Receive Error	Indicates that an error was detected in the received frame. This input is ignored when MRxDV0/1/2 is inactive, respectively.
MRxCLK0/1	I	Receive Clock	Provides the timing reference for the transfer of the MRxDV0/1/2, MRxD0/1/2, and MRxER0/1/2 signals respectively. Operates at either 25 MHz (100Mbps) or 2.5 MHz (10Mbps). **NOTE:** The nominal frequency of MRxCLK0/1/2 must match the nominal frequency of MTxCLK0/1/2, respectively.
MRxDV0/1	I	Receive Data Valid	Indicates that valid data is present on RxD0/1/2 lines (see Table 14), respectively. Synchronous to MRxCLK0/1/2, respectively.
MCRS0/1	I	Carrier Sense	In half duplex mode, this signal indicates that the transmit or receive medium is non-idle. **NOTE:** MCRS0 is ignored in full-duplex.
RMII Interface			
MTxEN0/1/2	O	Transmit Enable	Indicates that a packet is being transmitted to the PHY. MTxEN1/2 is synchronous to REF_CLK.
REF_CLK	I	Reference Clock	Provides the timing reference for the transfer of the MTxEN0/1/2, MTxD0/1/2, MRxD0/1/2, RxDV0/1/2 pins. Operates at 50MHz.

Table 13: Fast Ethernet Pin Functionality (Continued)

Pin Name	Type	Functionality	Description
MTxD0/1/ 2[1:0]	O	Transmit Data	MTxD0/1/2[1:0] is the transmit data output from MPCC, synchronous to REF_CLK.
MRxD0/1/ 2[1:0]	I	Receive Data	MRxD0/1/2[1:0] is the received input data. Synchronous to REF_CLK.
CRS_DV	I	Carrier Sense/Data Valid	Carrier Sense and Data Valid (RMII CRS_DV). Synchronous to REF_CLK.

.

Table 14: Serial Port Functionality

Pin Name	Type	Functionality	Description
MPSC Interface			
TxD0/1	O	Serial Transmit Data	Serial transmit data input to the MPSC.
RxD0/1	I	Serial Receive Data	Serial receive data input to the MPSC.
RTS0/1*	O	Request to Send	Indicates that the MPSC is ready to transmit data.
CTS0/1*	I	Clear to Send	Indicates to the MPSC that data transmission may begin.
CD0/1	I	Carrier Detect	Indicates to the MPSC that it can begin reception of data.
SCLK0/1 OSCLK0/1	I/O	Serial Input Clock (SCLK)	Can be used as both transmit and receive clock. Also serves as one of the input clocks to the Baud Rate Generators.
		Serial Output Clock (OSCLK)	Can be used when SCLK is not required. For example the MPSC is programmed to use one of the Baud Rate Generators as its clock source.
TSCLK0/1 OTSCLK0/1	I/O	Transmit Input Clock (TSCLK)	Can be used by the MPSC transmitter when separate receive and transmit clocks are needed. This clock also serves as one of the input clocks to the Baud Rate Generators.
		Transmit Output Clock (OTSCLK)	Can be used when TSCLK is not required. For example, the MPSC is programmed to use one of the Baud Rate Generators as its clock source or when there is no need for a separate Tx clock.

Use Multi Purpose Pins (MPPs) as peripherals interfaces or as general purpose I/Os. The exact routing of MPP pins is determined via the MPP Control register, see Section 19. “Pins Multiplexing” on page 480 for more information.

Table 15 summarizes the MPP pins functionality.

Table 15: MPP Pins Functionality

Pin Name	Type	Functionality	Description
DMAReq[7:0]*	I	DMA Request [7:0]	DMA channel trigger by external device.
DMAAck[7:0]*	O	DMA Acknowledge [7:0]	DMA channel acknowledge. Driven by the GT–64240A in response to DMAReq* when channel is activated.
EOT[7:0]	I	End of DMA Transfer [7:0]	External termination of a DMA channel operation.
TCEn[7:0]	I	Timer/Counter[7:0] Count Enable	Count enable input. **NOTE:** One pin per timer/counter. Counting starts two TClk cycles after TCEn assertion.
TCTCnt[7:0]	O	Timer/Counter[7:0] Terminal Count	Terminal count output. **NOTE:** One pin per timer/counter. Asserted one TClk cycle after the counter reaches zero.
GPP[31:0]	I/O	General Purpose Port [31:0]	General purpose input/output port, see Section 18. “General Purpose Port” on page 477 for more information.
InitAct	O	Initialization Active	Driven to 1 for the entire serial ROM initialization period.
PME0*	o/d O	PCI_0 Power Management Event	If PME is enabled, asserted by the GT–64240A upon CPU request.
PME1*	o/d O	PCI_1 Power Management Event	If PME is enabled, asserted by the GT–64240A upon CPU request.
MREQ*	I/O	UMA Request	SDRAM bus request asserted by a UMA slave device.
MGNT*	I/O	UMA Grant	Asserted by the UMA master in response to MREQ* to indicate bus mastership to the UMA slave device.
PCI0Req[5:0]*	I	PCI_0 Request[5:0]	External PCI bus requests when the GT–64240A PCI_0 bus arbiter is enabled.
PCI0Gnt[5:0]*	O	PCI_0 Grant[5:0]	Bus grant to external PCI masters when the GT–64240A PCI_0 bus arbiter is enabled.
PCI1Req[5:0]*	I	PCI_1 Request[5:0]	External PCI bus requests when the GT–64240A PCI_1 bus arbiter is enabled.
PCI1Gnt[5:0]*	O	PCI_1 Grant[5:0]	Bus grant to the external PCI masters when the GT–64240A PCI_1 bus arbiter is enabled.
DBurst*/ DLast*	O	Device Burst/Last	Used as device burst indication during the device access address phase. Indicates access of more than one data. Latching is done via ALE. Used as last data indication during the device data phase. Asserted on last data phase.

Table 15: MPP Pins Functionality (Continued)

Pin Name	Type	Functionality	Description
Int[3:0]*	O	CPI Interrupt[3:0]	Four CPU interrupt pins. These pins may be used to drive the different R5000/ RM7000 interrupt inputs.
BClkIn	I	Baud Rate Generator Clock In	Optional BRG clock input.
BClkOut0	O	Baud Rate Generator 0 Clock Out	Optional clock output of baud rate generator 0
WDNMI*	o/d O	Watch Dog NMI	Watch dog non-maskable interrupt.
WDE*	o/d O	Watch Dog Expired Interrupt	Typically causes the system to reset.
Debug[31:0]	O	Debug Port	Reserved for Galileo Technology usage.

3. ADDRESS SPACE DECODING

The GT–64240A has a fully programmable address map.

Three address spaces exist:

- The CPU address space.
- The PCI_0 address space.
- The PCI_1 address space.

The GT–64240A supports an advanced address decoding scheme. Every target device has its dedicated Address Map Registers. Each register can map up to 4GByte of space per device.

The IDMA and the Comm ports SDMAs use CPU address space map. However, they have an override capability that enables bypassing CPU address decoding and allows for direct transactions to the PCI bus.

NOTE: The GT–64240A address decoding is NOT software compatible with GT-64120/GT-64130 address decoding scheme. There is no two stage decoding process. Instead of a first level decoding of a device group followed by a second level decoding of the specific target device, the GT–64240A implements one level decoding that maps directly to the target device.

3.1 CPU Address Decoding

The CPU interface address decoding map consists of 20 address windows for the different devices, as shown in Table 16.

Each window can have a minimum of 1Mbytes of address space, and up to 4Gbyte space.

Table 16: CPU Interface Address Decoder Mappings

CPU Decoder	Associated Target
SCS[3:0]*	SDRAM chip selects.
CS[3:0]*, BootCS*	Devices chip selects.
PCI_0 I/O	PCI_0 I/O space.
PCI_0 Mem 0/1/2/3	PCI_0 Memory space.
PCI_1 I/O	PCI_1 I/O space.
PCI_1 Mem 0/1/2/3	PCI_1 Memory space.
Internal	GT–64240A internal registers.

Each address window is defined by two registers - Low and High. The CPU address is compared with the values in the various CPU Low and High Decode registers.

Address decoding works as follows:

1. Bits [35:32] of the CPU address are compared against bits [15:12] in the various CPU Low Decode registers. These values must match exactly ([35:32] = [15:12]).
2. Bits [31:20] of the CPU address are compared against bits [11:0] in the various CPU Low Decode registers. The value must be greater than or equal to the Low decode value ([31:20] >= [11:0]). This sets the lower boundary for the region.
3. Bits [31:20] of the CPU address are compared against the High Decode registers. The value must be less than or equal to this value ([31:20] <= High Decode register values). This sets the upper bound for the region.
4. If all of the above are true, the exact target device (e.g SCS[0]*) is selected

Example of the CPU address decode process is shown in Figure 2.

Figure 2: CPU Address Decode Example

If the CPU address is between the Low and the High decode addresses, then the access is passed to the target device.

35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	CPU Address Bits
=	=	=	=	>=	>=	>=	>=	>=	>=	>=	>=	>=	>=	>=	>=	Low Decode Register
				<=	<=	<=	<=	<=	<=	<=	<=	<=	<=	<=	<=	High Decode Register

Example: Set up a CPU decode region that starts at 0xA.4000.0000 and is 1Gbytes in length (0xa.4000.0000 to 0xa.7fff.ffff):

35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	CPU Address Bits
1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	Low Decode Register
				0	1	1	1	1	1	1	1	1	1	1	1	High Decode Register

NOTE: The CPU address windows are restricted to a size of 2^n and the start address must be aligned to the window size. For example, if using a 16 MB window, the start address bits [23:0] must be 0.

3.2 PCI Address Decoding

PCI slave interface address decoding map consists of 14 address windows for the different devices, as shown in Table 17.

Table 17: PCI Interface Address Decoder Mappings

PCI_0 Slave Decoder	Associated Target
SCS[3:0]*	SDRAM chip selects.
CS[3:0]*, BootCS*	Devices chip selects.
P2P Mem 0/1	Second PCI bus memory space.
P2P I/O	Second PCI bus I/O space.
Internal Mem	Memory mapped internal registers.
Internal I/O	I/O mapped internal registers.

In addition, PCI slave supports 11 more address windows for 64-bit addressing (using PCI Dual Access Cycle [DAC] transactions), as shown in Table 18.

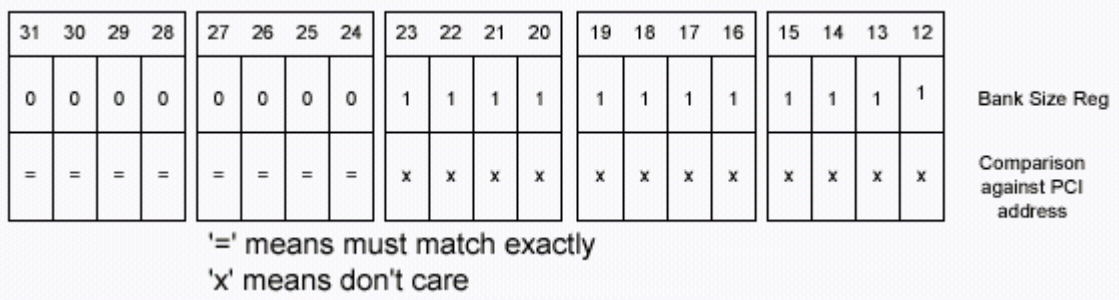
Table 18: PCI Interface 64-bit Addressing Address Decoder Mappings

PCI_0 Slave Decoder	Associated Target
DAC SCS[3:0]*	SDRAM chip selects.
DAC CS[3:0]*, DAC BootCS*	Devices chip selects.
DAC P2P Mem 0/1	Second PCI bus memory space.

NOTE: DAC address windows are not necessarily used for 64-bit addressing. They can be used as regular 32-bit addressing windows, allowing additional flexibility to PCI address mapping. See Section 8.5.5 “64-bit Addressing BARs” on page 162 for more details.

Each address window has two registers that defines the device address range - BAR (Base Address Register) and Size registers. Decoding starts with the PCI address being compared with the values in the various BARs. The size register sets which address bits are significant for the comparison between the active PCI address and the values in the BAR (see Figure 3).

Figure 3: Bank Size Register Function Example (16Meg Decode)



Bits [31:N] of the PCI address are compared against bits [31:N] in the various Base Address Registers (BARs). These values much match exactly. The value of ‘N’ is set by the least significant bit with a ‘0’ in the Bank Size Registers. For example, ‘N’ would be equal to 24 in the example shown in Figure 3.

The Bank Size register defines the size of the target device. It must be programed as a set of 1’s (staring from LSB) followed by a set of 0’s. The set of 1’s defines the size. For example, if the size register is set to 0x001fffff, it defines a size of 2Mbyte (number of 1’s is 21, 2^{21} = 2Mbyte).

As shown in Figure 3, PCI address is decoded starting with bit[12]. This means that each target device can have a minimum of 4Kbyte of address space.

3.3 Disabling Address Decoders

To disable the CPU address decoding window, set the value of the Low decoder to be higher than the High decoder.

PCI address decoding can be disabled through a BAR Enable register. If a BAR is disabled (it’s corresponding bit in BAR Enable register is set to ‘1’), the GT–64240A does not respond (no DEVSEL* asserted) to a PCI transaction that it’s address match the BARs address space, see Table 203 on page 197.

3.4 IDMA and Communication Unit Address Decoding

The IDMA and Communication Unit uses the address mapping of the CPU interface.

Whenever a DMA is activated, the DMA controller uses the CPU interface address mapping to determine whether the address is located in one of the SDRAM banks, Device banks, PCI_0, PCI_1, or CPU bus.

NOTE: The DMAs address decoding process is exactly the same as the CPU process. See Section 3.1 “CPU Address Decoding” on page 44 for details.

3.5 Address Space Decoding Errors

When the CPU tries to access an unmapped address:

- The GT–64240A latches the address into the CPU Error Address registers, see Section 4.15.5 “CPU Error Report Registers” on page 99.
- The CPU AddrErr bit [0] in the CPU Error Cause register is set, see Table 119 on page 100.
- An interrupt is asserted (if not masked).

This feature is especially useful during software debug, when errant code can cause fetches from unsupported addresses.

With CPU read from an unmapped address, a bus error indication is driven on SysCmd[5].

A PCI access that misses all of the GT–64240A BARs results in no response at all from the GT–64240A, since the address is targeted to some other target device on the PCI bus.

When an IDMA accesses an unmapped address:

- The GT–64240A latches the address into the DMA Error Address register, including failing DMA channel indication.
- The DMA AddrErr bit in the Interrupt Cause register is set, see .
- An interrupt is asserted (if not masked).

For an access to an unmapped address, the Communication Unit behavior is the same as the IDMA behavior.

NOTE: Address space decoders must never be programmed to overlap. Overlapping address space decoders results in unpredictable part behavior.

3.6 Default Memory Map

Table 19 shows the default CPU memory map that is valid following RESET.

Table 19: CPU Default Address Mapping

Decoder	Address Range
SCS0*	0x0 to 0x007f.ffff 8 Megabytes
SCS1*	0x0080.0000 to 0x00ff.ffff 8 Megabytes
SCS2*	0x0100.0000 to 0x017f.ffff 8 Megabytes
SCS3*	0x0180.0000 to 0x01ff.ffff 8 Megabytes
CS0*	0x1c00.0000 to 0x1c7f.ffff 8 Megabytes
CS1*	0x1c80.0000 to 0x1cff.ffff 8 Megabytes
CS2*	0x1d00.0000 to 0x1dff.ffff 16 Megabytes

Table 19: CPU Default Address Mapping (Continued)

Decoder	Address Range
CS3*	0x1f00.0000 to 0x1f7f.ffff 8 Megabyte
BootCS*	0x1f80.0000 to 0x1fff.ffff 8 Megabytes
Internal Registers	0x1400.0000 to 0x1400.ffff or 0xf100.0000 to 0xf100.ffff **NOTE:** Set at reset configuration AD[24]. 64 Kbytes
PCI_0 Mem0	0x1200.0000 to 0x13ff.ffff 32 Mbytes
PCI_0 Mem1	0xf200.0000 to 0xf3ff.ffff 32 Megabytes
PCI_0 Mem2	0xf400.0000 to 0xf5ff.ffff 32 Mbyte
PCI_0 Mem3	0xf600.0000 to 0xf7ff.ffff 32 Mbyte
PCI_0 I/O	0x1000.0000 to 0x11ff.ffff 32 Mbytes
PCI_1 Mem0	0x2200.0000 to 0x23ff.ffff 32 Mbytes
PCI_1 Mem1	0x2400.0000 to 0x25ff.ffff 32 Mbytes
PCI_1 Mem2	Ox2600.0000 to 0x27ff.ffff 32Mbyte
PCI_1 Mem3	0x2800.0000 to 0x29ff.ffff 32Mbyte
PCI_1 I/O	0x2000.0000 to 0x21ff.ffff 32 Mbytes

Table 20 shows the default PCI memory map that is valid following RESET.

Table 20: PCI Default Address Mapping

Decoder	Address Range
SCS0*	0x0 to 0x007f.ffff 8 Megabytes

Table 20: PCI Default Address Mapping

Decoder	Address Range
SCS1*	0x0080.0000 to 0x00ff.ffff 8 Megabytes
SCS2*	0x0100.0000 to 0x017f.ffff 8 Megabytes
SCS3*	0x0180.0000 to 0x01ff.ffff 8 Megabytes
CS0*	0x1c00.0000 to 0x1c7f.ffff 8 Megabytes
CS1*	0x1c80.0000 to 0x1cff.ffff 8 Megabytes
CS2*	0x1d00.0000 to 0x1dff.ffff 16 Megabytes
CS3*	0x1f00.0000 to 0x1f7f.ffff 8 Megabyte
BootCS*	0x1f80.0000 to 0x1fff.ffff 8 Megabytes
Internal Mem	0x1400.0000 to 0x1400.ffff 64 Kbytes
Internal I/O	0x1400.0000 to 0x1400.0fff 64 Kbytes
P2P Mem0	PCI_0: 0x2200.0000 to 0x23ff.ffff PCI_1: 0x1200.0000 to 0x13ff.ffff 32 Mbytes
P2P Mem1	PCI_0: 0x2400.0000 to 0x25ff.ffff PCI_1: 0xf200.0000 to 0xf3ff.ffff 32 Megabytes
P2P I/O	PCI_0: 0x2000.0000 to 0x21ff.ffff PCI_1: 0x1000.0000 to 0x11ff.ffff 32 Mbytes

Table 21 shows the default 64-bit addressing PCI memory map that is valid following RESET.

Table 21: 64-bit Addressing PCI Default Address Mapping

Decoder	Address Range
DAC SCS0*	0x0 to 0x007f.ffff 8 Megabytes
DAC SCS1*	0x0080.0000 to 0x00ff.ffff 8 Megabytes
DAC SCS2*	0x0100.0000 to 0x017f.ffff 8 Megabytes
DAC SCS3*	0x0180.0000 to 0x01ff.ffff 8 Megabytes
DAC CS0*	0x1c00.0000 to 0x1c7f.ffff 8 Megabytes
DAC CS1*	0x1c80.0000 to 0x1cff.ffff 8 Megabytes
DAC CS2*	0x1d00.0000 to 0x1dff.ffff 16 Megabytes
DAC CS3*	8 Megabyte
DAC BootCS*	8 Megabytes
DAC P2P Mem 0	0x2600.0000 to 0x27ff.ffff 32 Mbyte
DAC P2P Mem 1	0x2800.0000 to 0x29ff.ffff 32 Mbyte

3.7 Programming Address Decoding Registers

Since the software can't tell how long it takes for the programing to be executed within the GT–64240A, programming the address decoding registers might be problematic. Also, The software must confirm that the programing actually happened, before it attempts to access GT–64240A with an address that matches the new programed decoder.

3.7.1 PCI Programming of Address Decoders

PCI accesses to the GT–64240A PCI registers (including the Base Address register) are never posted.

The PCI slave completes the transaction on the PCI bus (asserts TRDY*) only when data is actually written to the register. This implementation guarantees that any new PCI accesses to GT–64240A only occurs after the registers are updated. There is no special software requirement.

3.7.2 CPU Programming of Address Decoders

The CPU setting of the CPU interface address decoders requires special care, especially if changing the mapping of the GT–64240A internal space. If for example, the CPU changes the Internal Space Decode Address register and accesses the internal registers based on the new address, the CPU might get an address mismatch, since the register is not updated yet.

To change Internal Space Decode Address register, perform the following steps:

1. If the required new value overlaps another address decoder, disable this address decoder. See Section 3.3 “Disabling Address Decoders” on page 47 for details.
2. Read the Internal Space Decode Address register. This guarantees that all previous transaction in the CPU interface pipe are flushed.
3. Only after the CPU interface pipe is flushed, program the register to its new value.
4. Read polling of the register. If the new value is not updated, there is an address mismatch and data of 0xffffffff is returned.

NOTE: The Address mismatch interrupt must be masked, in order to prevent a CPU interrupt.

5. Once a valid data is being read, the software continues to program the GT–64240A registers, based on the new Internal Space address.

NOTE: Instead of step #4, it is possible to use a wait loop of 8 SysClk cycles.

3.8 Address Remapping

The GT–64240A supports address remapping from CPU side and from PCI side. Address remapping enables to relocate an address range defined by address decoding registers, to a new location in the target address space.

3.8.1 CPU Address Remapping to PCI

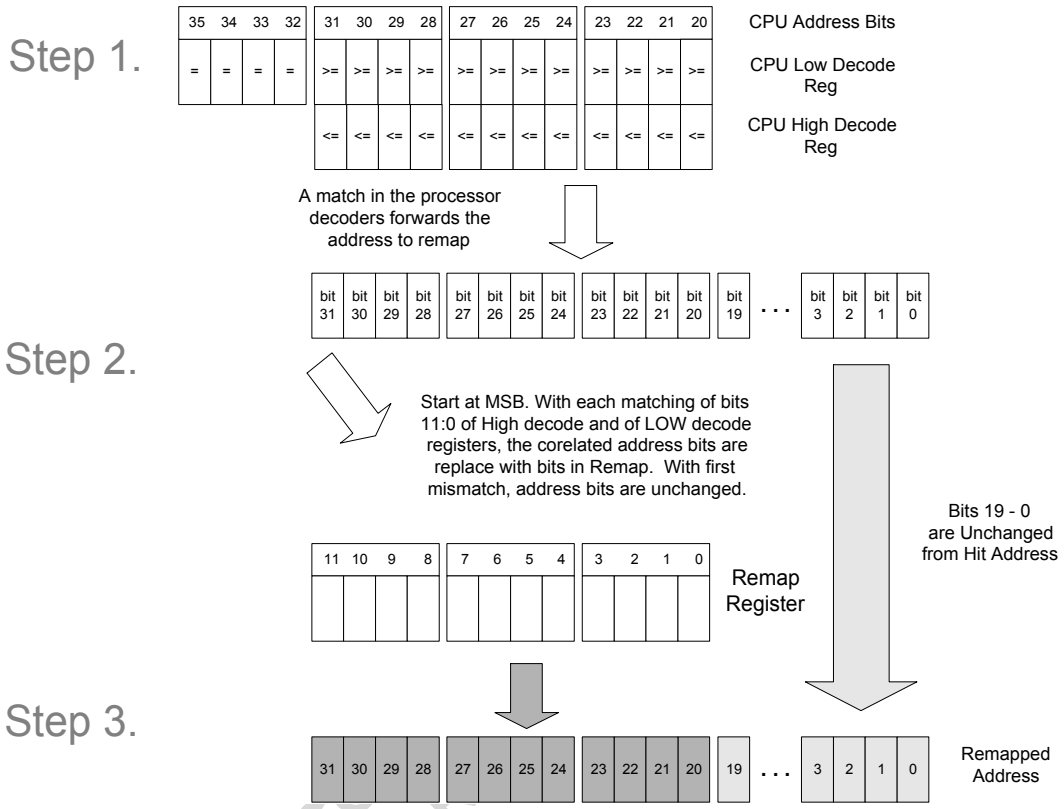
Each of the CPU to PCI address windows has a Remap Register associated with it.

An address presented on the CPU bus is decoded using the following steps:

1. Address bits [35:20] are checked for a hit in the CPU decoders.
2. Assuming there is a hit in the CPU decoders, part of bits[31:20] are remapped according to the resource size. Going from the MSB to LSB of the High Decode registers, any bit found matching to its respective bit in the LOW Decode register causes the corresponding bit in the Remap register to REPLACE the respective address bit. Upon the first mismatch, all remaining LSBs of address bits[31:20] are unchanged. Bits 19:0 are left unchanged.
3. The remapped address is transferred to the PCI bus.

See Figure 4 outlining this address remapping procedure.

Figure 4: CPU Address Remapping



3.8.2 Writing to CPU Decode Registers

When a LOW Decode register is written to, the least significant 12 bits are simultaneously written to the associated Remap register.

When a Remap register is written to, only its contents are affected. Following RESET, the default value of a Remap register is equal to its associated LOW Decode register bits [11:0]. Unless a specific write operation to a Remap register takes place, a 1:1 mapping is maintained.

Also, changing a LOW Decode register's contents automatically returns its associated space to a 1:1 mapping. This allows users that do not need this address remapping feature to change the CPU interface address decoding windows without dealing with the associated remap registers.

When setting RemapWrDis bit in CPU Configuration register to 1, writing to the LOW Decode register does not result in simultaneous write to the corresponding Remap registers.

3.8.3 PCI Address Remapping

Each of the PCI interface address windows has a Remap Register associated with it. An address presented on the PCI AD bus is decoded with the following steps:

1. Address bits [31:12] are checked for a hit in the PCI Base/Size registers.
2. Assuming there is a hit, bits[31:12] are remapped as follows:
   - Any address bit that is not masked by the Size register is REPLACED by the corresponding bit of the remap register.
   - Address bits that are masked by the size register are left unchanged.
3. The remapped address is transferred to the target device.

An example of this is summarized in Table 22.

Table 22: PCI Address Remapping Example

PCI address	0x1d98.7654
SCS[0]* BAR	0x1c00.0000
SCS[0]* Size	0x03ff.ffff
SCS[0]* Remap Register	0x3c00.0000
Remapped PCI Address Presented to SDRAM	0x3d98.7654

In Table 22, the Size register is programmed to 0x03ff.ffff. This indicates that this BAR requires a hit in the six MSB (bits 31:26) bits of the PCI address for their to be a hit in the BAR.

Therefore, the PCI address 0x1dxx.xxxx is a hit in a BAR programmed to 0x1fxx.xxxx as bits [31:26] of both of these addresses is 0b0001.11.

Then according to the Remap register, these same bit locations are remapped to 6'b111111. The rest of the PCI address bits (i.e. [25:0]) remain unchanged. This means that the final PCI slave address is 0x3d987654.

3.8.4 Writing to PCI Decode Registers

When a BAR register is written to, the associated Remap register is written to, simultaneously.

When a Remap register is written to, only its contents are affected. Following RESET, the default value of a Remap register is equal to its associated BAR decode register. Unless a specific write operation to a Remap register takes place, a 1:1 mapping is maintained.

Also, changing a BAR register's contents automatically returns its associated space to a 1:1 mapping. This allows users that do not need this address remapping feature to change the PCI interface address decoding windows without dealing with the associated remap registers.

In some applications, the operating system might re-program the Base Address registers after the Remap registers were already programed by the local driver. In such case, the 1:1 mapping due to the BARs re-programing is not desired.

If RemapWrDis bit in PCI Address Decode Control register is set to 1, writing to the BARs will NOT result in simultaneous write to the corresponding Remap registers.

3.8.5 64-bit Remap Registers

The CPU interface PCI memory windows and PCI interface P2P memory windows have the capability of remapping to 64-bit addresses. In addition to the regular remap register, each window has a High Remap register that

sets the upper 32-bit address. This enables access to addresses beyond the 4Gbyte space on the PCI bus using DAC cycles.

If the High Remap register is set to 0x0 (default), the address driven to the PCI master interface is a 32-bit address and it generates a SAC transaction on the PCI bus.

If the High Remap register is programed to a value other than 0x0, it is used as the upper 32-bit address of the PCI transaction. The PCI master generates a DAC transaction on the PCI bus.

NOTE: See Section 8. “PCI Interface” on page 155 for more details.

3.9 IDMA and Communication Unit Address Decoding Override

In default, the IDMA and the communication units use the CPU interface address decoding as in Section 3.4 “IDMA and Communication Unit Address Decoding” on page 47. However, the units can be configured to bypass the address decoding and have direct access to the PCI bus.

It is possible to configure each of the IDMA channels to drive the source, destination, and descriptors address directly to the PCI_0 or PCI_1 interfaces, without going through the CPU interface address decoders. This option is also available for the communication ports.

For more details see, IDMA, Section 10.3 “IDMA Address Decoding” on page 278, and Communication unit, Section 12.1 “Address Decoding” on page 321.

4. CPU INTERFACE

The GT–64240A supports all MIPS 64-bit bus CPUs. These include:

- PMC-Sierra RM5261A, RM7000, RM7000A, RM7000C
- IDT RC5000, RC64575
- NEC Rv5000, Rv5464
- Any 64-bit SysAD compatible CPU

NOTE: The GT–64240A supports RM7000C Simplified External Cache mode.

The CPU interface can only work as a slave interface responding to CPU transactions.

4.1 CPU Address Decoding

The CPU interface uses a one stage decoding process, as described in Section 3. “Address Space Decoding” on page 44. This section summarizes CPU address decoding and emphasizes few details.

NOTE: For an exact list of CPU Address Decoding registers, seeTable 30 on page 73.

The CPU interface supports 20 address windows.

- Four for SDRAM chip selects.
- Five for device chip selects.
- Five for PCI_0 interface (4 memory + one I/O).
- Five for PCI_1 interface (4 memory + one I/O).
- One for the GT–64240A internal registers space.

NOTE: The CPU address windows are restricted to a size of 2^n and the start address must be aligned to the window size. For example, if using a 16 MB window, the start address bits [23:0] must be ‘0’.

Each window is defined by a Low and High register and can decode up to 4Gbyte space.

The CPU interface also supports address remapping to the PCI bus. This is useful when a CPU address range must be reallocated to a different location on the PCI bus. Also, it enables CPU access to a PCI agent located above the 4Gbyte space.

The CPU interface contains High PCI Remap registers that defines the upper 32-bit PCI address. If the register is set to 0, the CPU access to PCI results in a Single Address Cycle (SAC) transaction. If it is set to a value other than 0, the PCI master issues a DAC transaction with the high 32 address bits set according to the High PCI Remap register’s value.

The CPU accesses the GT–64240A internal registers space when address matches the Internal Space Low register.

NOTE: There is no High register for Internal Space, since it has a fixed size.

4.2 CPU Access Protection

The CPU interface supports configurable access protection. This includes up to eight address ranges defined to a different protection type - whether the address range is cacheable or not, whether it is writable or not, and whether it is accessible or not.

A Low and High register defines each address window. The minimum address range of each window is 1Mbyte.

An address driven by the CPU, in addition to the address decoding and remapping process, is compared against the eight Access Protection Low/High registers.

- Bits[35:32] of the address are checked to be equal to bits[23:20] of Low register.
- Bits[31:20] of the address are checked to be between the lower and upper addresses defined by bits[19:0] of Low and High registers.

If an address matches one of the windows, the GT–64240A checks the transaction type against the protection bits defined in CPU Access Protection register, to determine if the access is allowed.

Three types of protection are supported:

- Access protection: Any CPU access to this region is forbidden.
- Write protection: Any CPU write access to this region is forbidden.
- Cacheable protection: Any CPU burst access to this region is forbidden.

If there is an access violation, the CPU interface completes the transaction properly against the CPU but ignores the transaction internally. The transaction address is latched in the CPU Error Address register and the CPU AddrErr bit in the interrupt cause register is set.

4.3 CPU Slave Operation

The CPU slave interface contains 256 bytes of posted write data buffer and 64 bytes of read data buffer. It can absorb up to two read or write transactions.

The write buffer accepts up to eight cache lines. CPU writes are posted. They are written into the write buffer and only then driven to the target. If the target device is busy and cannot accept the transaction, the write buffer can still accept new CPU write transactions, with zero wait states.

The read buffer accepts up to two cache lines. The CPU interface tries to drive read data to the CPU when data arrives from the target device. If the bus is occupied by another bus master, data is written first to the read buffer.

The GT–64240A supports split read transactions. The CPU interface pipelines up to eight transactions to target devices. In this case, data may be returned out of order. For example, if the first read transaction is directed to the PCI and the second is directed to SDRAM, data from SDRAM will return first.

If the CPU supports out of order completion (e.g. RM7000), data from SDRAM is driven first on the CPU bus. If the CPU doesn't support out of order completion (e.g. R5000), the data must first be placed in a read buffer and then wait for the PCI read response to complete.

The CPU transactions are issued to the target device in order. The first transaction appearing on the CPU bus is the first one to be issued towards the target device. There is no transaction bypassing. The GT–64240A architecture guarantees the execution of the CPU consecutive transactions to the same target device in the same order they appeared on the CPU bus.

4.4 MIPS 64-bit Multiplexed Address/Data Bus Interface

The GT–64240A supports 64-bit MIPS CPUs multiplexed address/data bus protocol and partial read/writes from one byte up to eight bytes, as well as 32-byte block reads/writes.

4.4.1 Signals Description

The CPU interface incorporates the following signals:

Table 23: CPU Interface Signals

Signal	Description
SysAD[63:0]	Multiplexed address/data bus. Used as address during the issue cycle and as data during the read/write data phase.
SysCmd[8:0]	Multiplexed command/data identifier bus. Used as command during issue cycle (read/write, size information) and as data identifier during data phase (good/bad data, last data information).
SysADC[7:0]	SysAD parity bus: An 8-bit bus containing even parity for the SysAD bus. Valid only on the data phase.
ValidOut*	CPU indication for driving valid address/data and command/data identifier on the SysAD and SysCmd busses.
Release*	CPU indication for releasing the bus. The CPU stops driving SysAD and SysCmd busses the next cycle after Release* assertion. It is floating the busses for the GT–64240A completion of a read transaction.
ValidIn*	The GT–64240A indication for driving valid read data and data identifier on SysAD and SysCmd busses.
SysRdyOut*	The GT–64240A indication that it is capable of accepting a new read or write transaction.
SysRdyIn[2:0]*	SysRdy* input used in a multi-GT–64240A configuration.
PRqst*	The CPU request from the GT–64240A for bus mastership so it can issue a new transaction.
PAck*	The GT–64240A bus acknowledge to CPU. The CPU may issue a new transaction on the next cycle.
RspSwap*	The GT–64240A indication to the CPU that read data is returned out of order.
TcMatch	L3 cache Tag RAM hit indication. **NOTE:** Not relevant when working in Simplified External Cache mode.
TcDOE*	L3 cache data RAM output enable. Asserted by the GT–64240A on L3 read hit.
TcTCE*	L3 cache Tag RAM chip enable. Sampled by the GT–64240A to identify L3 access.

Table 23: CPU Interface Signals (Continued)

Signal	Description
TcWord[1:0]	L3 cache word index. Driven by the GT–64240A during L3 read miss.
CPUInt*	Level sensitive CPU interrupt asserted by the GT–64240A.

4.4.2 SysAD and SysCmd Encoding

SysCmd[8:0] is used to transfer command during the transaction address phase (SysCmd[8] = 0) and data identifier during data phase (SysCmd[8] = 1), as shown in Table 24.

Table 24: Read/Write Request Command Bits Summary

SysCmd Bit	Function
SysCmd[8]	0 = Command 1 = Data identifier
SysCmd[7:5]	0x0 - Read request 0x1 - Reserved 0x2 - Write request 0x3 - Null request 0x4-0x7 - Reserved
SysCmd[4:3]	0x0,0x1 - Reserved 0x2 - Block read or write 0x3 - Partial read or write
SysCmd[2] - block read/write	0 - Cache line not retained 1 - cache line retained
SysCmd[1:0] - block read/write	0x0 - Reserved 0x1 - 8 words block size 0x2,0x3 - Reserved
SysCmd[2:0] - partial read/write	0x0 - one byte 0x1 - 2 bytes 0x2 - 3 bytes 0x3 - 4 bytes 0x4 - 5 bytes 0x5 - 6 bytes 0x6 - 7 bytes 0x7 - 8 bytes

Table 25: Null Request Command Bits Summary

SysCmd Bit	Function
SysCmd[8:5]	0x3
SysCmd[4:3]	0x0 - Bus release 0x1-0x3 - Reserved
SysCmd[2:0]	Reserved

Table 26: Data Identifier Bits Summary

SysCmd Bit	Function
SysCmd[8]	1
SysCmd[7]	0 - Last data element 1 - Not last data element
SysCmd[6]	0 - Data is read response data 1 - Data is not response data
SysCmd[5]	0 - Data is error free 1 - Data is erroneous
SysCmd[4]	0 - Check data and check bits 1 - Do not check data and check bits Reserved for no read response data
SysCmd[3]	Reserved
SysCmd[2:0]	Cache state

Table 27: Partial Word Byte Lane
NOTE: On partial read/write transactions, the exact partial data being taken depends on address offset.

Byte Count (SysCmd[2:0])	SysAD [2:0]	SysAD Byte Lanes (Big Endian)							
		7:0	15:8	23:16	31:24	39:32	47:40	55:48	63:56
		SysAD Byte Lanes (Little Endian)							
		63:56	55:48	47:40	39:32	31:24	23:16	15:8	7:0
1 (000)	0x0	-	-	-	-	-	-	-	A
	0x1	-	-	-	-	-	-	A	-
	0x2	-	-	-	-	-	A	-	-
	0x3	-	-	-	-	A	-	-	-
	0x4	-	-	-	A	-	-	-	-
	0x5	-	-	A	-	-	-	-	-
	0x6	-	A	-	-	-	-	-	-
	0x7	A	-	-	-	-	-	-	-
2 (001)	0x0	-	-	-	-	-	-	A	A
	0x2	-	-	-	-	A	A	-	-
	0x4	-	-	A	A	-	-	-	-
	0x6	A	A	-	-	-	-	-	-
3 (010)	0x0	-	-	-	-	-	A	A	A
	0x1	-	-	-	-	A	A	A	-
	0x4	-	A	A	A	-	-	-	-
	0x5	A	A	A	-	-	-	-	-
4 (011)	0x0	-	-	-	-	A	A	A	A
	0x4	A	A	A	A	-	-	-	-
5 (100)	0x0	-	-	-	A	A	A	A	A
	0x3	A	A	A	A	A	-	-	-
6 (101)	0x0	-	-	A	A	A	A	A	A
	0x2	A	A	A	A	A	A	-	-
7 (110)	0x0	-	A	A	A	A	A	A	A
	0x1	A	A	A	A	A	A	A	-
8 (111)	0x0	A	A	A	A	A	A	A	A

4.4.3 SysAD Read Protocol

The CPU starts a read transaction with the assertion of ValidOut*. It drives the valid address and command on the SysAD and SysCmd busses. It also asserts Release* to indicate its release of the bus mastership to the GT–64240A for completion of the read.

Two cycles after the Release* assertion, the GT–64240A starts driving the bus.

NOTE: There is a one turn-around cycle between the CPU drive and the GT–64240A drive.

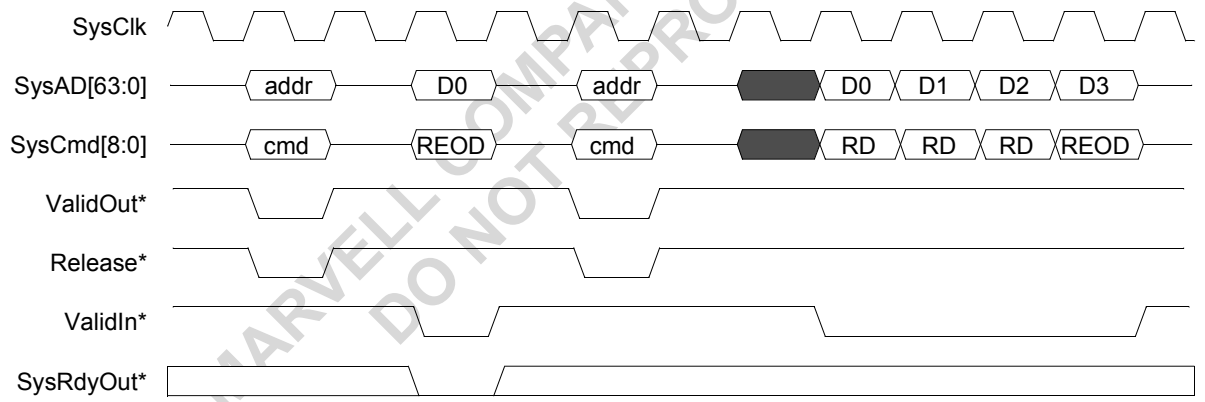
As soon as read data is available, the GT–64240A asserts ValidIn* and drives valid data on SysAD, and valid read response (mnemonic = RD) on SysCmd. On the last data, the GT–64240A drives last data identifier (mnemonic = REOD) on SysCmd.

On the clock cycle following REOD, the GT–64240A floats SysAD and SysCmd buses, returning ownership to the CPU.

NOTE: The CPU reads cannot be issued as long as SysRdyOut* is deasserted (HIGH). If SysRdyOut* is high and a CPU read is attempted, a previous transaction might be corrupted. All MIPs compliant processors follow this protocol. Only DMA engines on the SysAD bus that need to be concerned with sampling SysRdyOut* before initiating a read.

An example of two consecutive read transactions is shown in Figure 5.

Figure 5: SysAD Read Protocol



NOTE: Figure 5 is a demonstration of the SysAD read protocol. This figure does not reflect the actual read latency of the GT–64240A.

4.4.4 Write Protocol

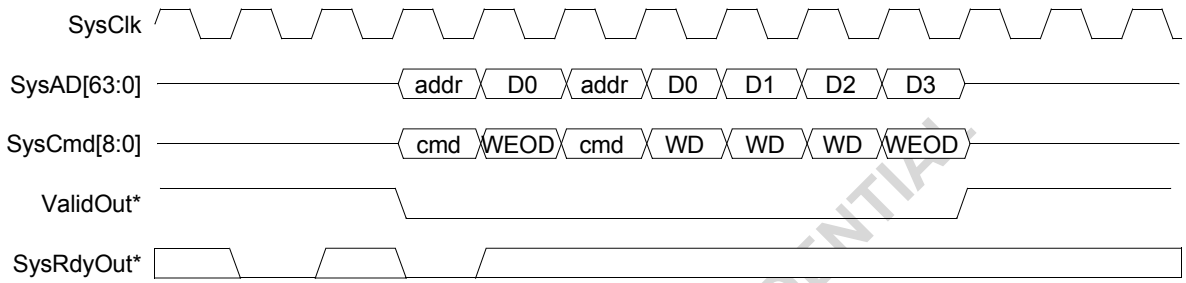
The CPU starts a write transaction with the assertion of ValidOut*. It drives a valid address and command on the SysAD and SysCmd busses. The next cycle it starts driving valid data on SysAD bus and a valid write command (mnemonic = WD) on the SysCmd bus. On the last data, it drives the last data identifier (mnemonic = WEOD) on the SysCmd bus. ValidOut* remains asserted throughout the write transaction.

NOTE: The CPU writes cannot be issued as long as SysRdyOut* is deasserted (HIGH). If SysRdyOut* is high and a CPU write is attempted, a previous transaction might be corrupted. All MIPs compliant processors

follow this protocol. Only the DMA engines on the SysAD bus need to be concerned with sampling SysRdyOut* before initiating a write.

An example of two consecutive back to back CPU write transactions is shown in Figure 6.

Figure 6: SysAD Write Protocol



4.5 RM7000 Split Transactions Support

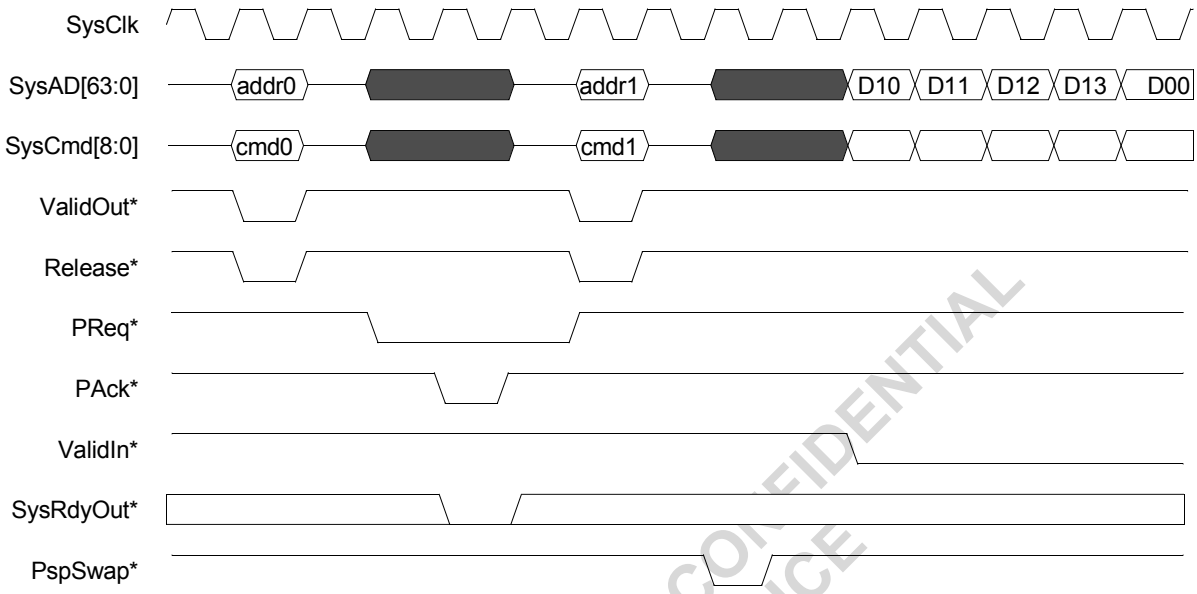
The GT–64240A supports the Non-Pendant mode of the RM7000 processor.

This mode allows the CPU to pipe up to two read transactions. Since RM7000 bus is multiplexed address/data bus, the CPU issues a new read transaction by gaining back bus mastership by asserting the PReq* signal. If the GT–64240A is able to handle a new read request, it acknowledges the CPU by asserting PAck* signal. The CPU then issues a new read transaction, and releases the bus (asserts Release*), enabling GT–64240A to complete both reads.

The RM7000 also supports out-of-order completion of the read transactions.

If the GT–64240A is able to complete the second read transaction before the first one, it asserts RspSwap*. This indicates to the CPU that the data is returned out-of-order. An example of two split read transaction with out-of-order completion is shown in Figure 7.

Figure 7: R7000 Split Read Transaction Example



NOTE: Figure 7 is a demonstration of the SysAD split read protocol. This figure does not reflect the actual read latency of the GT–64240A.

As explained, the pipeline support enables minimum CPU read latency. In the case of out-of-order completion, latency might be even better. In the above example, if the first read targets a slow device and the second read targets the SDRAM (which is fast), since data from SDRAM arrives first, it is driven first on the CPU bus with RspSwap* indication.

NOTE: The RM7000 pipeline is restricted to read transactions. The CPU never pipelines a read into a write transaction or a write into a read transaction.

When configured to multi-GT mode, the CPU Configuration register's RdOOO bit must be set to '0', see Table 92 on page 88. Out-of-order is not supported in multi-GT mode.

4.6 Burst Support

Block (cache line) read or write results in burst read/write transactions on the bus.

The MIPS CPU cache line is 32 bytes long. On a 64-bit wide bus, the CPU block read or write results in burst of four 64-bit words. Block write address is aligned to cache line (address bits[4:0] are 0). Block read address can

point to any of the four double-words of the cache line. Block read burst order is sub-block ordering, as shown in Table 28 (DW0 is the least significant dword, DW3 is the most).

Table 28: 64-bit Bus Sub-block Ordering

Data Transfer	Start Address - SysAD[4:3]			
	00	01	10	11
1st data beat	DW0	DW1	DW2	DW3
2nd data beat	DW1	DW0	DW3	DW2
3rd data beat	DW2	DW3	DW0	DW1
4th data beat	DW3	DW2	DW1	DW0

4.7 Transactions Flow Control

The MIPS CPUs bus protocol requires that a target accepting a write request completes the transaction with zero wait states

NOTE: A write transaction cannot be held in the middle.

This implies that for the GT–64240A to accept a new CPU write transaction it must have “room” in both the transactions queue and in the write data buffer.

The GT–64240A micro architecture guarantees that when there is “room” in the transaction queue there is also “room” in the read and write data buffers. Since the transaction queue is shared for reads and writes, and since only the transaction queue affects the GT–64240A’s ability to accept a new transaction, there is a single SysRdyOut* signal driven by the GT–64240A rather than separate RdRdy* and WrRdy*. The GT–64240A SysRdyOut* output must be connected to both RdRdy* and WrRdy* inputs of the CPU.

The GT–64240A supports two write modes:

- The R4000 compatible mode.
- Pipeline mode.

NOTE: For more details, see the CPU User’s Manual.

The CPU issues a new write request if its WrRdy* input samples low two cycles before the issue cycle. The GT–64240A CPU interface deasserts SysRdyOut* according to the write mode it is programed to use and according to the available room in its transaction queue.

The CPU issues a new read request if its RdRdy* input samples low two cycles before the issue cycle. The GT–64240A deasserts SysRdyOut* according to the available room in its transaction queue.

The write protocol does not allow data flow control - the write data rate is fixed. The MIPS CPUs support different write rates (in order to interface slow target devices). The GT–64240A supports only DDDD write pattern (64-bit data every clock cycle).

GT–64240A controls read data flow using the ValidIn* signal. If the CPU accesses a slow device, the GT–64240A keeps ValidIn* deasserted until read data arrives from the target device. In case of burst read from a slow device, the GT–64240A can deassert ValidIn* to create wait states between data beats.

4.8 MIPS CPU Cache Support

The GT–64240A supports third level (second level) cache placed on the SysAD bus.

NOTE: MIPS L3 cache implementation consists of an external Tag and data RAMs placed on the SysAD bus. The external RAMs control signaling is shared between the CPU and the GT–64240A. See the PMC-Sierra application note for more details.

The GT–64240A samples the TcMatch signal. In case of a CPU access that hits the L3 cache line (Tag RAM asserts TcMatch signal), the GT–64240A ignores the transaction. This enables the CPU to complete the transaction against L3 cache.

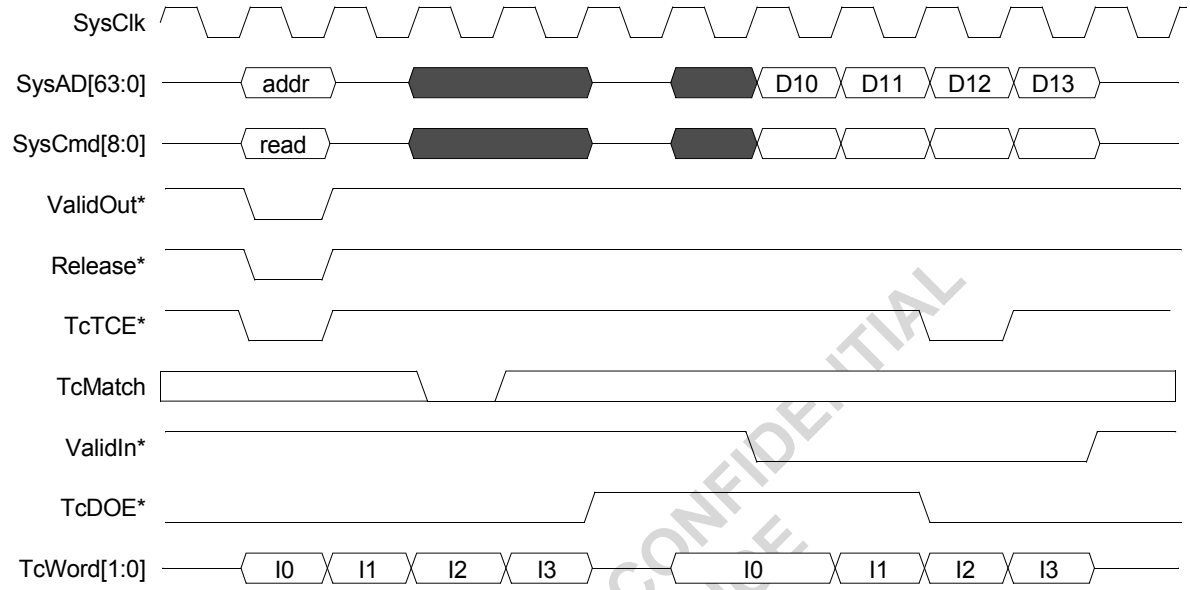
NOTE: Due to PMC-Sierra RM7000 errata, the RdOOO bit in the CPU Configuration register, see Table 92 on page 88, must be set to'0' when using L3 cache.

The GT–64240A also samples TcTCE* signal driven by the CPU and drives TcDOE*. It also drives TcWord[1:0] in case of block read miss.

If a CPU initiates a block read transaction with TcTCE* asserted (indicating L3 read request), and TcMatch is asserted two cycles after issue cycle (indicating L3 hit), the GT–64240A ignores the transaction but keeps TcDOE* asserted. This enables a L3 data RAM drive read data on the SysAD bus. In this case, the TcDOE[1:0] word index is driven by the R7000 L3 cache controller.

In case of a cache miss (TcMatch deasserted two cycles after block read issue cycle), the GT–64240A responds to the transaction. It also deasserts TcDOE* preventing L3 data RAM from driving the bus, and drives TcWord[1:0] for the L3 data RAM to load the data the GT–64240A returns to the CPU. An example of L3 read miss is shown in Figure 8.

Figure 8: R7000 L3 Read Miss Example



NOTE: Figure 8 is a demonstration of the L3 read miss protocol. This figure does not reflect the actual read latency of the GT–64240A.

The GT–64240A supports PMC-Sierra RM7000C SimpleCache mode. In this mode, there is no longer a need for special Tag RAM. The external L3 RAM contains both the tags and the data and the tag compare is performed by the processor and the GT–64240A

NOTE: For further information about the RM7000C Simplified External Cache mode, see the RM7000C spec.

If the CPU Configuration register's SimpleCache bit [24] is set to '1' (see Table 92 on page 88), the GT–64240A performs tag compare internally. The TcMatch signal is no longer used.

NOTE: The GT–64240A supports Simplified External Cache mode, only when interfacing fast write L3 SRAM. It does not support late write SRAM (see RM7000C specification for more information).

The RM7000C does not support non-pedant reads (split reads) in Simplified External Cache mode (see RM7000C specification for more information)

4.9 Multi-GT Support

Up to four GT–64240A devices can be connected to the SysAD bus without the need for any glue logic. This capability enables the CPU to interface with multiple PCI busses and adds significant flexibility for system design.

Multiple GT–64240A is enabled through the reset configuration. See Section 24.1 "Pins Sample Configuration" on page 522.

NOTE: A Multi-GT–64240A configuration can also be used for the CPU to interface GT–64240A device(s) and other slaves on the SysAD bus, as long as these slaves follow the SysAD bus rules.

4.9.1 Hardware Connections

In multi-GT–64240A configuration, ValidIn*, PAck*, and TcDOE* signals function are sustained tri-state outputs requiring 4.7 KOhm pull-up resistors.

All ValidIn* outputs from the GT–64240A devices must be tied together to drive the CPU ValidIn* input.

All PAck* outputs from the GT–64240A devices must be tied together to drive the CPU PAck* input.

All TcDOE* outputs from the GT–64240A devices must be tied together to drive L3 TcDOE* input.

ValidIn* and TcDOE* are only driven by the target GT–64240A. After last the ValidIn* cycle, the GT–64240A drives it HIGH for another cycle and then tri-states it. This also applies to PAck* and TcDOE*.

NOTE: In multi-GT mode, RspSwap* is NC.

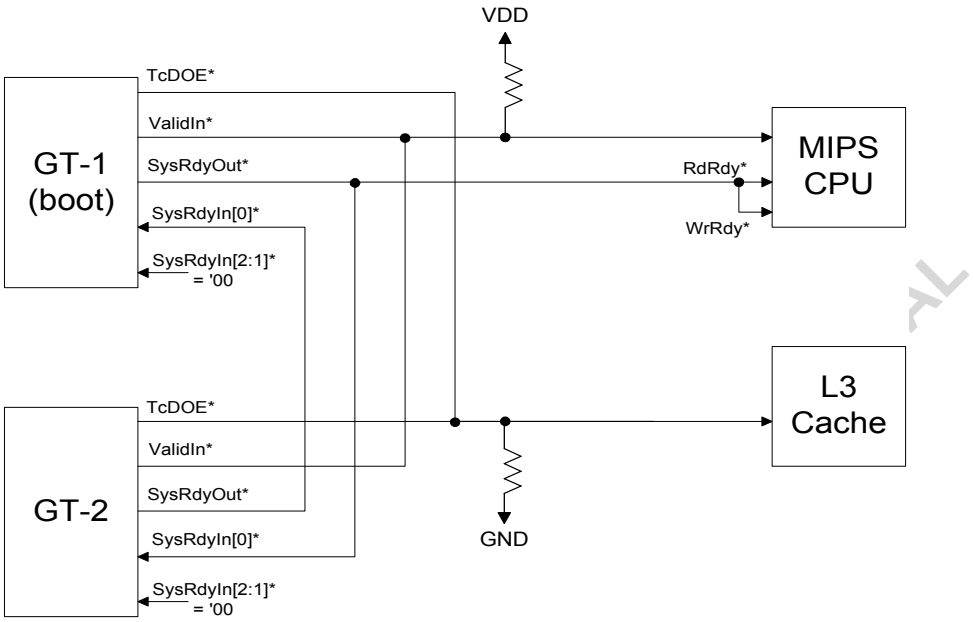
There is a new input signal related to the multi-GT–64240A configuration - SysRdyIn[2:0]*. This signal functions differently in the boot GT–64240A device than the other GT–64240A devices. The CPU RdRdy* and WrRdy* inputs are connected only to SysRdyOut* of the boot GT–64240A device. SysRdyOut* of the other GT–64240A devices are connected to SysRdyIn[2:0] of the boot device. They are internally ORed together in the boot device to generate a combined SysRdyOut* signal to the CPU.

NOTE: In multi-GT mode, SysRdyOut* signal deasserts earlier than in non multi-GT–64240A configurations. To compensate on the sampling stage this signal passes in the boot GT–64240A device.

SysRdyIn[0]* of all the GT–64240A devices, except for the boot device, are connected to the boot device SysRdyOut* (which is also the CPU RdRdy* and WrRdy*) and are used as a qualifier to the CPU issue cycle.

An example of hardware connection of two GT–64240A devices is shown in Figure 9.

Figure 9: Multi-GT–64240A Hardware Connections to the MIPS CPU Bus



In case of a bad CPU read address, that misses all address windows, no device will respond and the system might hang. By setting the NoMatchCntEn bit in the CPU Configuration Register to '1', the boot GT–64240A responds after a timeout period defined in NoMatchCnt field and completes the transaction, see Table 92 on page 88.

NOTES: In a multi-GT–64240A configuration, the GT–64240A cannot detect an address mismatch of write transactions. Also, it does not support read address mismatches if the R7000 split reads are enabled and the L3 cache is present.

In a multi-GT–64240A configuration, the NoMatch counter is applicable only to the boot GT device (the one with Multi-GT ID of '11). If the boot ROM is connected to a slave device other than GT–64240A, the system might hang in case of address mismatch. To avoid a system hang, the non-GT–64240A slave device must have some address mismatch protection mechanism.

4.9.2 Multi-GT Mode Enabled

In multi-GT mode, each GT–64240A device has a two bit ID. This ID distinguishes between the devices. Each device responds to a transaction address that matches it's ID, as shown in Table 29.

Table 29: Multi-GT ID Encoding

Pin	Configuration Function
ID	Multi-GT–64240A Address ID
00	GT–64240A responds to SysAD[26:25]='00'
01	GT–64240A responds to SysAD[26:25]]='01'

Table 29: Multi-GT ID Encoding (Continued)

Pin	Configuration Function
10	GT–64240A responds to SysAD[26:25]='10'
11	GT–64240A responds to SysAD[26:25]='11' **NOTE:** The boot GT–64240A ID must be programmed to '11'.

If the GT–64240A is configured to multi-GT mode during reset, the MultiGTDec bit in the CPU Configuration register is SET, indicating that the CPU Interface address decoding is reduced to:

1. If SysAD[26:25] == ID AND it's a WRITE, the access is directed to the internal space of the CPU Interface registers with bits[11:0] defining the specific register offset.
2. If SysAD[26:25] == ID AND it's a READ AND SysAD[27] == 0, the access is directed to the internal space of the CPU Interface registers with bits[11:0] defining the specific register offset.
3. If SysAD[26:25] == ID AND it's a READ AND SysAD[27] == 1, the access is directed to BootCS*.

NOTE: Since 0x0.1FC0.0000 implies SysAD[26:25] == 3, the GT–64240A holding the boot device must be strapped to ID = 3.

4. When the MultiGTDec bit is CLEARED, the CPU Interface resumes normal address decoding.

4.9.3 Initializing a Multi-GT–64240A System

The following procedure is recommended to initialize a system with two GT–64240As attached to the same CPU.

NOTE: For this example, the two GT–64240As are called GT-1 and GT-2, GT-1 ID is '11' (boot GT–64240A) and GT-2 ID is '00'.

1. Access GT-1's BootROM and reconfigure GT-2's CPU Interface Address Space registers. After reset, the processor executes from the BootROM on GT-1 because the address on SysAD is 0x0.1FCx.xxxx where SysAD[27:25] = '111' and it's a read cycle. Registers on GT-1 are accessible via address SysAD[26:25]=11, [11:0]=offset]. Registers on GT-2 are accessible via address {SysAD[26:25]=00, [11:0]=offset].
2. Access GT-1's BootROM and reconfigure GT-1's CPU Interface address space registers. Also, reconfigure the Internal Space Address Decode register so that later, once the multi-GT mode is disabled, it is possible to differ between internal accesses to GT-1 or GT-2.
3. Lower GT-2 BootCS* high decode register BELOW 0x0.1FCx.xxxx (i.e. 0x0.1FBx.xxxx). This causes GT-2 to ignore accesses to 0x0.1FCx.xxxx once taken out of multi-GT mode. Also, each GT–64240A address mapping must be unique. There must not be any address decoding range in one device that overlaps any part of the other device address mapping.
4. Clear GT-2 MultiGTDec bit.
5. Clear GT-1 MultiGTDec bit.

Now both GT–64240As resume NORMAL operation with USUAL address decoding.

NOTE: In the presence of multiple GT–64240A devices, each devices' CPU Configuration register must be programed to the same value.

4.10 Parity Support

The GT–64240A supports even data parity driven on the SysADC bus.

It samples data parity on write transactions and drives parity on reads. It also propagates bad parity between the CPU bus and the other interfaces (SDRAM, PCI). In case of bad parity detection, it also asserts an interrupt.

For full description of parity support, see Section 6. “Address and Data Integrity” on page 133.

NOTE:

NOTE: When running MPX bus mode, the internal arbiter must be used.

4.11 CPU Endian Support

The CPU bus endianess is determined via the CPU Configuration register’s Endianess bit, see Table 92 on page 88. The GT–64240A provides the capability to swap the byte order of data that enables endianess conversion between the CPU interface and some other interfaces.

The endianess convention of the local memory attached to the GT–64240A (SDRAM, devices) is assumed to be the same one as the CPU. This means data transfered to/from the local memory is NEVER swapped.

The internal registers of the GT–64240A are always programmed in Little Endian. On a CPU access to the internal registers, if the CPU bus is configured to Big Endian because the CPU Configuration register’s Endianess bit is set to ’0’, data is swapped.

Data swapping on a CPU access to the PCI is controlled via PCISwap bits of each PCI Low Address register. This configurable setting allows a CPU access to PCI agents using a different endianess convention.

For software compatibility with the GT-64120/130 devices, the GT–64240A maintains MByteSwap and MWordSwap bits in the PCI Command register, see Table 233 on page 206. If the PCI Command register’s MSwapEn bit is set to ‘1’, the GT–64240A PCI master performs data swapping according to PCISwap bits setting. If set to ‘0’ (default), it works according to MByteSwap and MWordSwap bits setting, as in the GT-64120/130 devices.

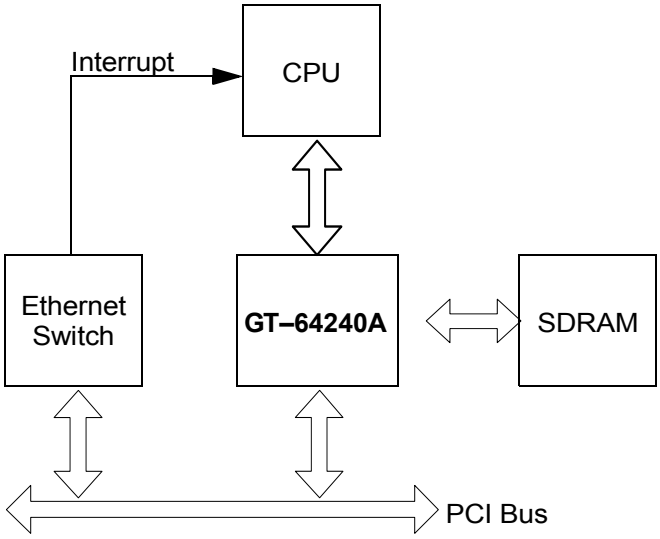
See Section 8.13 “Data Endianess” on page 170 for more information on data swapping.

4.12 CPU Synchronization Barrier

The GT–64240A supports a sync barrier mechanism. This mechanism is a hardware hook to help software synchronize between the CPU and PCI activities. The GT–64240A supports sync barrier in both directions - CPU to PCI and PCI to CPU.

Figure 10 shows an example of a CPU sync barrier application.

Figure 10: CPU Sync Barrier Example



In the example, an ethernet switch sends a packet through the PCI bus to the SDRAM. The ethernet switch then notifies the CPU that it has a packet waiting in SDRAM to handle by asserting CPU interrupt. Since the packet might still reside in GT–64240A PCI slave write buffer rather than SDRAM, the CPU interrupt handler must perform a sync barrier action to make sure the packet is flushed to SDRAM.

The CPU interface treats PCI I/O reads and configuration reads as “synchronization barrier” cycles. These reads receive a response once no posted data remains within the PCI slave write buffer.

NOTE: To disable these sync barrier, set ConfSBDis and IOSBDis bits in CPU Configuration register to 1.

The GT–64240A provides the CPU with a simpler way to perform synchronization with the PCI bus. The CPU issues a read request to the PCI Sync Barrier Virtual register. Once no posted data remains within the addressed PCI interface, the dummy read is complete.

NOTE: Data from this read must be discarded.

As an option, use the CPU sync barrier to invalidate the PCI slave read buffers. If SBInv bit in PCI Slave Control register is set to 1 (default), the slave read buffers are invalidated with each CPU sync barrier.

4.13 Clocks Synchronization

The CPU interface can be driven from the core clock (TClk) or by a separate clock input, not synchronized to TClk. This CPU clocking scheme is determined via reset configuration, see Section 24. “Reset Configuration” on page 522. If driven by the core clock (TClk), the SysClk input pin is not used. If driven by a separate clock input, SysClk frequency must not exceed the TClk frequency.

The CPU interface includes synchronization logic that synchronizes between the SysClk and TClk clock domains. When running the CPU interface with TClk, these synchronizers are bypassed, eliminating the latency penalty of the synchronizers.

4.14 Programing the CPU Configuration Register

The CPU setting of the CPU Configuration register requires special care, since it affects the GT–64240A behavior on consecutive CPU accesses.

To change the register, the following steps are recommended:

1. Read the CPU Configuration register. This guarantees that all previous transactions in the CPU interface pipe are flushed.
2. Only after the CPU interface pipe is flushed, program the register to its new value.
3. Read polling of the register until the new data is being read.

NOTE: CPU Configuration register wakes up with split transactions disabled.It is recommended to change this default in order gain the maximum CPU interface performance.

Setting the CPU Configuration register must be done once. For example, if the CPU interface is configured to support Out of Order read completion, changing the register to not support OOO read completion is fatal.

4.15 CPU Interface Registers

Table 30: CPU Address Decode Register Map

Register	Offset	Page
SCS[0]* Low Decode Address	0x008	page 76
SCS[0]* High Decode Address	0x010	page 77
SCS[1]* Low Decode Address	0x208	page 77
SCS[1]* High Decode Address	0x210	page 77
SCS[2]* Low Decode Address	0x018	page 77
SCS[2]* High Decode Address	0x020	page 77
SCS[3]* Low Decode Address	0x218	page 77
SCS[3]* High Decode Address	0x220	page 78
CS[0]* Low Decode Address	0x028	page 78
CS[0]* High Decode Address	0x030	page 78
CS[1]* Low Decode Address	0x228	page 78
CS[1]* High Decode Address	0x230	page 78
CS[2]* Low Decode Address	0x248	page 78
CS[2]* High Decode Address	0x250	page 79
CS[3]* Low Decode Address	0x038	page 79
CS[3]* High Decode Address	0x040	page 79

Table 30: CPU Address Decode Register Map (Continued)

Register	Offset	Page
Boot CS* Low Decode Address	0x238	page 79
Boot CS* High Decode Address	0x240	page 79
PCI_0 I/O Low Decode Address	0x048	page 79
PCI_0 I/O High Decode Address	0x050	page 80
PCI_0 Memory 0 Low Decode Address	0x058	page 80
PCI_0 Memory 0 High Decode Address	0x060	page 81
PCI_0 Memory 1 Low Decode Address	0x080	page 81
PCI_0 Memory 1 High Decode Address	0x088	page 81
PCI_0 Memory 2 Low Decode Address	0x258	page 81
PCI_0 Memory 2 High Decode Address	0x260	page 81
PCI_0 Memory 3 Low Decode Address	0x280	page 82
PCI_0 Memory 3 High Decode Address	0x288	page 82
PCI_1 I/O Low Decode Address	0x090	page 82
PCI_1 I/O High Decode Address	0x098	page 82
PCI_1 Memory 0 Low Decode Address	0x0a0	page 82
PCI_1 Memory 0 High Decode Address	0x0a8	page 83
PCI_1 Memory 1 Low Decode Address	0x0b0	page 83
PCI_1 Memory 1 High Decode Address	0x0b8	page 83
PCI_1 Memory 2 Low Decode Address	0x2a0	page 83
PCI_1 Memory 2 High Decode Address	0x2a8	page 84
PCI_1 Memory 3 Low Decode Address	0x2b0	page 84
PCI_1 Memory 3 High Decode Address	0x2b8	page 84
Internal Space Decode Address	0x068	page 84
PCI_0 I/O Address Remap	0x0f0	page 85
PCI_0 Memory 0 Remap (Low)	0x0f8	page 86
PCI_0 Memory 0 Remap (High)	0x320	page 86
PCI_0 Memory 1 Remap (Low)	0x100	page 86
PCI_0 Memory 1 Remap (High)	0x328	page 86
PCI_0 Memory 2 Remap (Low)	0x2f8	page 86
PCI_0 Memory 2 Remap (High)	0x330	page 86

Table 30: CPU Address Decode Register Map (Continued)

Register	Offset	Page
PCI_0 Memory 3 Remap (Low)	0x300	page 87
PCI_0 Memory 3 Remap (High)	0x338	page 87
PCI_1 I/O Remap	0x108	page 87
PCI_1 Memory 0 Remap (Low)	0x110	page 87
PCI_1 Memory 0 Remap (High)	0x340	page 87
PCI_1 Memory 1 Remap (Low)	0x118	page 87
PCI_1 Memory 1 Remap (High)	0x348	page 88
PCI_1 Memory 2 Remap (Low)	0x310	page 88
PCI_1 Memory 2 Remap (High)	0x350	page 88
PCI_1 Memory 3 Remap (Low)	0x318	page 88
PCI_1 Memory 3 Remap (High)	0x358	page 88

Table 31: CPU Control Register Map

Register	Offset	Page
CPU Configuration	0x000	page 88
CPU Mode	0x120	page 91
CPU Read Response Crossbar Control (Low)	0x170	page 92
CPU Read Response Crossbar Control (High)	0x178	page 92

Table 32: CPU Sync Barrier Register Map

Register	Offset	Page
PCI_0 Sync Barrier Virtual Register	0x0c0	page 93
PCI_1 Sync Barrier Virtual Register	0x0c8	page 93

Table 33: CPU Access Protection Register Map

Register	Offset	Page
Protect Low Address 0	0x180	page 93
Protect High Address 0	0x188	page 94
Protect Low Address 1	0x190	page 94

Table 33: CPU Access Protection Register Map (Continued)

Register	Offset	Page
Protect High Address 1	0x198	page 94
Protect Low Address 2	0x1a0	page 94
Protect High Address 2	0x1a8	page 95
Protect Low Address 3	0x1b0	page 95
Protect High Address 3	0x1b8	page 96
Protect Low Address 4	0x1c0	page 96
Protect High Address 4	0x1c8	page 96
Protect Low Address 5	0x1d0	page 96
Protect High Address 5	0x1d8	page 97
Protect Low Address 6	0x1e0	page 97
Protect High Address 6	0x1e8	page 98
Protect Low Address 7	0x1f0	page 98
Protect High Address 7	0x1f8	page 98

Table 34: CPU Error Report Register Map

Register	Offset	Page
CPU Error Address (Low)	0x070	page 99
CPU Error Address (High)	0x078	page 99
CPU Error Data (Low)	0x128	page 99
CPU Error Data (High)	0x130	page 100
CPU Error Parity	0x138	page 100
CPU Error Cause	0x140	page 100
CPU Error Mask	0x148	page 101

4.15.1 CPU Address Decode Registers

Table 35: SCS[0]* Low Decode Address, Offset: 0x008

Bits	Field Name	Function	Initial Value
15:0	LowAddr	SCS[0] Base Address	0x0
31:16	Reserved	Must be 0.	0x0

Table 36: SCS[0]* High Decode Address, Offset: 0x010

Bits	Field Name	Function	Initial Value
11:0	HighAddr	SCS[0] Top Address	0x007
31:12	Reserved	Reserved.	0x0

Table 37: SCS[1]* Low Decode Address, Offset: 0x208

Bits	Field Name	Function	Initial Value
15:0	LowAddr	SCS[1] Base Address	0x0008
31:16	Reserved	Reserved.	0x0

Table 38: SCS[1]* High Decode Address, Offset: 0x210

Bits	Field Name	Function	Initial Value
11:0	HighAddr	SCS[1] Top Address	0x00f
31:12	Reserved	Reserved.	0x0

Table 39: SCS[2]* Low Decode Address, Offset: 0x018

Bits	Field Name	Function	Initial Value
15:0	LowAddr	SCS[2] Base Address	0x0010
31:16	Reserved	Reserved.	0x0

Table 40: SCS[2]* High Decode Address, Offset: 0x020

Bits	Field Name	Function	Initial Value
11:0	HighAddr	SCS[2] Top Address	0x017
31:12	Reserved	Reserved.	0x0

Table 41: SCS[3]* Low Decode Address, Offset: 0x218

Bits	Field Name	Function	Initial Value
15:0	LowAddr	SCS[3] Base Address	0x0018
31:16	Reserved	Reserved.	0x0

Table 42: SCS[3]* High Decode Address, Offset: 0x220

Bits	Field Name	Function	Initial Value
11:0	HighAddr	SCS[3] Top Address	0x01f
31:12	Reserved	Reserved.	0x0

Table 43: CS[0]* Low Decode Address, Offset: 0x028

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CS[0] Base Address	0x01c0
31:16	Reserved	Reserved.	0x0

Table 44: CS[0]* High Decode Address, Offset: 0x030

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CS[0] Top Address	0x1c7
31:12	Reserved	Reserved.	0x0

Table 45: CS[1]* Low Decode Address, Offset: 0x228

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CS[1] Base Address	0x01c8
31:16	Reserved	Reserved.	0x0

Table 46: CS[1]* High Decode Address, Offset: 0x230

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CS[1] Top Address	0x1cf
31:12	Reserved	Reserved.	0x0

Table 47: CS[2]* Low Decode Address, Offset: 0x248

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CS[2] Base Address	0x01d0
31:16	Reserved	Reserved.	0x0

Table 48: CS[2]* High Decode Address, Offset: 0x250

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CS[2] Top Address	0x1df
31:12	Reserved	Reserved.	0x0

Table 49: CS[3]* Low Decode Address, Offset: 0x038

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CS[3] Base Address	0x01f0
31:16	Reserved	Reserved.	0x0

Table 50: CS[3]* High Decode Address, Offset: 0x040

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CS[3] Top Address	0x1f7
31:12	Reserved	Reserved.	0x0

Table 51: BootCS* Low Decode Address, Offset: 0x238

Bits	Field Name	Function	Initial Value
15:0	LowAddr	BootCS Base Address	0x01f8
31:16	Reserved		0x0

Table 52: BootCS* High Decode Address, Offset: 0x240

Bits	Field Name	Function	Initial Value
11:0	HighAddr	BootCS Top Address	0x1ff
31:12	Reserved	Reserved.	0x0

Table 53: PCI_0 I/O Low Decode Address, Offset: 0x048

Bits	Field Name	Function	Initial Value
15:0	LowAddr	PCI_0 I/O Space Base Address	0x0100
23:16	Reserved	Reserved.	0x0

Table 53: PCI_0 I/O Low Decode Address, Offset: 0x048

Bits	Field Name	Function	Initial Value
26:24	PCISwap	PCI Master Data Swap Control 000 - Byte Swap 001 - No swapping 010 - Both byte and word swap 011 - Word swap 1xx - Reserved	0x1
31:27	Reserved	Reserved.	0x0

Table 54: PCI_0 I/O High Decode Address, Offset: 0x050

Bits	Field Name	Function	Initial Value
11:0	HighAddr	PCI_0 I/O Space Top Address	0x11f
31:12	Reserved	Reserved.	0x0

Table 55: PCI_0 Memory 0 Low Decode Address, Offset: 0x058

Bits	Field Name	Function	Initial Value
15:0	LowAddr	PCI_0 Memory 0 Base Address	0x0120
23:16	Reserved	Reserved.	0x0
26:24	PCISwap	PCI master data swap control 000 - Byte Swap 001 - No swapping 010 - Both byte and word swap 011 - Word swap 1xx - Reserved	0x1
27	PCIReq64[1]	PCI master REQ64* policy 0 - Asserts REQ64* only when transaction is longer than 64-bits. 1 - Always assert REQ64*.	0x0
31:28	Reserved	Reserved.	0x0

1. Relevant only when configured to 64-bit PCI bus

Table 56: PCI_0 Memory 0 High Decode Address, Offset: 0x060

Bits	Field Name	Function	Initial Value
11:0	HighAddr	PCI_0 Memory 0 Top Address	0x13f
31:10	Reserved	Reserved.	0x0

Table 57: PCI_0 Memory 1 Low Decode Address, Offset: 0x080

Bits	Field Name	Function	Initial Value
15:0	LowAddr	PCI_0 Memory 1 Base Address	0x0f20
23:16	Reserved	Reserved.	0x0
26:24	PCISwap	Same as PCI_0 Memory 0 Low Decode Address.	0x1
27	PCIReq64	Same as PCI_0 Memory 0 Low Decode Address.	0x0
31:28	Reserved	Reserved.	0x0

Table 58: PCI_0 Memory 1 High Decode Address, Offset: 0x088

Bits	Field Name	Function	Initial Value
11:0	HighAddr	PCI_0 Memory 1 Top Address	0xf3f
31:12	Reserved	Reserved.	0x0

Table 59: PCI_0 Memory 2 Low Decode Address, Offset: 0x258

Bits	Field Name	Function	Initial Value
15:0	LowAddr	PCI_0 Memory 2 Base Address	0x0f40
23:16	Reserved	Reserved.	0x0
26:24	PCISwap	Same as PCI_0 Memory 0 Low Decode Address.	0x1
27	PCIReq64	Same as PCI_0 Memory 0 Low Decode Address.	0x0
31:28	Reserved	Reserved.	0x0

Table 60: PCI_0 Memory 2 High Decode Address, Offset: 0x260

Bits	Field Name	Function	Initial Value
11:0	HighAddr	PCI_0 Memory 2 Top Address	0xf5f
31:12	Reserved	Reserved.	0x0

Table 61: PCI_0 Memory 3 Low Decode Address, Offset: 0x280

Bits	Field Name	Function	Initial Value
15:0	LowAddr	PCI_0 Memory 3 Base Address	0x0f60
23:16	Reserved	Reserved.	0x0
26:24	PCISwap	Same as PCI_0 Memory 0 Low Decode Address.	0x1
27	PCIReq64	Same as PCI_0 Memory 0 Low Decode Address.	0x0
31:28	Reserved	Reserved.	0x0

Table 62: PCI_0 Memory 3 High Decode Address, Offset: 0x288

Bits	Field Name	Function	Initial Value
11:0	HighAddr	PCI_0 Memory 3 Top Address	0xf7f
31:12	Reserved	Reserved.	0x0

Table 63: PCI_1 I/O Low Decode Address, Offset: 0x090

Bits	Field Name	Function	Initial Value
:0	LowAddr	PCI_1 I/O Space Base Address	0x0200
26:	Reserved	Reserved.	0x0
27	PCIReq64	Same as PCI_0 Memory 0 Low Decode Address.	0x0
31:28	Reserved	Reserved.	0x0

Table 64: PCI_1 I/O High Decode Address, Offset: 0x098

Bits	Field Name	Function	Initial Value
11:0	HighAddr	PCI_1 I/O Space Top Address	0x21f
31:12	Reserved	Reserved.	0x0

Table 65: PCI_1 Memory 0 Low Decode Address, Offset: 0x0a0

Bits	Field Name	Function	Initial Value
:0	LowAddr	PCI_1 Memory 0 Base Address	0x0220
23:	Reserved	Reserved.	0x0
26:24	PCISwap	Same as PCI_0 Memory 0 Low Decode Address.	0x1

Table 65: PCI_1 Memory 0 Low Decode Address, Offset: 0x0a0 (Continued)

Bits	Field Name	Function	Initial Value
27	PCIReq64	Same as PCI_0 Memory 0 Low Decode Address.	0x0
31:28	Reserved	Reserved.	0x0

Table 66: PCI_1 Memory 0 High Decode Address, Offset: 0x0a8

Bits	Field Name	Function	Initial Value
11:0	HighAddr	PCI_1 Memory 0 Top Address	0x23f
31:12	Reserved	Reserved.	0x0

Table 67: PCI_1 Memory 1 Low Decode Address, Offset: 0x0b0

Bits	Field Name	Function	Initial Value
:0	LowAddr	PCI_1 Memory 1 Base Address	0x0240
23:	Reserved	Reserved.	0x0
26:24	PCISwap	Same as PCI_0 Memory 0 Low Decode Address.	0x1
27	PCIReq64	Same as PCI_0 Memory 0 Low Decode Address.	0x0
31:28	Reserved	Reserved.	0x0

Table 68: PCI_1 Memory 1 High Decode Address, Offset: 0x0b8

Bits	Field Name	Function	Initial Value
11:0	HighAddr	PCI_1 Memory 1 Top Address	0x25f
31:12	Reserved	Reserved.	0x0

Table 69: PCI_1 Memory 2 Low Decode Address, Offset: 0x2a0

Bits	Field Name	Function	Initial Value
:0	LowAddr	PCI_1 Memory 2 Base Address	0x0260
23:	Reserved	Reserved.	0x0
26:24	PCISwap	Same as PCI_0 Memory 0 Low Decode Address.	0x1
27	PCIReq64	Same as PCI_0 Memory 0 Low Decode Address.	0x0
31:28	Reserved	Reserved.	0x0

Table 70: PCI_1 Memory 2 High Decode Address, Offset: 0x2a8

Bits	Field Name	Function	Initial Value
11:0	HighAddr	PCI_1 Memory 2 Top Address	0x27f
31:12	Reserved	Reserved.	0x0

Table 71: PCI_1 Memory 3 Low Decode Address, Offset: 0x2b0

Bits	Field Name	Function	Initial Value
:0	LowAddr	PCI_1 Memory 3 Base Address	0x0280
23:	Reserved	Reserved.	0x0
26:24	PCISwap	Same as PCI_0 Memory 0 Low Decode Address.	0x1
27	PCIReq64	Same as PCI_0 Memory 0 Low Decode Address.	0x0
31:28	Reserved	Reserved.	0x0

Table 72: PCI_1 Memory 3 High Decode Address, Offset: 0x2b8

Bits	Field Name	Function	Initial Value
11:0	HighAddr	PCI_1 Memory 3 Top Address	0x29f
31:12	Reserved	Reserved.	0x0

Table 73: Internal Space Decode, Offset: 0x068

Bits	Field Name	Function	Initial Value
11:0	IntDecode	GT–64240A Internal Space Base Address **NOTE:** The initial value is dependent on the reset strapping.	0x0140 or 0xf100
23:15	Reserved	Reserved.	0x0
26:24	PCISwap	Same as PCI_0 Memory 0 Low Decode Address. Relevant only for PCI master configuration transactions on the PCI bus. **NOTE:** Reserved for Galileo Technology usage.	0x1
31:27	Reserved	Reserved.	0x0

Table 74: PCI_0 I/O Address Remap, Offset: 0x0f0

Bits	Field Name	Function	Initial Value
11:0	Remap	PCI_0 I/O Space Address Remap	0x100
31:12	Reserved	Reserved.	0x0

Table 75: PCI_0 Memory 0 Address Remap (Low), Offset: 0x0f8

Bits	Field Name	Function	Initial Value
11:0	Remap	PCI_0 Memory 0 Address Remap (low 32 bits)	0x120
31:12	Reserved	Reserved.	0x0

Table 76: PCI_0 Memory 0 Address Remap (High), Offset: 0x320

Bits	Field Name	Function	Initial Value
31:0	Remap	PCI_0 Memory 0 Address Remap (high 32 bits)	0x0

Table 77: PCI_0 Memory 1 Address Remap (Low), Offset: 0x100

Bits	Field Name	Function	Initial Value
11:0	Remap	PCI_0 Memory 1 Address Remap (low 32 bits)	0xf20
31:12	Reserved	Reserved.	0x0

Table 78: PCI_0 Memory 1 Address Remap (High), Offset: 0x328

Bits	Field Name	Function	Initial Value
31:0	Remap	PCI_0 Memory 1 Address Remap (high 32 bits)	0x0

Table 79: PCI_0 Memory 2 Address Remap (Low), Offset: 0x2f8

Bits	Field Name	Function	Initial Value
11:0	Remap	PCI_0 Memory 0 Address Remap (low 32 bits)	0xf40
31:12	Reserved	Reserved.	0x0

Table 80: PCI_0 Memory 2 Address Remap (High), Offset: 0x330

Bits	Field Name	Function	Initial Value
31:0	Remap	PCI_0 Memory 2 Address Remap (high 32 bits)	0x0

Table 81: PCI_0 Memory 3 Address Remap (Low), Offset: 0x300

Bits	Field Name	Function	Initial Value
11:0	Remap	PCI_0 Memory 1 Address Remap (low 32 bits)	0xf60
31:12	Reserved	Reserved.	0x0

Table 82: PCI_0 Memory 3 Address Remap (High), Offset: 0x338

Bits	Field Name	Function	Initial Value
31:0	Remap	PCI_0 Memory 3 Address Remap (high 32 bits)	0x0

Table 83: PCI_1 I/O Address Remap, Offset: 0x108

Bits	Field Name	Function	Initial Value
11:0	Remap	PCI_1 I/O Space Address Remap	0x200
31:12	Reserved	Reserved.	0x0

Table 84: PCI_1 Memory 0 Address Remap (Low), Offset: 0x110

Bits	Field Name	Function	Initial Value
11:0	Remap	PCI_1 Memory 0 Address Remap (low 32 bits)	0x220
31:12	Reserved	Reserved.	0x0

Table 85: PCI_1 Memory 0 Address Remap (High), Offset: 0x340

Bits	Field Name	Function	Initial Value
31:0	Remap	PCI_1 Memory 0 Address Remap (high 32 bits)	0x0

Table 86: PCI_1 Memory 1 Address Remap (Low), Offset: 0x118

Bits	Field Name	Function	Initial Value
11:0	Remap	PCI_1 Memory 1 Address Remap (low 32 bits)	0x240
31:12	Reserved	Reserved.	0x0

Table 87: PCI_1 Memory 1 Address Remap (High), Offset: 0x348

Bits	Field Name	Function	Initial Value
31:0	Remap	PCI_1 Memory 1 Address Remap (high 32 bits)	0x0

Table 88: PCI_1 Memory 2 Address Remap (Low), Offset: 0x310

Bits	Field Name	Function	Initial Value
11:0	Remap	PCI_1 Memory 2 Address Remap (low 32 bits)	0x260
31:12	Reserved	Reserved.	0x0

Table 89: PCI_1 Memory 2 Address Remap (High), Offset: 0x350

Bits	Field Name	Function	Initial Value
31:0	Remap	PCI_1 Memory 2 Address Remap (high 32 bits)	0x0

Table 90: PCI_1 Memory 3 Address Remap (Low), Offset: 0x318

Bits	Field Name	Function	Initial Value
11:0	Remap	PCI_1 Memory 3 Address Remap (low 32 bits)	0x280
31:12	Reserved	Reserved.	0x0

Table 91: PCI_1 Memory 3 Address Remap (High), Offset: 0x358

Bits	Field Name	Function	Initial Value
31:0	Remap	PCI_1 Memory 3 Address Remap (high 32 bits)	0x0

4.15.2 CPU Control Registers

Table 92: CPU Configuration, Offset: 0x000

Bits	Field Name	Function	Initial Value
7:0	NoMatchCnt	CPU Address Miss Counter.	0xff

Table 92: CPU Configuration, Offset: 0x000 (Continued)

Bits	Field Name	Function	Initial Value
8	NoMatchCntEn	CPU Address Miss Counter Enable **NOTE:** Relevant only if multi-GT is enabled. 0 - Disabled 1 - Enabled.	0x0
9	NoMatchCntExt	CPU Address Miss Counter MSB	0x0
11:10	Reserved	Reserved.	0x0
12	Endianess	CPU bus byte Orientation 0 - Big Endian 1 - Little Endian	AD[4] sampled at reset.
13	SplitRd	Split Read Transaction Support 0 - Not Supported **NOTE:** PReq* input is not sampled, PAck* never asserted. 1 - Supported	0x0
14	R7KL3	R7000 (R5000) Third (Second) Level Cache Present 0 - R7KL3 not present **NOTE:** TcMatch input is not sampled. 1 - R7KL3 present	0x0
15	Reserved	Reserved.	0x1
16	RdOOO	Read Out of Order Completion 0 - Not Supported. Data is always returned in order. **NOTE:** RspSwap is never asserted. 1 - Supported **NOTE:** When configured for multi-GT mode, RdOOO must be set to '0'.	0x0
17	Stop Retry	**NOTE:** Relevant only if PCI Retry is enabled. 0 - Keep Retry all PCI transactions targeted to GT–64240A. 1 - Stop PCI transactions retry.	0x0
18	MultiGTDec	Multi-GT Address Decode 0 - Normal address decoding 1 - Multi-GT address decoding	Reset Initialization.
19	SysADCValid	CPU SysADC Connection 0 - Not connected The CPU write parity is not checked. The GT–64240A drives SysCmd[4] to 1 during reads. This indicates to the CPU not to check read parity. 1 - Connected	0x0

Table 92: CPU Configuration, Offset: 0x000 (Continued)

Bits	Field Name	Function	Initial Value
21:20	Reserved	Reserved.	0x0
22	PErrProp	Parity Error Propagation 0 - The GT–64240A drives good parity on SysADC during CPU reads 1 - The GT–64240A drives bad parity on SysADC in case the read response from the target interface comes with erroneous data indication (e.g. ECC error from SDRAM interface).	0x0
23	FastClk	The number of pipe stages in the CPU interface. 0 - Two pipe stages 1 - Additional pipe stage	0x1
24	SimpleCache	0 - Traditional R7000 L3 1 - RM7000C Simplified External Cache mode	0x0
26:25	Reserved	Reserved.	0x1
27	RemapWrDis	Address Remap Registers Write Control 0 - Write to the Low Address decode register. Results are also in written to the corresponding Remap register. 1 - Write to Low Address decode register. This has no affect on the corresponding Remap register	0x0
28	ConfSBDis	Configuration Read Sync Barrier Disable 0 - Sync Barrier enabled 1 - Sync Barrier disabled	0x1
29	IOSBDis	I/O Read Sync Barrier Disable 0 - Sync Barrier enabled 1 - Sync Barrier disabled	0x1
30	ClkSync	Clocks Synchronization 0 - The CPU interface is running with SysClk, which is asynchronous to TClk. 1 - The CPU interface is running with TClk.	AD[5] sampled at reset.
31	Reserved	Reserved.	0x0

Table 93: CPU Mode, Offset: 0x120

Bits	Field Name	Function	Initial Value
1:0	MultiGTID	Multi-GT ID Represents the ID to which the GT–64240A responds to during a multi-GT address decoding period. Set during reset initialization. Read only.	AD[11:10] sampled at reset.
2	MultiGT	Set during the reset initialization. Read only. 0 - Single GT configuration 1 - Multi-GT configuration	AD[9] sampled at reset.
3	RetryEn	Set during reset initialization. Read Only. 0 - Don't Retry PCI transactions 1 - Retry PCI transactions	AD[16] sampled at reset.
7:4	CPUType	Read Only (reset and bonding configuration). 0x0 - 64-bit MIPS CPU 0x1 - 0xf - Reserved	AD[7:6] sampled at reset.
31:8	Reserved	Reserved.	0x0

Table 94: CPU Read Response Crossbar Control (Low), Offset: 0x170

Bits	Field Name	Function	Initial Value
3:0	Arb0	Slice 0 of CPU Slave “pizza” Arbiter 0x0 - SDRAM read data 0x1 - Device read data 0x2 - NULL 0x3 - PCI_0 read data 0x4 - PCI_1 read data 0x5 - Comm unit internal registers read data 0x6 - IDMA 0/1/2/3 internal registers read data 0x7 - IDMA 4/5/6/7 internal registers read data 0x8 - 0xf - Reserved	0x0
7:4	Arb1	Slice 1 of CPU Slave “pizza” Arbiter	0x1
11:8	Arb2	Slice 2 of CPU Slave “pizza” Arbiter	0x3
15:12	Arb3	Slice 3 of CPU Slave “pizza” Arbiter	0x4
19:16	Arb4	Slice 4 of CPU Slave “pizza” Arbiter	0x5
23:20	Arb5	Slice 5 of CPU Slave “pizza” Arbiter	0x6
27:24	Arb6	Slice 6 of CPU Slave “pizza” Arbiter	0x7
31:28	Arb7	Slice 7 of CPU Slave “pizza” Arbiter	0x2

Table 95: CPU Read Response Crossbar Control (High), Offset: 0x178

Bits	Field Name	Function	Initial Value
3:0	Arb8	Slice 8 of CPU Slave “pizza” Arbiter	0x0
7:4	Arb9	Slice 9 of CPU Slave “pizza” Arbiter	0x1
11:8	Arb10	Slice 10 of CPU Slave “pizza” Arbiter	0x3
15:12	Arb11	Slice 11 of CPU Slave “pizza” Arbiter	0x4
19:16	Arb12	Slice 12 of CPU Slave “pizza” Arbiter	0x5
23:20	Arb13	Slice 13 of CPU Slave “pizza” Arbiter	0x6
27:24	Arb14	Slice 14 of CPU Slave “pizza” Arbiter	0x7
31:28	Arb15	Slice 15 of CPU Slave “pizza” Arbiter	0x2

4.15.3 CPU Sync Barrier Registers

Table 96: PCI_0 Sync Barrier Virtual Register, Offset: 0x0c0

Bits	Field Name	Function	Initial Value
31:0	SyncBarrier_0	A CPU read from this register creates a synchronization barrier cycle. **NOTE:** The read data is random and should be ignored.	0x0

Table 97: PCI_1 Sync Barrier Virtual Register, Offset: 0x0c8

Bits	Field Name	Function	Initial Value
31:0	SyncBarrier_1	A CPU read from this register creates a synchronization barrier cycle. **NOTE:** The read data is random and should be ignored.	0x0

4.15.4 CPU Access Protect Registers

Table 98: CPU Protect Address 0 (Low), Offset: 0x180

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CPU Protect Region 0 Base Address Corresponds to address bits[35:20].	0xfff
16	AccProtect	CPU Access Protect. 0 - Access allowed. 1 - Access forbidden.	0x0
17	WrProtect	CPU Write Protect 0 - Write allowed. 1 - Write forbidden.	0x0
18	CacheProtect	CPU caching protect 0 - Caching (block read) is allowed. 1 - Caching is forbidden.	0x0
31:19	Reserved	Reserved.	0x0

Table 99: CPU Protect Address 0 (High), Offset: 0x188

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CPU Protect Region 0 Top Address Corresponds to address bits[31:20]	0x0
31:12	Reserved	Reserved.	0x0

Table 100: CPU Protect Address 1 (Low), Offset: 0x190

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CPU Protect Region 1 Base Address Corresponds to address bits[35:20]	0xfff
16	AccProtect	CPU Access Protect. 0 - Access allowed. 1 - Access forbidden.	0x0
17	WrProtect	CPU Write Protect 0 - Write allowed. 1 - Write forbidden.	0x0
18	CacheProtect	CPU Caching Protect 0 - Caching (block read) allowed. 1 - Caching forbidden.	0x0
31:19	Reserved	Reserved.	0x0

Table 101: CPU Protect Address 1 (High), Offset: 0x198

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CPU Protect region 1 Top Address Corresponds to address bits[31:20].	0x0
31:12	Reserved	Reserved.	0x0

Table 102: CPU Protect Address 2 (Low), Offset: 0x1a0

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CPU Protect Region 2 Base Address Corresponds to address bits[35:20]	0xfff
16	AccProtect	CPU Access Protect 0 - Access allowed. 1 - Access forbidden.	0x0

Table 102: CPU Protect Address 2 (Low), Offset: 0x1a0 (Continued)

Bits	Field Name	Function	Initial Value
17	WrProtect	CPU Write Protect 0 - Write allowed. 1 - Write forbidden.	0x0
18	CacheProtect	CPU Caching Protect 0 - Caching (block read) allowed. 1 - Caching is forbidden.	0x0
31:19	Reserved	Reserved.	0x0

Table 103: CPU Protect Address 2 (High), Offset: 0x1a8

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CPU Protect Region 2 Top Address Corresponds to address bits[31:20].	0x0
31:12	Reserved	Reserved.	0x0

Table 104: CPU Protect Address 3 (Low), Offset: 0x1b0

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CPU Protect Region 3 Base Address Corresponds to address bits[35:20].	0xfff
16	AccProtect	CPU Access Protect 0 - Access allowed. 1 - Access forbidden.	0x0
17	WrProtect	CPU Write Protect 0 - Write allowed 1 - Write forbidden	0x0
18	CacheProtect	CPU Caching Protect 0 - Caching (block read) allowed. 1 - Caching forbidden.	0x0
31:19	Reserved	Reserved.	0x0

Table 105: CPU Protect Address 3 (High), Offset: 0x1b8

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CPU Protect Region 3 Top Address Corresponds to address bits[31:20].	0x0
31:12	Reserved	Reserved.	0x0

Table 106: CPU Protect Address 4 (Low), Offset: 0x1c0

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CPU Protect Region 4 Base Address Corresponds to address bits[35:20].	0xfff
16	AccProtect	CPU Access Protect 0 - Access allowed. 1 - Access forbidden.	0x0
17	WrProtect	CPU Write Protect 0 - Write allowed. 1 - Write forbidden.	0x0
18	CacheProtect	CPU Caching Protect 0 - Caching (block read) allowed. 1 - Caching is forbidden.	0x0
31:19	Reserved	Reserved.	0x0

Table 107: CPU Protect Address 4 (High), Offset: 0x1c8

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CPU Protect Region 4 Top Address Corresponds to address bits[31:20].	0x0
31:12	Reserved	Reserved.	0x0

Table 108: CPU Protect Address 5 (Low), Offset: 0x1d0

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CPU Protect Region 5 Base Address Corresponds to address bits[35:20]	0xfff
16	AccProtect	CPU Access Protect. 0 - Access allowed. 1 - Access forbidden.	0x0

Table 108: CPU Protect Address 5 (Low), Offset: 0x1d0 (Continued)

Bits	Field Name	Function	Initial Value
17	WrProtect	CPU Write Protect 0 - Write allowed. 1 - Write forbidden.	0x0
18	CacheProtect	CPU Caching Protect 0 - Caching (block read) allowed. 1 - caching is forbidden.	0x0
31:19	Reserved	Reserved.	0x0

Table 109: CPU Protect Address 5 (High), Offset: 0x1d8

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CPU Protect Region 5 Top Address Corresponds to address bits[31:20].	0x0
31:12	Reserved	Reserved.	0x0

Table 110: CPU Protect Address 6 (Low), Offset: 0x1e0

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CPU Protect Region 6 Base Address Corresponds to address bits[35:20].	0xfff
16	AccProtect	CPU Access Protect. 0 - Access allowed. 1 - Access forbidden.	0x0
17	WrProtect	CPU Write Protect 0 - Write allowed. 1 - Write forbidden.	0x0
18	CacheProtect	CPU Caching Protect 0 - Caching (block read) allowed. 1 - Caching is forbidden.	0x0
31:19	Reserved	Reserved.	0x0

Table 111: CPU Protect Address 6 (High), Offset: 0x1e8

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CPU Protect Region 6 Top Address Corresponds to address bits[31:20]	0x0
31:12	Reserved	Reserved.	0x0

Table 112: CPU Protect Address 7 (Low), Offset: 0x1f0

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CPU Protect Region 7 Base Address Corresponds to address bits[35:20].	0xfff
16	AccProtect	CPU Access Protect 0 - Access allowed. 1 - Access forbidden.	0x0
17	WrProtect	CPU Write Protect 0 - Write allowed. 1 - Write forbidden.	0x0
18	CacheProtect	CPU Caching Protect 0 - Caching (block read) allowed. 1 - Caching forbidden.	0x0
31:19	Reserved	Reserved.	0x0

Table 113: CPU Protect Address 7 (High), Offset: 0x1f8

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CPU Protect region 7 Top Address Corresponds to address bits[31:20].	0x0
31:12	Reserved	Reserved.	0x0

4.15.5 CPU Error Report Registers

Table 114: CPU Error Address (Low), Offset: 0x070[1]

Bits	Field Name	Function	Initial Value
31:0	ErrAddr	Latched address bits [31:0] of a CPU transaction in case of: • illegal address (failed address decoding) • access protection violation • bad data parity Upon address latch, no new address are registered (due to additional error condition), until the register is being read. Read Only.	0x0

1. In case of multiple errors, only the first one is latched. New error report latching is enabled only after the CPU Error Address (Low) register is being read.

Table 115: CPU Error Address (High), Offset: 0x078[1]

Bits	Field Name	Function	Initial Value
3:0	ErrAddr	Latched address bits [35:32] of a CPU transaction in case of: • illegal address (failed address decoding) • access protection violation • bad data parity. Read Only.	0x0
31:4	Reserved	Read Only	0x0

1. Once data is latched, no new data can be registered (due to additional error condition), until CPU Error Low Address is being read (which implies, it should be the last being read by the interrupt handler).

Table 116: CPU Error Data (Low), Offset: 0x128

Bits	Field Name	Function	Initial Value
31:0	PErrData	Latched data bits [31:0] in case of bad data parity sampled on write transactions. Read only.	0x0

Table 117: CPU Error Data (High), Offset: 0x130

Bits	Field Name	Function	Initial Value
31:0	PErrData	Latched data bits [63:32] in case of bad data parity sampled on write transactions. Read only.	0x0

Table 118: CPU Error Parity, Offset: 0x138

Bits	Field Name	Function	Initial Value
7:0	PErrPar	Latched data parity bus in case of bad data parity sampled on write transactions. Read only.	0x0
31:10	Reserved	Reserved.	0x0

Table 119: CPU Error Cause, Offset: 0x140[1]

Bits	Field Name	Function	Initial Value
0	AddrErr	CPU Address Out of Range	0x0
1	Reserved	Read only.	0x0
2	TTErr	Transfer Type Violation. The CPU attempts to burst (read or write) to an internal register.	0x0
3	AccErr	Access to a Protected Region	0x0
4	WrErr	Write to a Write Protected Region	0x0
5	CacheErr	Read from a Caching protected region	0x0
6	WrDataPErr	Bad Write Data Parity Detected	0x0
26:7	Reserved	Read only.	0x0
31:27	Sel	Specifies the error event currently being reported in Error Address, Error Data, and Error Parity registers. 0x0 - AddrOut 0x1 - Reserved 0x2 - TTErr 0x3 - AccErr 0x4 - WrErr 0x5 - CacheErr 0x6 - WrDataPErr 0x7 - 0x1f - Reserved Read Only.	

1. Bits[7:0] are clear only. A cause bit is set upon an error condition occurrence. Write a ‘0’ value to clear the bit. Writing a 1 value has no affect.

Table 120: CPU Error Mask, Offset: 0x148

Bits	Field Name	Function	Initial Value
0	AddrErr	If set to ‘1’, enables AddrOut interrupt.	0x0
1	Reserved	Read only.	0x0
2	TTErr	If set to ‘1’, enables TTErr interrupt.	0x0
3	AccErr	If set to ‘1’, enables AccErr interrupt.	0x0
4	WrErr	If set to ‘1’, enables WrErr interrupt.	0x0
5	CacheErr	If set to ‘1’, enables CacheErr interrupt.	0x0
6	WrDataPErr	If set to ‘1’, enables WrDataPErr interrupt.	0x0
31:	Reserved	Reserved.	0x0

5. SDRAM CONTROLLER

The SDRAM controller supports up to four banks of SDRAMs (four SDRAM chip selects). It has a 15-bit address bus (DAdr[12:0] and BankSel[1:0]) and a 64-bit data bus (SData[63:0]).

The SDRAM controller supports 16, 64, 128, 256 or 512Mbit SDRAMs. Up to 1 Gbytes can be addressed by each SCS for a total SDRAM address space of 4 Gbytes by the GT–64240A.

NOTE: Whenever this datasheet refers to 64-bit SDRAM, it means 64-bits of data plus eight additional bits for ECC.

The memory controller will only MASTER read and write transactions to SDRAM initiated by the CPU, IDMA, one of communication ports SDMAs, or the PCI. The SDRAM bus may be shared with other masters through the UMA bus arbitration protocol.

The SDRAM controller supports unbuffered and registered SDRAM DIMMS. It runs at up to 133MHz, which results in bandwidth of up 1Gbyte/sec. This upper limit bandwidth number is easily achieved by taking advantage of the DRAM controller bank interleave feature.

It is also possible to configure the DRAM controller to keep pages open. This eliminates the need to close a page (precharge cycle) and re-open it (activate cycle) in case of consecutive accesses to the same page. This is typically useful when the CPU fetches the code from DRAM to its internal cache, or in case of long DMA bursts to/from DRAM.

5.1 SDRAM Controller Implementation

The SDRAM controller contains two 512bytes write buffers and two 512 bytes read buffers. It can absorb up to four read transactions plus four write transactions.

Once a DRAM access is requested, it is pushed into a transaction queue. The SDRAM controller drives the transaction to DRAM as soon as it receives the address. It drives part of the address bits on DAdr[12:0] and BankSel[1:0] during the activate cycle (RAS*) and the remaining bits during the command cycle (CAS*).

In case of a write transaction, write data is placed in the write buffer. The SDRAM controller pops the data from the write buffer and drives it on the DRAM data bus right after the command (CAS*) cycle.

The DRAM write buffer allows the originating unit to complete a write transaction, even if the DRAM controller is currently busy in serving a previous transaction. The maximum input bandwidth to the DRAM controller is 2 Gbyte/sec. This bandwidth peak is attainable during simultaneous accesses to DRAM from multiple interfaces (CPU, PCI, DMAs). In such cases, the write buffers are utilized.

In case of a read transaction, after command cycle (RAS*), the SDRAM controller samples read data driven by the DRAM (sample window depends on CL parameter), pushes the data into the read buffer, and drives it back to the originating unit.

In case the read buffer is empty, the DRAM controller bypasses the read buffer and drives read data directly to the originating unit, in order to gain minimum read latency. However, if there is some data in the read buffer from a previous transaction, data is written first to the buffer. This typically happens when an originating unit issues multiple read transactions (split transactions).

For example, if the CPU interface issues a read from the PCI, and latter issues another read from DRAM, by the time the DRAM controller is able to return read data, the CPU interface unit might not be able to absorb the data

The CPU interface is busy in receiving read data from the PCI. In this case, read data from DRAM is placed in the read buffer and only pushed to the CPU interface unit later, when it is ready to receive the data.

The two read buffers are also used for decoupling reads to different resources. Via the SDRAM Configuration register, each requesting interface (CPU, PCI, IDMA, and Comm ports) can be assigned to use one of the two buffers. For example, if the CPU read latency is important and shouldn't be delayed due to some PCI read data waiting in the buffer head, assigning one buffer for the CPU interface and the other buffer to the other interfaces guarantees the minimum CPU read latency.

5.2 DRAM Type

It is possible to configure the GT–64240A DRAM controller to interface SDRAM or registered SDRAM, according to the setting of DType bits in the SDRAM Configuration register, see Table 127 on page 124.

NOTE: All DRAM banks must be of the same type.

The following figures show typical read transactions.

NOTE: DRAM timing parameters (Trcd and CL) in these examples are the same (See Section 5.4 "SDRAM Timing Parameters" on page 106 Timing Parameters for more details).

Figure 11 shows a SDRAM burst read of 4. It consists of activate cycle (RAS*); followed by command cycle (CAS*); followed by precharge.

Figure 11: SDRAM Read Example

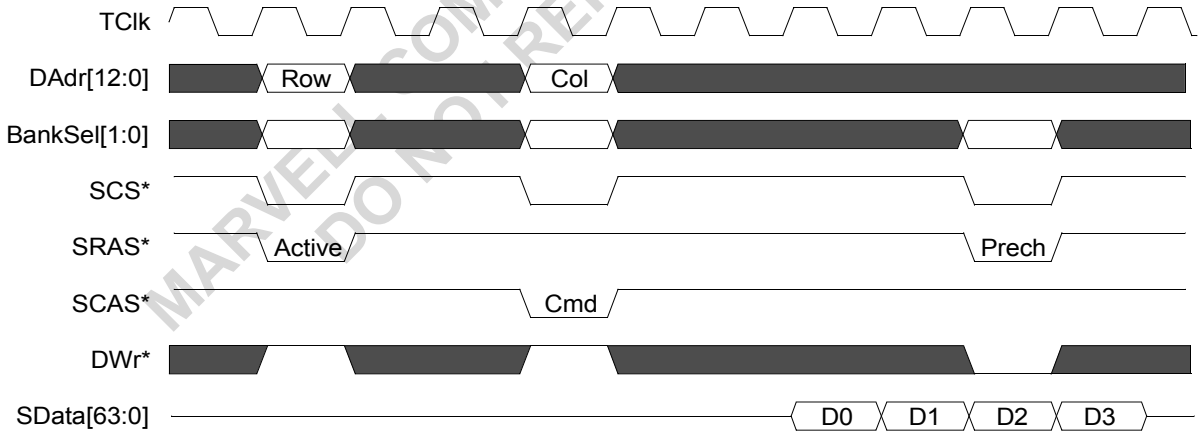
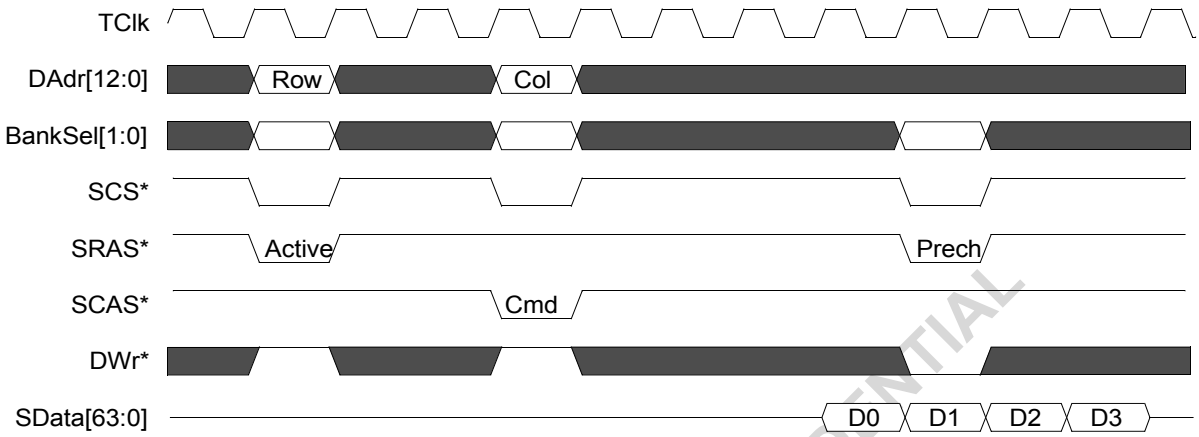


Figure 12 shows a registered SDRAM read. In registered SDRAM, all address and control signals (DAdr[12:0], BankSel[1:0], RAS*, CAS*, DWr*, CS* and DQM*) are registered externally. This means the signals arrive to the SDRAM device one cycle after they are driven by the DRAM controller. It also means that read data arrives back to the DRAM controller one cycle later (in comparison to non-registered SDRAM configuration).

In case of a write transaction, the DRAM controller drives the data one cycle later.

Figure 12: Registered SDRAM Read Example



NOTE: Implement registered SDRAM by using registered DIMMs or on board registers.

5.3 SDRAM Density

The GT–64240A supports 16, 64, 128, 256 and 512Mbit SDRAM devices. Each SDRAM physical bank (SCS[3:0]) can be built of different SDRAM devices. The DRAM density is configured via DRAM Bank Parameter registers.

The different DRAM devices differ in the usage of DAdr[12:0] and BankSel[1:0] lines, as described in the following sections.

5.3.1 16MBit SDRAM

When interfacing with 16Mbit SDRAMs, DAdr[10:0] and BankSel[0] must be connected to address bits 10-0 and the Bank Select of the DRAM device.

NOTE: DAdr[12:11] and BankSel[1] are NOT used when interfacing 16 Mbit SDRAMs.

Therefore, during a SRAS cycle, a valid row address is placed on the DAdr[10:0] and BankSel[0] lines. During the SCAS cycle, a valid column address is placed on DAdr[9:0] (10-bit). DAdr[10] is used as the auto-precharge select bit and is always written “0” during SCAS cycles (no auto precharge). BankSel[0] is held constant from the SRAS cycle.

With 16MBit SDRAMs, the GT–64240A supports a maximum of 4M addresses, 12 address bits for SRAS and 10 address bits for SCAS.

5.3.2 64Mbit SDRAM

When interfacing with 64MBit SDRAMs, DAdr[11:0] and BankSel[1:0] must be connected to address bits 11-0 and the Bank Select of the DRAM device.

NOTE: DAdr[12] is NOT used when interfacing 64Mbit SDRAMs.

Therefore, during a SRAS cycle, a valid row address is placed on the DAdr[11:0] and BankSel lines. During the SCAS cycle, a valid column address is placed on DAdr[9:0] (10-bit). DAdr[10] is used as the auto-precharge select bit and is always written “0” during SCAS cycles (no auto precharge). BankSel is held constant from the SRAS cycle.

With 64MBit SDRAMs, the GT–64240A supports a maximum of 16M addresses, 14 address bits for SRAS and 10 address bits for SCAS.

5.3.3 128Mbit SDRAM

When interfacing 128MBit SDRAMs, DAdr[11:0] and BankSel[1:0] must be connected to address bits 11-0 and the Bank Select of the actual SDRAM.

NOTE: DAdr[12] is NOT used when interfacing 128Mbit SDRAMs.

Therefore, during a SRAS cycle, a valid row address is placed on the DAdr[11:0] and BankSel lines. During the SCAS cycle, a valid column address is placed on DAdr[11,9:0] (11-bit). DAdr[10] is used as the auto-precharge select bit and is always written “0” during SCAS cycles (no auto precharge). BankSel is held constant from the SRAS cycle.

With 128MBit SDRAMs, the GT–64240A supports a maximum of 32M addresses, 14 address bits for SRAS and 11 address bits for SCAS.

5.3.4 256Mbit SDRAMs

When interfacing 256MBit SDRAMs, DAdr[12:0] and BankSel[1:0] must be connected to address bits 12-0 and the Bank Select of the actual SDRAM.

Therefore, during a SRAS cycle, a valid row address is placed on the DAdr[12:0] and BankSel lines. During the SCAS cycle, a valid column address is placed on DAdr[11,9:0] (11-bit). DAdr[10] is used as the auto-precharge select bit and is always written “0” during SCAS cycles (no auto precharge). BankSel is held constant from the SRAS cycle.

With 256MBit SDRAMs, the GT–64240A supports a maximum of 64M addresses, 15 address bits for SRAS and 11 address bits for SCAS.

5.3.5 512Mbit SDRAMs

When interfacing 512MBit SDRAMs, DAdr[12:0] and BankSel[1:0] must be connected to address bits 12-0 and the Bank Select of the actual SDRAM.

Therefore, during a SRAS cycle, a valid row address is placed on the DAdr[12:0] and BankSel lines. During the SCAS cycle, a valid column address is placed on DAdr[12:11,9:0] (11-bit). DAdr[10] is used as the auto-precharge select bit and is always written “0” during SCAS cycles (no auto precharge). BankSel is held constant from the SRAS cycle.

With 512MBit SDRAMs, the GT–64240A supports a maximum of 128M addresses, 15 address bits for SRAS and 12 address bits for SCAS.

5.4 SDRAM Timing Parameters

The SDRAM controller supports a range of SDRAM timing parameters. These parameters can be configured through the SDRAM Timing Parameters register, see Table 130 on page 126.

NOTE: If using different SDRAM devices in each DRAM bank, the SDRAM Timing Parameters register must be programed based on the slowest DRAM device being used.

5.4.1 SCAS* Latency (CL)

SCAS* Latency is the number of TClk cycles from the assertion of SCAS* to the sampling of the first read data (see Figure 13). It is possible to program this parameter for two or three TClks cycles. Selecting this parameter depends on TClk frequency and the speed grade of the SDRAM.

NOTE: In case of changing SCAS* latency, follow the procedure outlined in Section 5.11.4 “Setting SDRAM Mode Register (MRS command)” on page 117 to update the SDRAM’s Mode Register.

5.4.2 SRAS* Precharge (Trp)

The SRAS precharge time specifies the number of TClk cycles following a precharge cycle that a new SRAS* transaction may occur (see Figure 13). It is possible to program this parameter for two or three TClks cycles.

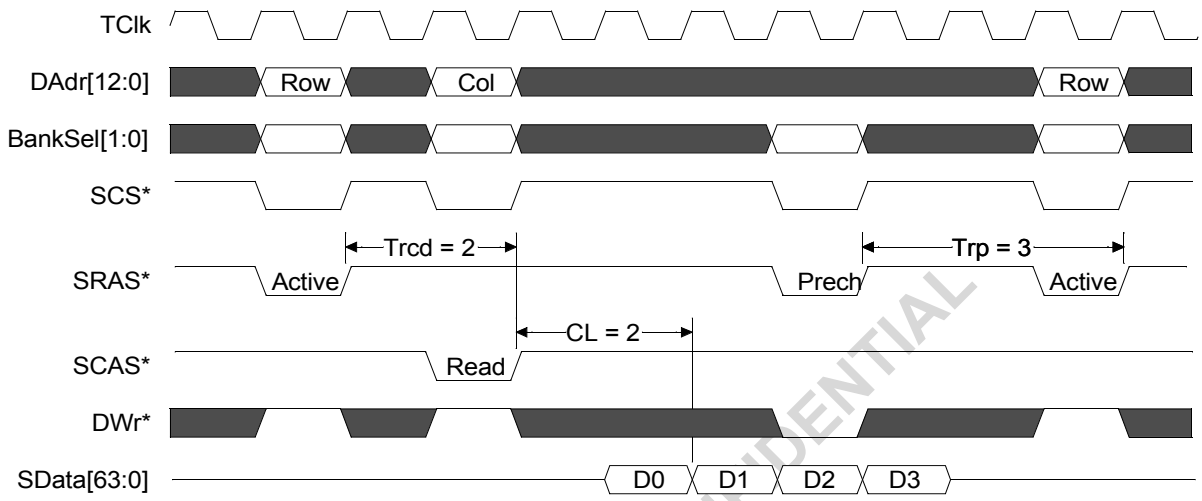
5.4.3 SRAS* to SCAS* (Trcd)

SRAS* to SCAS* specifies the number of TClk cycles that the DRAM controller inserts between the assertion of SRAS* with a valid row address to the assertion of SCAS* with a valid column address (see Figure 13). It is possible to program this parameter for two or three TClks cycles.

5.4.4 Row Active Time (Tras)

Specifies the minimum number of TClk cycles between SRAS* of activate cycle to SRAS* of precharge cycle. The minimum number of cycles guaranteed by design (regardless of this parameter setting) is five TClk cycles when Trcd is set to two TClk cycles, or six when Trcd is set to three TClk cycles. This behavior meets the required Tras of PC100 AC spec. However, when running a faster frequency, Tras might need to be set to six or seven to meet the DIMM AC spec.

Figure 13: SDRAM Timing Parameters



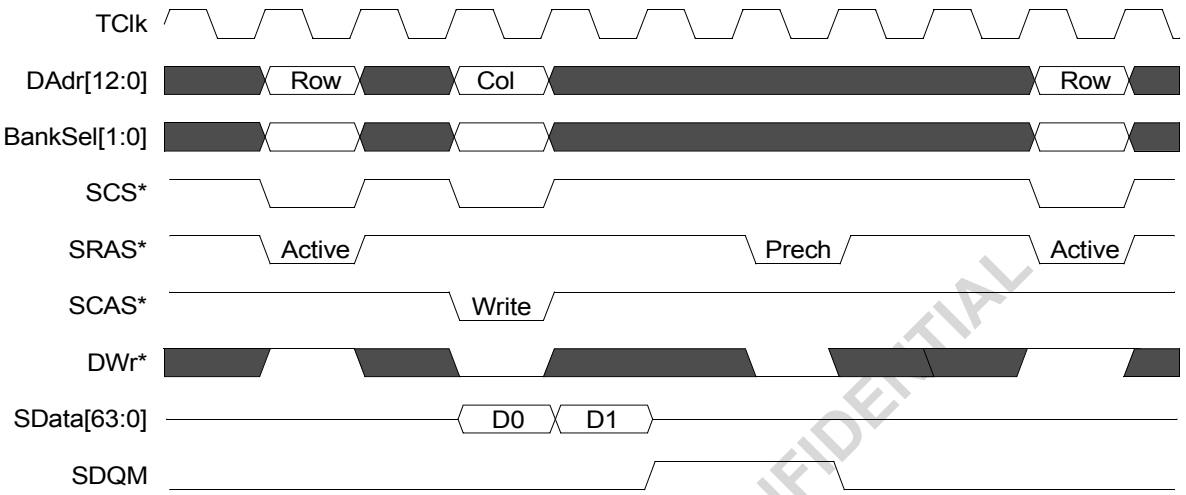
5.5 SDRAM Burst

An SDRAM device can be configured to different burst lengths and burst ordering.

The GT–64240A DRAM controller always configures the DRAM to a burst length of four and linear burst order. It drives the DRAM address and control signals at the appropriate time windows to support the different bursts size and ordering required by the different units.

Access to DRAM does not mean that a full multiple of DRAM bursts is required. When a shorter burst is required, the DRAM controller terminates the burst by driving an early precharge cycle and deasserting SDQM signals. An example is shown in Figure 14.

Figure 14: Burst Write Termination Example



The CPU access to DRAM is single data (one byte up to eight bytes), or full cache line (32-bytes). Other interfaces may burst longer transfers to DRAM. In case of a burst access to DRAM that crosses the burst length alignment, the DRAM controller drives a new SCAS* cycle with new column address.

For a CPU block read, which uses sub-block read ordering, the SCAS* assertion depends on the read start address. If the read starts at offset 0x0 or 0x10, the sub-block and linear wrap around bursts order are the same. There is no special treatment required from the DRAM controller. If it starts at offset 0x8 or 0x18, a new column address (SCAS* assertion) is required for every data of the burst.

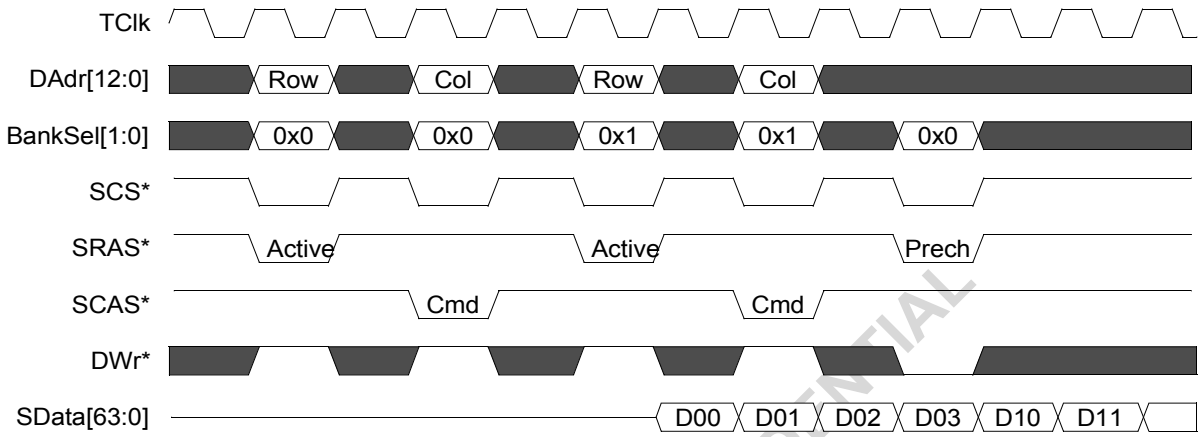
5.6 SDRAM Interleaving

The GT–64240A supports both physical banks (SCS[3:0]*) interleaving and virtual banks (BankSel[1:0]) interleaving. It supports two virtual bank interleaving with 16Mbit SDRAM and four virtual bank interleaving with 64, 128, 256 or 512Mbit SDRAMs.

Interleaving provides higher system performance by hiding a new transaction's activate and command cycles during a previous transaction's data cycles. This reduces the number of wait states before data can be read from or written to SDRAM, which increases bandwidth.

An example of interleaving between two reads to different virtual banks is shown in Figure 15.

Figure 15: Virtual DRAM Banks Interleaving Example



Since the two accesses are targeted to different virtual banks (BankSel[1:0]), interleaving is enabled. Activate and command cycles of the second transaction are issued while the first transaction is receiving read data.

NOTE: A precharge is required to each bank at the end of the burst, unless the page is kept open, see Section 5.7 “SDRAM Open Pages” on page 112.

5.6.1 Bank Interleaving Implementation

Interleaving occurs when there are multiple pending accesses to different SDRAM banks.

It occurs in the GT–64240A when a DRAM access requests from different units (PCI, CPU, IDMA, Comm Ports) or during multiple transactions from the same unit. Since most of the GT–64240A units support split transactions, they issue a new transaction before a previous transaction completes.

The DRAM devices have two or four virtual banks. The GT–64240A DRAM controller supports two bank interleaving for 16Mbit devices and four bank interleaving for 64, 128, 256, and 512Mbit devices. In case of a two way interleave, it performs transaction interleaving when the two transactions require different BankSel[0] values. If programed to four way interleave, it executes interleaving if the two transactions require different BankSel[1:0] values.

When the two transactions are targeted to different physical banks (different SCS*), the DRAM controller also performs interleaving. In some applications, this type of interleaving is unwanted. The user can disable interleaving between physical banks via SDRAM Configuration register, see Table 127 on page 124.

5.6.2 SDRAM Address Control

The Address Control Register is a four bit register that determines how address bits driven by the CPU, PCI, or DMA to the SDRAM controller are translated to row and column address bits on DAdr[12:0] and BankSel[1:0]. This flexibility allows the designer to choose the address decode setting which gives the software a better chance of virtual banks interleaving, thus enhancing overall system performance.

If, for example, the CPU, PCI_0, PCI_1, and IDMA access the same physical bank (SCS*), and each of them is using a different 16Mbyte slice of the DRAM in a configuration in which address bits[25:24] are mapped to BankSel[1:0], bank interleaving always takes place between accesses to DRAM from the different units.

The row and column address translation is different for 16Mbit, 64/128Mbit, or 256/512Mbit SDRAMs, as shown in Table 121 through Table 123.

Table 121: Address Control for 16Mbit SDRAM

Address Control	BankSel[0]	Initiator Address Bits used for Row Address DAdr[10:0]	Initiator Address Bits used for Column Address DAdr[10:0]
0000[1]	5	22-12	”0”, 24-23, 11-6, 4-3
0001[2]	6	22-12	”0”, 24-23, 11-7, 5-3
1000	7	22-12	“0”, 24-23, 11-8, 6-3
0010	11	22-12	”0”, 24-23, 10-3
1001	12	22-13, 11	“0”, 24-23, 10-3
0011	13	22-14, 12-11	”0”, 24-23, 10-3
0100	21	22, 20-11	“0”, 24-23, 10-3
0101	22	21-11	“0”, 24-23, 10-3
0110[3]	23	22-12	“0”, 24, 11-3
0111[4]	24	22-12	“0”, 23, 11-3

1. Only for SDRAM maximum burst of 4.

2. Only for SDRAM maximum burst of 4 or 8.

3. Only for x4 or x8 devices.

4. Only for x4 devices.

Table 122: Address Control for 64/128Mbit SDRAM

Address Control	BankSel[1:0]	Initiator Address Bits used for Row Address DAdr[11:0]	Initiator Address Bits used for Column Address DAdr[11:0]
0000[1]	6-5	24-13	27, “0”, 26-25, 12-7, 4-3
0001[2]	7-6	24-13	27, “0”, 26-25, 12-8, 5-3
1000	8-7	24-13	27, “0”, 26-25, 12-9, 6-3
0010	12-11	24-13	27, “0”, 26-25, 10-3
1001	13-12	24-14,11	27, “0”, 26-25, 10-3

Table 122: Address Control for 64/128Mbit SDRAM (Continued)

Address Control	BankSel[1:0]	Initiator Address Bits used for Row Address DAdr[11:0]	Initiator Address Bits used for Column Address DAdr[11:0]
0011	14-13	24-15, 12-11	27, “0”, 26-25, 10-3
0100	22-21	24-23, 20-11	27, “0”, 26-25, 10-3
1010	23-22	24, 21-11	27, “0”, 26-25, 10-3
0101	24-23	22-11	27, “0”, 26-25, 10-3
0110[3]	25-24	22-11	27, “0”, 26, 23, 10-3
0111[4]	26-25	22-11	27, “0”, 24-23, 10-3
1011[5]	27-26	22-11	25, “0”, 24-23, 10-3

1. Only for SDRAM maximum burst of 4.

2. Only for SDRAM maximum burst of 4 or 8.

3. Only for x4 or x8 or 8Mx16 devices.

4. Only for x4 or 16Mx8 devices.

5. Only for 32Mx4 devices.

Table 123: Address Control for 256/512Mbit SDRAM

Address Control	BankSel[1:0]	Initiator Address Bits used for Row Address DAdr[12:0]	Initiator Address Bits used for Column Address DAdr[12:0]
0000	6-5[1]	25-13	29-28, “0”, 27-26, 12-7, 4-3
0001	7-6[2]	25-13	29-28, “0”, 27-26, 12-8, 5-3
1000	8-7	25-13	29-28, “0”, 27-26, 12-9, 6-3
0010	12-11	25-13	29-28, “0”, 27-26, 10-3
1001	13-12	25-14,11	29-28, “0”, 27-26, 10-3
0011	14-13	25-15, 12-11	29-28, “0”, 27-26, 10-3
0100	22-21	25-23, 20-11	29-28, “0”, 27-26, 10-3
0101	24-23	25, 22-11	29-28, “0”, 27-26, 10-3

Table 123: Address Control for 256/512Mbit SDRAM (Continued)

Address Control	BankSel[1:0]	Initiator Address Bits used for Row Address DAdr[12:0]	Initiator Address Bits used for Column Address DAdr[12:0]
0110	25-24	23-11	29-28, “0”, 27-26, 10-3
0111	26-25[3]	24, 22-11	29-28, “0”, 24-23, 10-3
1010	27-26[4]	25, 22-11	29-28, “0”, 24-23, 10-3
1011	28-27[5]	25, 22-11	29,26, “0”, 24-23, 10-3
1100	29-28[6]	25, 22-11	27-26, “0”, 24-23, 10-3

1. Only for SDRAM maximum burst of 4.

2. Only for SDRAM maximum burst of 4 or 8.

3. Only for x4 or x8 or x16 or 16Mx32 devices.

4. Only for x4 or x8 or 32Mx16 devices.

5. Only for x4 or 64Mx8 devices.

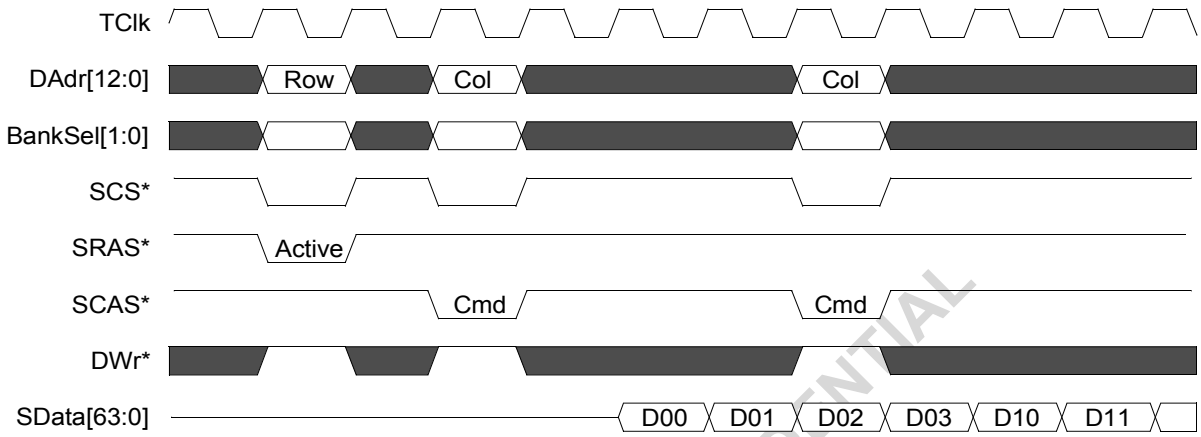
6. Only for 128Mx4 devices.

5.7 SDRAM Open Pages

It is possible to configure the GT–64240A DRAM controller to keep DRAM pages open. It supports up to 16 pages - one per each virtual bank.

When a page is kept open at the end of a burst (no precharge cycle) and if the next cycle to the same virtual bank hits the same page (same row address), there is no need for a new activate cycle. An example is shown in Figure 16.

Figure 16: Sequential Accesses to the Same Page



Via the DRAM Bank Parameters registers, each of the 16 virtual banks can be configured separately to keep the page open at the end of a burst transaction, for fast consecutive accesses to the same page, or close the page, for faster accesses that follow to a different row of the same bank.

If a virtual bank is configured to keep pages open, a bank row is kept open until one of the following events happen:

- An access occurs to the same bank but to a different row address. In this case, the DRAM controller precharges, to close the page, and opens a new one, the new row address.
- The access is smaller than the DRAM burst length. The DRAM controller needs to terminate the burst in the middle using early precharge.
- The Refresh counter expires. The DRAM controller closes all open pages and performs a refresh to all banks.

5.8 Read Modify Write

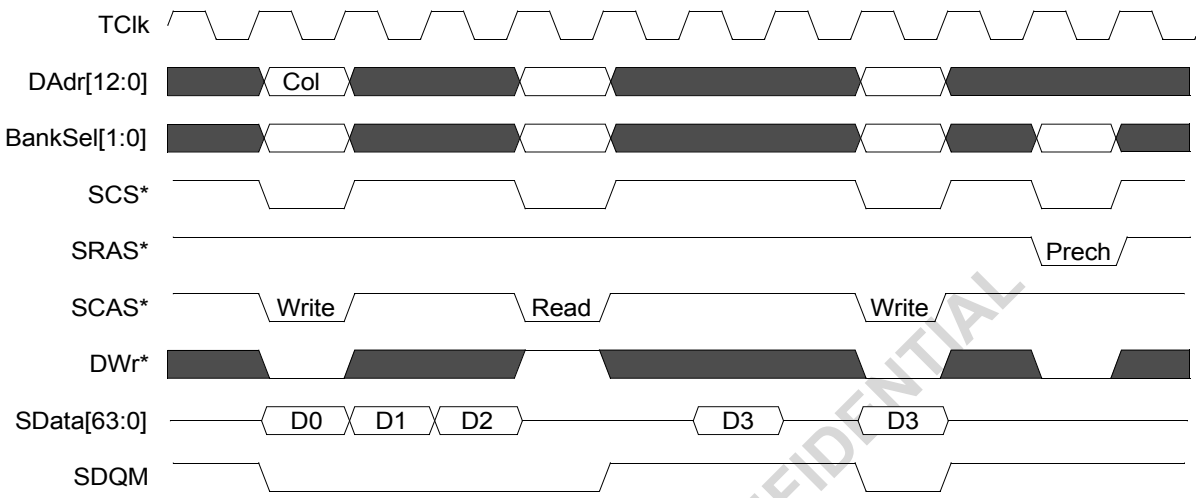
The GT–64240A supports Error Checking and Correction (ECC).

ECC is enabled via DRAM Timing Parameters register. ECC checking and generation requires a 72-bit wide DRAM to store the ECC information, 64 bits for data and eight bits for ECC. In order to generate the ECC on partial writes (less than 64 bits), Read Modify Write (RMW) access is required to do the following:

1. Read the existing 64-bit data from DRAM.
2. Merge the new incoming data with the 64-bit read data. Calculate new ECC byte based on the data that is to be written.
3. Write the new data and new ECC byte back to the DRAM bank. On this write, all SDQM lines are deasserted (LOW). This means that the byte enabled for the ECC byte can be connected to ANY of the SDQM[7:0] outputs.

In case of burst write to DRAM, the GT–64240A executes a RMW access only for the required data. A typical example is shown in Figure 17. The DRAM controller performs a burst write of four, with RMW only to last data (which is not a full 64-bit data).

Figure 17: SDRAM RMW Example



For more details on DRAM ECC support, see Section 6. “Address and Data Integrity” on page 133.

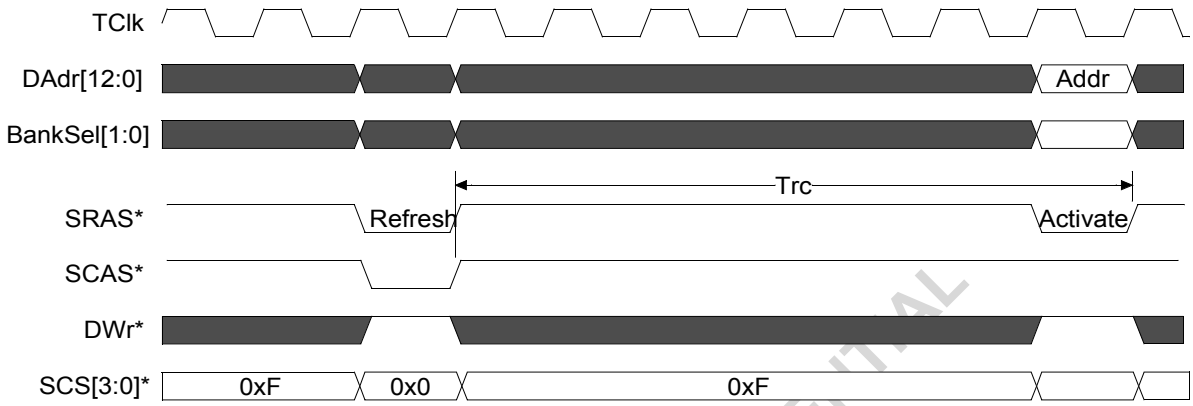
5.9 SDRAM Refresh

The GT–64240A implements standard SCAS before SRAS refreshing.

The refresh rate for all banks is determined according to the 14-bit RefIntCnt value in SDRAM Configuration register. For example, the default value of RefIntCnt is 0x200. If the TClk cycle is 133MHz, a refresh sequence occurs every 5.12us. Every time the refresh counter reaches its terminal count, a refresh request is sent to the SDRAM Controller to be executed.

Non-staggered or staggered refresh for all banks is determined according to StagRef bit in SDRAM Configuration register. In non-staggered refresh, SCS[3:0]*, SRAS*, and SCAS* simultaneously assert refreshing all banks at the same time as shown in Figure 18.

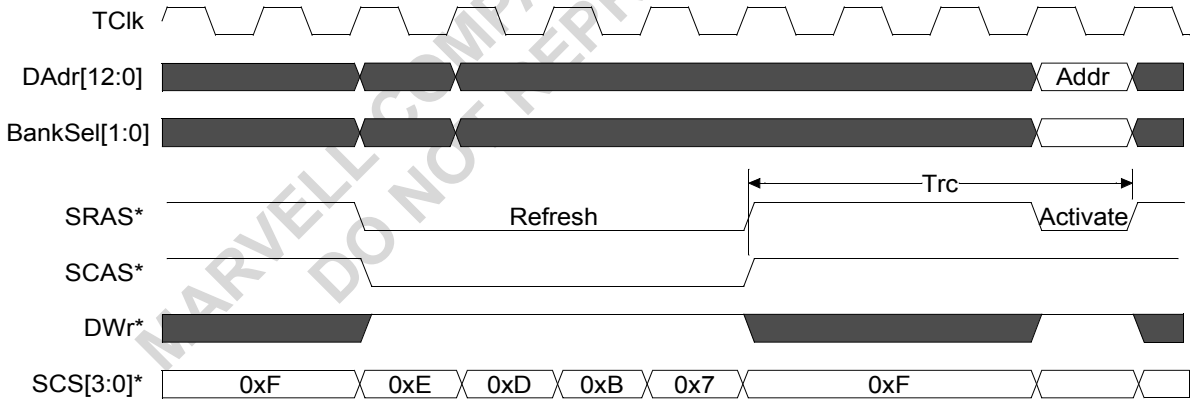
Figure 18: Non-Staggered Refresh Waveform



If the SDRAM Controller is programmed to perform staggered refresh (the default setting), SCS[0]* goes LOW for one TClk cycle, followed by SCS[1]* on the next TClk cycle, and so on. After the last SCS[3]* has asserted LOW for one cycle, SCAS* and SRAS* goes HIGH again.

Staggered Refresh is useful for load balancing, see Figure 19.

Figure 19: Staggered Refresh Waveform



NOTE: The DRAM controller will not issue a new access to DRAM (new activate cycle) for the number of Trc cycles as specified by SDRAM AC spec.

5.10 SDRAM Initialization

The DRAM controller executes the SDRAM initialization sequence as soon as the GT–64240A goes out of reset.

The initialization sequence consists of the following steps:

1. SRAS* and DWr* are asserted with DAdr[10] HIGH and SCS[3:0] = 0000. This indicates a Precharge to all of the SDRAM Banks.
2. SRAS* and SCAS* are asserted with SCS[3:0] = 0000. This indicates an auto refresh (CBR) to all SDRAM Banks. This occurs twice in a row.
3. SRAS*, SCAS*, and DWr* are asserted 4 times in a row, once with SCS[3:0] = 1110, once with SCS[3:0] = 1101, once with SCS[3:0] = 1011, and once with SCS[3:0] = 0111. This command programs each of the SDRAM Mode registers by individually activating each of the four chip selects (SCS[3:0]).

The DRAM controller performs an MRS cycle based on the default DRAM parameters (CL = 3,burst length = 4,burst order = linear). The software can change CL to ‘2’ if the DRAM device is capable of this CAS latency. See 5.11 for more information.

NOTES: The DRAM controller postpones any attempt to access SDRAM before the initialization sequence completes.

If the serial ROM initialization is enabled, the DRAM controller postpones the above DRAM initialization sequence until the serial ROM initialization completes.

The DRAM controller drives the DRAM address and control signals to their inactive value during reset assertion, as required by the DRAM spec (100us of idle cycles before DRAM initialization).

5.11 SDRAM Operation Mode Register

The SDRAM Operation Mode register is used to execute commands other than standard memory reads and writes to the SDRAM. These operations include:

- Normal SDRAM Mode
- NOP Commands
- Precharge All Banks
- Writing to the SDRAM Mode Register
- Force a Refresh Cycle

The register contains three command type bits plus an activate bit. In order to execute one of the above commands on the SDRAM, the following procedure must occur:

1. Write to the SDRAM Operation Mode register the required command.
2. Read the SDRAM Operation Mode register. This read guarantees that the following step is executed after the register value is updated.
3. Dummy word (32-bit) writes to an SDRAM bank. This eventually causes that the required cycle is driven to the selected DRAM bank.
4. Polling on SDRAM Operation Mode register until activate bit is sampled ‘1’. A ‘1’ indicates that the MRS cycle is done.

5. Write a value of 0x0 to the SDRAM Operation Mode Register. This value returns the register to Normal SDRAM Mode.
6. Read the SDRAM Operation Mode register. This read guarantees the execution of the following access to the DRAM, after the register value is updated.

NOTE: The above sequence is different than the sequence required in the GT-64120/130 devices.

5.11.1 Normal SDRAM Mode

Write 0x0 to the SDRAM Operation Mode register to enable normal reading and writing to the SDRAM.

5.11.2 NOP Commands

Use the NOP command to perform a NOP to an SDRAM selected by the SDRAM Chip Select register (SCS[3:0]*). This prevents unwanted commands from being registered during idle or wait states.

5.11.3 Precharge All Banks

Use the Precharge All Banks command to close open rows in all four (two) virtual banks.

When a bank has been precharged, it is in the idle state and must be activated prior to any read or write commands being issued to that bank.

5.11.4 Setting SDRAM Mode Register (MRS command)

Each SDRAM has its own Mode register.

Use the Mode register to define the DRAM burst length, burst order, and SCAS latency.

As part of the DRAM initialization sequence, the DRAM controller generates an MRS cycle to each of the four DRAM banks right after reset. The software can then change CAS latency using the procedure specified in 5.11. Since the DRAM controller restricts CAS latency to be the same for all four banks (SCS[3:0]*), it must perform an MRS cycle to all banks. An MRS cycle means a dummy write to each DRAM bank.

NOTES: When using DRAM DIMMs, the DRAM parameters are recorded in the DIMM Serial Presence Detect (SPD) serial ROM. The CPU reads the SPD via the GT–64240A I^2C interface and programs the DRAM parameters accordingly.

The software code that performs the sequence of changing the DRAM mode register must not be located in the DRAM. It can be located anywhere else (boot ROM, CPU cache).

5.11.5 Force Refresh

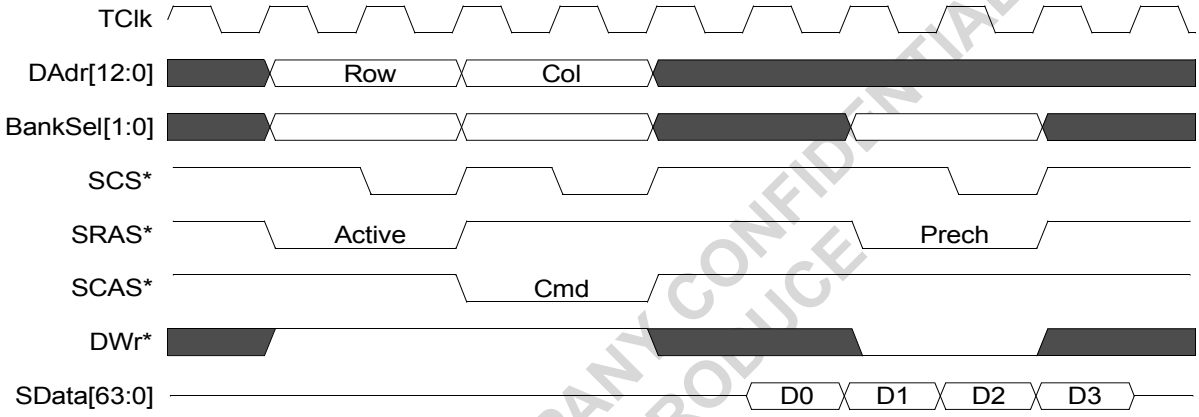
On the particular bank that is accessed, use the Force Refresh Command to execute a refresh cycle.

5.12 Heavy Load Interface

When interfacing heavy load, unbuffered DIMMs (above 50 pF), the GT–64240A might not meet the DRAM control lines AC spec at 133MHz. The DRAM controller includes a mechanism to stretch these signals over two clock cycles, thus guaranteeing proper AC timing. However, when using this method, there is a penalty of latency cycles per each transaction.

An example is shown in Figure 20.

Figure 20: Heavy Load Example



The minimum penalty is one cycle, since row address need to be prepared one cycle before the actual activate cycle (SCS* assertion). During a burst access that requires changing column address in the middle, there is a one cycle penalty per each additional SCAS*.

When interfacing multiple DRAM DIMMS at 133MHz, it is recommended to use registered a SDRAM that has a small load on the DRAM control signals (since they are registered), rather than the above heavy load method. There is a one cycle latency penalty per a single transaction in both methods, in comparison to the regular SDRAM. However, when running many back to back transactions to DRAM, stretching the RAS* and CAS* cycles delays the issuance of a new DRAM transaction. More over, bank interleaving is less likely to happen.

5.13 SDRAM Clocking

The GT–64240A SDRAM interface is working in TClk domain. All output signals are toggled on the rising edge of TClk and all inputs are sampled on rising edge of TClk.

The GT–64240A integrates an internal PLL. The PLL guarantees that the clock signal triggering the output signals is phase locked on the external TClk signal. This implementation minimizes the output delay of the DRAM interface output signals.

The GT–64240A is designed to interface SDRAM at 133MHz, assuming both the GT–64240A and the SDRAM are clocked from the same external clock driver (up to 0.35ns clock skew/gitter between the SDRAM clock and the GT–64240A clock). However, the GT–64240A also has alternative mechanisms that guarantees 100133MHz DRAM interface in case of problematic board design.

NOTE: Select the appropriate clocking scheme based on board simulation, using GT–64240A and DRAM IBIS models.

5.13.1 SDRAM Clock Output

If AD[23] pin is sampled High during reset, the GT–64240A SDClkOut/SDClkIn pin is configured as SDClkOut (see reset configuration section). The GT–64240A SDClkOut pin can be used as the DRAM clock source, instead of the external TClk source. SDClkOut is the same internal clock used to toggle the DRAM interface output signals (the end point of DRAM interface clock tree). If using this clock, the DRAM interface signals have improved output delays (see Table 27 on page 538).

NOTE: It is recommended that the board be designed to support SDRAM clocking from both the TClk clock generator and SDClkOut signal. For details, see the corresponding evaluation board specification.

5.13.2 Read Data Sample

The read data coming from DRAM is sampled with the internal PLL clock. If driving the SDRAM with SDClkOut, the read data path gets shorter and the GT–64240A might not be able to sample the incoming data on time.

To overcome this obstacle, the DRAM interface supports an additional sampling stage of the incoming data triggered by SDClkOut rather than the internal PLL clock. Setting the SDRAM Timing Parameters register's RdDelay bit to '1' enables this additional sampling stage, see Table 130 on page 126.

NOTES: The routing of SDClkOut back to this additional sampling stage is done inside the device.

With the additional sampling stage, DRAM read latency is increased by one cycle

5.13.3 SDRAM Clock Input

The large DRAM output delay (5.4ns), does not give much margin for the read path. In many cases, for the DRAM controller deal with such an output delay, it must have 0ns (or even negative) setup time. For these cases, the GT–64240A also supports separate SDClkIn for the read path.

If AD[23] is sampled Low during reset, the GT–64240A SDClkOut/SDClkIn pin is configured as SDClkIn (see Table 682 on page 522). Under this configuration, the additional read sample stage (see section 5.13.2), is triggered with the incoming SDClkIn clock.

With this configuration, the clock that is routed to the DRAM, is also routed back from the DRAM pin, back to the GT–64240A SDClkIn. This scheme gaurantees, that the long read path is compensated with a similar clock path, and the read data is sampled properly by the GT–64240A.

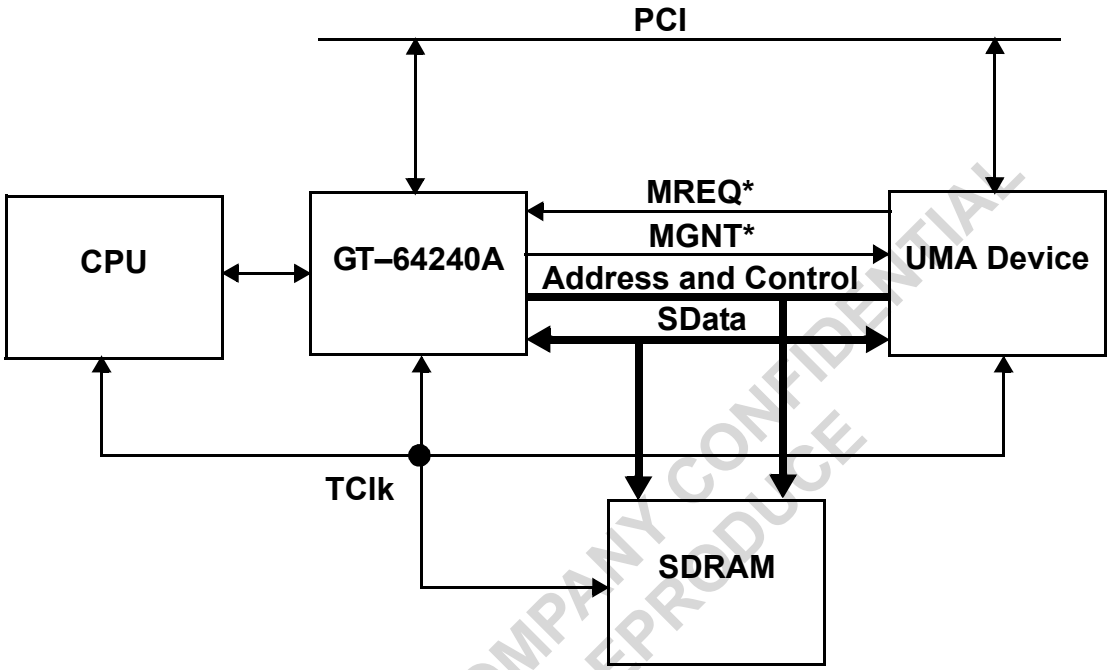
NOTE: If the board design also suffers from address and control line timing problem, externally generate two separate TClk signals - one for the GT–64240A, and a "late" TClk for the DRAM. The exact skew between the two is board design dependent.

5.14 Unified Memory Architecture Support

The GT–64240A supports Unified Memory Architecture (UMA). This feature allows an external master device to share the same physical SDRAM memory that is controlled by the GT–64240A.

A UMA device refers to any type of controller which needs to share the same physical system memory and have direct access to it as shown in Figure 21.

Figure 21: UMA Device and GT–64240A Sharing SDRAM



At reset, the GT–64240A can be configured to act as a UMA master or slave. This is particularly required when the DRAM is shared between multiple GT–64240A devices. With two GT–64240A devices sharing the same DRAM, the devices can be connected gluelessly. One device acts as a master and the other device acts as a slave. When more than two devices are sharing the DRAM, an external arbiter is required.

UMA is enabled by setting UMAEn bit in SDRAM UMA Control register to '1'. The GT–64240A is configured to act as a UMA master or slave via UMAMode bit. In addition, two of the MPP pins must be configured as MREQ* and MGNT* pins, see Section 19.1 "MPP Multiplexing" on page 480.

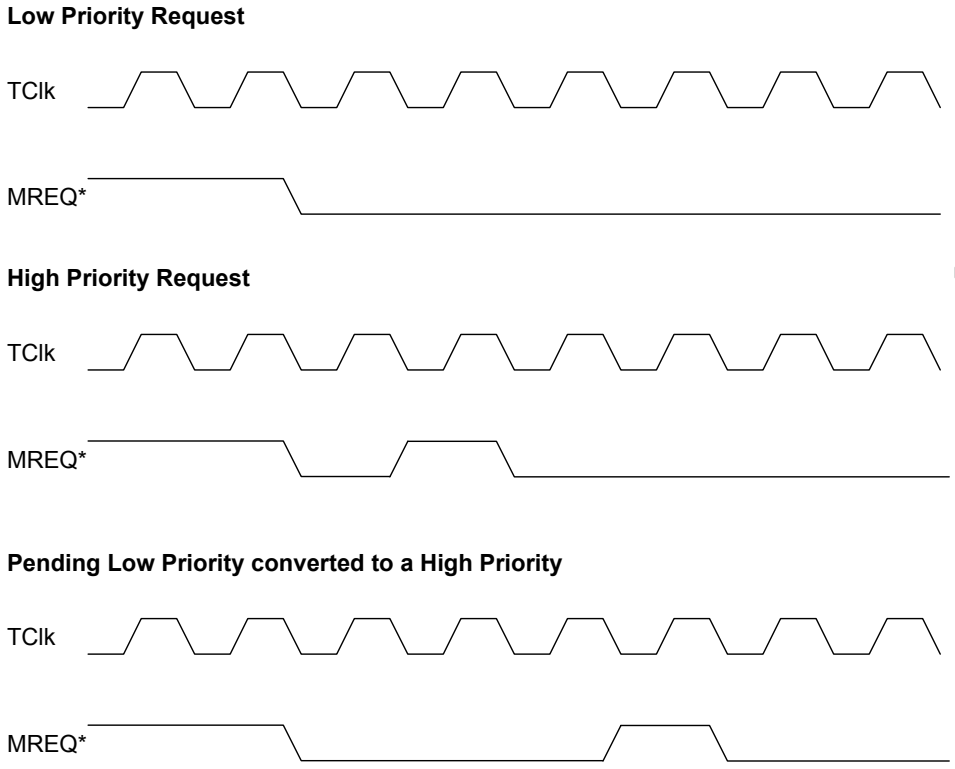
5.14.1 SDRAM Bus Arbitration

MREQ* is an output of the UMA slave device, indicating to the master that it requests ownership on the DRAM bus.

MGNT* is an output of the master to the UMA slave device, indicating that it has received DRAM bus ownership.

UMA devices may request access to SDRAM with either a low or high priority. Both of these priorities are conveyed to the master through the single MREQ* signal, as shown in Figure 22.

Figure 22: UMA Device Requests



The UMA slave device must adhere to the following rules:

- Once MREQ* is asserted by the UMA device for a low priority request, it must be kept asserted until the UMA device is given access to SDRAM via MGNT*. The only reason to change the status of the MREQ* pin is to raise a high priority request or raise the priority of an already pending low priority request.
- Once the UMA device samples MGNT* asserted, it gains and retains access to SDRAM until MREQ* is de-asserted.
- When the UMA device has ownership of the bus, it has full responsibility to execute refresh cycles on the SDRAM.
- Before the UMA device hands over the bus, it must perform refresh cycles to all DRAM banks, and wait Trc cycles before deasserts MREQ*
- Once the UMA device de-asserts MREQ* to transfer ownership back to the GT–64240A, MREQ* must be de-asserted for at least three TClks before asserting it again to raise a request.

If a UMA device places a low priority request for access to SDRAM, there is no set time specified by the GT–64240A to assert MGNT*. Once there are no pending SDRAM access requests, MGNT* is asserted.

If a UMA device places a high priority request for an access to SDRAM, the GT–64240A asserts MGNT* and release the bus, as soon as it’s done with the current outstanding transaction.

NOTE: When the GT–64240A asserts MGNT*, it keeps MGNT* asserted as long as MREQ* is asserted and there is no pending internal request. As soon as any of the GT–64240A interfaces request access to SDRAM or MREQ* is deasserted, the GT–64240A deasserts MGNT* to indicate that it requires bus ownership.

After reset deassertion, the GT–64240A generates DRAM initialization sequence. It responds to MREQ* only after initialization completes.

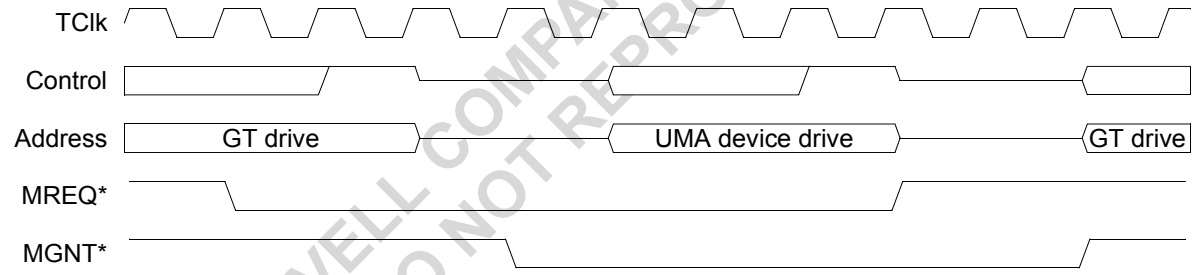
The following rules must be followed by a UMA master device:

- UMA master device must not take bus ownership for three cycles after MREQ* is sampled de-asserted.
- After de-asserting MGNT*, the UMA master device must not assert MGNT* for three cycles.

Once the GT–64240A asserts MGNT* and the UMA slave device gains access to SDRAM, the SCS[3:0]*, SRAS*, SCAS*, DWr*, SData[63:0], SDQM[7:0], DAdr[12:0], and BankSel[1:0] are held in sustained tri-state until the GT–64240A regains access to SDRAM. During this period, the UMA device must drive these signals to access SDRAM.

When the GT–64240A and the UMA device hand the bus over to each other, they must drive all of the above signals HIGH for one TClk and then float the pins, except the SDRAM address lines. There is no need to drive the SDRAM address lines before floating the bus. A sample waveform is shown in Figure 23.

Figure 23: Handing the Bus Over



NOTE: The DRAM bus is floated for two cycles during bus hand over.

The above figure is just an example of bus hand over between the GT–64240A and the UMA device. In reality, the UMA device drives the bus for much longer period.

5.14.2 UMA Arbitration Control

The DRAM controller uses a round robin arbiter to select between refresh requests, DRAM access request or high priority UMA request. With low priority requests, the GT–64240A grants the bus to the UMA device when there is no pending internal request. With high priority requests, the round robin arbiter guarantees, in the worst case, that the UMA device acquires the bus mastership after a refresh cycle plus one DRAM access.

When configured as a UMA slave device, the GT–64240A asserts MREQ* (low priority request) as soon as it has a pending SDRAM access request. The DRAM controller contains a UMA High Priority Request Counter that determines after how many cycles the request must be converted to high priority. Setting the counter to '0' keeps the requests in a low priority status.

As a UMA slave device, the GT–64240A also contains a UMA Bus Release Counter that determines how many cycles after gaining bus ownership the GT–64240A must release the bus. Setting this counter to '0' implies it releases the bus (deassert MREQ*) only when it has no pending SDRAM transactions.

Using these two counters, allows a maximum flexibility of glueless arbitration between two GT–64240A devices sharing the same DRAM.

NOTE: When the GT–64240A gives bus mastership to the UMA slave device, it first performs a refresh cycle, to guarantee a sufficient refresh rate.

5.15 SDRAM Interface Registers

Table 124: SDRAM Configuration Register Map

Register	Offset	Page
SDRAM Configuration	0x448	page 124
SDRAM Operation Mode	0x474	page 125
SDRAM Address Control	0x47c	page 126
SDRAM Timing Parameters	0x4b4	page 126
SDRAM UMA Control	0x4a4	page 127
SDRAM Interface Crossbar Control (Low)	0x4a8	page 128
SDRAM Interface Crossbar Control (High)	0x4ac	page 128
SDRAM Interface Crossbar Timeout	0x4b0	page 129

Table 125: SDRAM Banks Parameters Register Map

Register	Offset	Page
SDRAM Bank0 Parameters	0x44c	page 129
SDRAM Bank1 Parameters	0x450	page 130

Table 125: SDRAM Banks Parameters Register Map (Continued)

Register	Offset	Page
SDRAM Bank2 Parameters	0x454	page 130
SDRAM Bank3 Parameters	0x458	page 130

Table 126: Error Report Register Map

Register	Offset	Page
SDRAM Error Data (Low)	0x484	page 130
SDRAM Error Data (High)	0x480	page 130
SDRAM Error Address	0x490	page 131
SDRAM Received ECC	0x488	page 131
SDRAM Calculated ECC	0x48c	page 131
SDRAM ECC Control	0x494	page 131
SDRAM ECC Error Counter	0x498	page 132

5.15.1 SDRAM Configuration Registers

Table 127: SDRAM Configuration, Offset: 0x448

Bits	Field Name	Function	Initial Value
13:0	RefIntCnt	Refresh Interval Count Value	0x0200
14	VInterEn	Enable Virtual banks (within the same SDRAM device) Interleaving 0 - Interleaving enabled 1 - Interleaving disabled	0x0
15	PhInterEn	Enable Physical banks (SCS[3:0]*) Interleaving 0 - Interleaving enabled 1 - Interleaving disabled	0x0
16	StagRef	Staggered Refresh 0 - Staggered refresh 1- Non-staggered refresh	0x0
18:17	SDType	Select SDRAM Type 00 - SDRAM 01 - Registered SDRAM 1x - Reserved	0x0

Table 127: SDRAM Configuration, Offset: 0x448 (Continued)

Bits	Field Name	Function	Initial Value
19	SDLoad	SDRAM Load 0 - Normal operation 1 - Heavy load operation In heavy load operation: • The DRAM controller drives the row and column addresses for two cycles. • All pages must be closed.	0x1
20	Reserved		0x0
23:21	Reserved	Must be set to 0x6.	0x6
25:24	Reserved	Reserved.	0x0
31:26	RdBuff	Read buffer Assignment per Each Interface If the bit is set to 0, the corresponding unit receives read data from read buffer 0. If the bit is set to 1, the corresponding unit receives read data from the read buffer 1. Bit[26] - CPU read Bit[27] - PCI_0 read Bit[28] - PCI_1 read Bit[29] - Comm ports read Bit[30] - IDMA channels 0/1/2/3 read Bit[31] - IDMA channels 4/5/6/7 read	0x1

Table 128: SDRAM Operation Mode, Offset: 0x474

Bits	Field Name	Function	Initial Value
2:0	SDRAMOp	Special SDRAM Mode Select 000 - Normal SDRAM Mode 001 - NOP Command 010 - All banks precharge command 011 - Mode register command enable 100 - CBR cycle enable 101,110,111 - Reserved	0x0
30:3	Reserved	Reserved.	0x0
31	Active	Active bit. Set by the DRAM controller after it performs the required transaction to DRAM bank.	0x0

Table 129: SDRAM Address Control, Offset: 0x47c

Bits	Field Name	Function	Initial Value
3:0	AddrSel	SDRAM Address Select Determines what address bits to drive on DAdr[12:0] and BankSel[1:0] during activate and command phases. **NOTE:** See Section 5.6.2 "SDRAM Address Control" on page 109.	0x2
31:4	Reserved	Reserved.	0x0

Table 130: SDRAM Timing Parameters, Offset: 0x4b4

Bits	Field Name	Function	Initial Value
1:0	CL	CAS Latency 0x1 - 2 cycles 0x2 - 3 cycles 0x3,0x0 - Reserved	0x2
3:2	Trp	SRAS Precharge Time 0x1 - 2 cycles 0x2 - 3 cycles 0x3,0x0 - Reserved	0x2
5:4	Trcd	SRAS to SCAS Delay 0x1 - 2 cycles 0x2 - 3 cycles 0x3,0x0 - Reserved	0x2
7:6	Reserved	Reserved.	0x0
11:8	Tras	Row Active Time. The minimum number of TClk cycles between activate and precharge cycles. 0x5-0x7 - Valid Tras values 0x0-0x4, 0x8-0xf - Reserved	0x5
12	RdDelay	Additional read data sampling stage. 0 - Disabled 1 - Enabled Reserved for Galileo Technology usage.	0x0
13	ECCEn	ECC Support 0 - No ECC support 1 - ECC supported	0x0

Table 130: SDRAM Timing Parameters, Offset: 0x4b4 (Continued)

Bits	Field Name	Function	Initial Value
14	RdSample	Number of pipe states in the DRAM read data path. 0 - Two pipe states (this is the default) 1 - One pipe stage. **NOTE:** When ECC is enabled, the DRAM controller has two pipe stages, regardless of this bit's setting.	0x0
31:13	Reserved	Reserved.	0x0

Table 131: SDRAM UMA Control, Offset: 0x4a4

Bits	Field Name	Function	Initial Value
7:0	L2HCnt	When configured as a UMA slave, used as a high priority request counter that determines after how many cycles, the request should be converted from low to high priority. **NOTE:** If set to 0, the request is never converted to high priority.	0x0
15:8	GntCnt	When configured as a UMA slave, used as a bus release counter that determines the number of cycles, after gaining bus ownership, that it must release the bus. Setting this counter to 0 means it releases the bus (deassert MREQ*) only when there are no pending SDRAM transactions.	0x1
16	UMAEn	UMA Enable 0 - Disable 1 - Enable **NOTE:** Two MPP pins must be configured to act as MREQ* and MGNT* in order to run UMA	Reset initialization
17	UMAMode	UMA Operation Mode 0 - UMA master 1 - UMA slave device	Reset initialization
31:18	Reserved	Reserved.	0x0

Table 132: SDRAM Interface Crossbar Control (Low), Offset: 0x4a8

Bits	Field name	Function	Initial Value
3:0	Arb0	Slice 0 of device controller “pizza” arbiter. 0x0 - NULL request 0x1 - Reserved 0x2 - CPU access 0x3 - PCI_0 access 0x4 - PCI_1 access 0x5 - Comm unit access 0x6 - IDMA channels 0/1/2/3 access 0x7 - IDMA channels 4/5/6/7 access 0x8 - 0xf - Reserved	0x2
7:4	Arb1	Slice 1 of device controller “pizza” arbiter.	0x3
11:8	Arb2	Slice 2 of device controller “pizza” arbiter.	0x4
15:12	Arb3	Slice 3 of device controller “pizza” arbiter.	0x5
19:16	Arb4	Slice 4 of device controller “pizza” arbiter.	0x6
23:20	Arb5	Slice 5 of device controller “pizza” arbiter.	0x7
27:24	Arb6	Slice 6 of device controller “pizza” arbiter.	0x0
31:28	Arb7	Slice 7 of device controller “pizza” arbiter.	0x0

Table 133: SDRAM Interface Crossbar Control (High), Offset: 0x4ac

Bits	Field name	Function	Initial Value
3:0	Arb8	Slice 8 of device controller “pizza” arbiter.	0x2
7:4	Arb9	Slice 9 of device controller “pizza” arbiter.	0x3
11:8	Arb10	Slice 10 of device controller “pizza” arbiter.	0x4
15:12	Arb11	Slice 11 of device controller “pizza” arbiter.	0x5
19:16	Arb12	Slice 12 of device controller “pizza” arbiter.	0x6
23:20	Arb13	Slice 13 of device controller “pizza” arbiter.	0x7
27:24	Arb14	Slice 14 of device controller “pizza” arbiter.	0x0
31:28	Arb15	Slice 15 of device controller “pizza” arbiter.	0x0

Table 134: SDRAM Interface Crossbar Timeout, Offset: 0x4b0
NOTE: Reserved for Galileo Technology usage.

Bits	Field name	Function	Initial Value
7:0	Timeout	Crossbar Arbiter Timeout Preset Value	0xff
15:8	Reserved	Reserved.	0x0
16	TimeoutEn	Crossbar Arbiter Timer Enable 0 - Enable 1 - Disable	0x1
31:17	Reserved	Reserved.	0x0

5.15.2 SDRAM Banks Parameters Registers

Table 135: SDRAM Bank0 Parameters, Offset: 0x44c

Bits	Field name	Function	Initial Value
13:0	Reserved	Reserved.	0x0
15:14	SDType	SDRAM type 0x1 - 16Mbit 0x2 - 64Mbit or 128Mbit 0x3 - 256Mbit or 512Mbit 0x0 - Reserved	0x3
16	OpenP0	Keeps virtual bank0 pages open. 0 - Page is closed at the end of an access. 1 - Page is kept open at the end of an access.	0x0
17	OpenP1	Keeps virtual bank1 pages open. 0 - Page is closed at the end of an access. 1 - Page is kept open at the end of an access.	0x0
18	OpenP2	Keeps virtual bank2 pages open. 0 - Page is closed at the end of an access. 1 - Page is kept open at the end of an access. **NOTE:** When using 16Mbit SDRAM (which means there are only two DRAM virtual banks), set OpenP2 to the same value as OpenP0.	0x0
19	OpenP3	Keeps virtual bank3 pages open. 0 - Page is closed at the end of an access. 1 - Page is kept open at the end of an access. **NOTE:** When using 16Mbit SDRAM (which means there are only two DRAM virtual banks), set OpenP3 to the same value as OpenP1.	0x0
31:20	Reserved	Reserved.	0x0

Table 136: SDRAM Bank1 Parameters, Offset: 0x450

Bits	Field Name	Function	Initial Value
19:0	Various	Same as SDRAM Bank0 Parameters.	0xc000
31:20	Reserved	Reserved.	0x0

Table 137: SDRAM Bank2 Parameters, Offset: 0x454

Bits	Field Name	Function	Initial Value
19:0	Various	Same as SDRAM Bank0 Parameters.	0xc000
31:20	Reserved	Reserved.	0x0

Table 138: SDRAM Bank3 Parameters, Offset: 0x458

Bits	Field Name	Function	Initial Value
19:0	Various	Same as SDRAM Bank0 Parameters.	0xc000
31:20	Reserved	Reserved.	0x0

5.15.3 SDRAM Error Report Registers

Table 139: SDRAM Error Data (Low), Offset: 0x484[1]

Bits	Field Name	Function	Initial Value
31:0	ECCData	Sampled 32 low bits of the last data with ECC error.	0x0

1. In case of multiple errors, only the first one is latched. New error report latching is enabled only after SDRAM Error Address register is being read

Table 140: SDRAM Error Data (High), Offset: 0x480

Bits	Field Name	Function	Initial Value
31:0	ECCData	Sampled 32 high bits of the last data with ECC error.	0x0

Table 141: SDRAM Error Address, Offset: 0x490

Bits	Field Name	Function	Initial Value
1:0	ErrType[1]	Error Type 00 - No errors 01 - One error detected and corrected 10 - Two or more errors detected 11 - Reserved	0x0
31:2	ECCAddr	Sampled address of the last data with ECC error.	0x0

1. In case of one or two errors detection, an interrupt is generated (if not masked). Write of 0x0 to ErrType, clears the interrupt.

Table 142: SDRAM Received ECC, Offset: 0x488

Bits	Field Name	Function	Initial Value
7:0	ECCRec	ECC code being read from SDRAM.	0x0
31:8	Reserved	Reserved.	0x0

Table 143: SDRAM Calculated ECC, Offset: 0x48c

Bits	Field Name	Function	Initial Value
7:0	ECCCalc	ECC code calculated by the GT–64240A.	0x0
31:8	Reserved	Reserved.	0x0

Table 144: SDRAM ECC Control, Offset: 0x494

Bits	Field Name	Function	Initial Value
7:0	ForceECC	User defined ECC byte written to the ECC bank.	0x0
8	ForceECC	Force user defined ECC byte on SDRAM writes. 0 - Write calculated ECC byte 1 - Write user defined ECC byte	0x0
9	ErrProp	Propagate Parity Errors to ECC Bank 0 - DRAM controller always generate correct ECC on write access to DRAM 1 - DRAM controller generates an uncorrectable ECC error (2 bits) on write access to DRAM, in case of parity error indication from the originating interface	0x0
15:10	Reserved	Reserved.	0x0

Table 144: SDRAM ECC Control, Offset: 0x494 (Continued)

Bits	Field Name	Function	Initial Value
23:16	ThrEcc	Threshold ECC Interrupt Number of single bit errors that occur before the GT–64240A generates an interrupt. **NOTE:** If set to 0x0, the GT–64240A does not generate an interrupt in case of a single bit error.	0x0
31:24	Reserved	Reserved.	0x0

Table 145: SDRAM ECC Counter, Offset: 0x498

Bits	Field Name	Function	Initial Value
31:0	Count	Number of single bit ECC errors detected. If the number of errors reaches 2^{32}, this register wraps around to 0x0	0x0

6. Address and Data Integrity

The GT–64240A supports address and data integrity on most of its interfaces.

- It supports parity checking and generation on the CPU and PCI busses
- It supports ECC checking and generation on the SDRAM bus
- CRC checking and generation on the Ethernet and Serial ports.

6.1 CPU Parity Support

The CPU interface generates and checks data parity.

On CPU writes, the GT–64240A samples data parity driven by the CPU with each data.

When a parity error occurs, the GT–64240A generates an interrupt and latches the following:

- Bad address in the CPU Error Address register.
- Data in the CPU Error Data register.
- Parity in the CPU Error Parity register.

On CPU reads, the GT–64240A drives parity with each read data it drives on the CPU bus.

NOTE: In case of multiple errors are detected, the address, data, and parity are latched in the corresponding registers only for the first error. Latching of new data into these registers is only enabled when reading the CPU Error Address (Low) register. The interrupt handler must read this register last.

6.2 SDRAM ECC

The GT–64240A implements Error Checking and Correction (ECC) on accesses to the SDRAM. It supports detection and correction of one data bit errors, detection of two errors, and detection of three or four bit errors within the same nibble.

6.2.1 ECC Calculation

Each of the 64 data bits and eight check bits has a unique 8-bit ECC check code, as shown in Table 146. For example, data bit 12 has the check value of 01100001, and check bit 5 has the check value of 00100000.

Table 146: ECC Code Matrix

Check Bit	Data Bit	ECC Code Bits 7	6	5	4	3	2	1	0	Number of 1s in syndrome
	63	1	1	0	0	1	0	0	0	3
	62	1	1	0	0	0	1	0	0	3
	61	1	1	0	0	0	0	1	0	3
	60	1	1	0	0	0	0	0	1	3

Table 146: ECC Code Matrix (Continued)

Check Bit	Data Bit	ECC Code Bits								Number of 1s in syndrome
		7	6	5	4	3	2	1	0	
	59	1	1	1	1	0	1	0	0	5
	58	1	0	0	0	1	1	1	1	5
4		0	0	0	1	0	0	0	0	1
3		0	0	0	0	1	0	0	0	1
	57	1	1	1	0	0	0	0	0	3
	56	1	0	1	1	0	0	0	0	3
	55	0	0	0	0	1	1	1	0	3
	54	0	0	0	0	1	0	1	1	3
	53	1	1	1	1	0	0	1	0	5
	52	0	0	0	1	1	1	1	1	5
5		0	0	1	0	0	0	0	0	1
2		0	0	0	0	0	1	0	0	1
	51	1	0	0	0	0	1	1	0	1
	50	0	1	0	0	0	1	1	0	3
	49	0	0	1	0	0	1	1	0	3
	48	0	0	0	1	0	1	1	0	3
	47	0	0	1	1	1	0	0	0	3
	46	0	0	1	1	0	1	0	0	3
	45	0	0	1	1	0	0	1	0	3
	44	0	0	1	1	0	0	0	1	3
	43	1	0	1	0	1	0	0	0	3
	42	1	0	1	0	0	1	0	0	3
	41	1	0	1	0	0	0	1	0	3
	40	1	0	1	0	0	0	0	1	3
	39	1	0	0	1	1	0	0	0	3
	38	1	0	0	1	0	1	0	0	3
	37	1	0	0	1	0	0	1	0	3
	36	1	0	0	1	0	0	0	1	3
	35	0	1	0	1	1	0	0	0	3

Table 146: ECC Code Matrix (Continued)

Check Bit	Data Bit	ECC Code Bits								Number of 1s in syndrome
		7	6	5	4	3	2	1	0	
	34	0	1	0	1	0	1	0	0	3
	33	0	1	0	1	0	0	1	0	3
	32	0	1	0	1	0	0	0	1	3
	31	1	0	0	0	1	0	1	0	3
	30	0	1	0	0	1	0	1	0	3
	29	0	0	1	0	1	0	1	0	3
	28	0	0	0	1	1	0	1	0	3
	27	1	0	0	0	1	0	0	1	3
	26	0	1	0	0	1	0	0	1	3
	25	0	0	1	0	1	0	0	1	3
	24	0	0	0	1	1	0	0	1	3
	23	1	0	0	0	0	1	0	1	3
	22	0	1	0	0	0	1	0	1	3
	21	0	0	1	0	0	1	0	1	3
	20	0	0	0	1	0	1	0	1	3
	19	1	0	0	0	1	1	0	0	3
	18	0	1	0	0	1	1	0	0	3
	17	0	0	1	0	1	1	0	0	3
	16	0	0	0	1	1	1	0	0	3
	15	0	1	1	0	1	0	0	0	3
	14	0	1	1	0	0	1	0	0	3
	13	0	1	1	0	0	0	1	0	3
	12	0	1	1	0	0	0	0	1	3
	11	1	1	1	1	1	0	0	0	5
	10	0	1	0	0	1	1	1	1	5
7		1	0	0	0	0	0	0	0	1
0		0	0	0	0	0	0	0	1	1
	9	0	1	1	1	0	0	0	0	3
	8	1	1	0	1	0	0	0	0	3

Table 146: ECC Code Matrix (Continued)

Check Bit	Data Bit	ECC Code Bits 7	6	5	4	3	2	1	0	Number of 1s in syndrome
	7	0	0	0	0	0	1	1	1	3
	6	0	0	0	0	1	1	0	1	3
	5	1	1	1	1	0	0	0	1	5
	4	0	0	1	0	1	1	1	1	5
6		0	1	0	0	0	0	0	0	1
1		0	0	0	0	0	0	1	0	1
	3	1	0	0	0	0	0	1	1	3
	2	0	1	0	0	0	0	1	1	3
	1	0	0	1	0	0	0	1	1	3
	0	0	0	0	1	0	0	1	1	3

The GT–64240A calculates ECC by taking the EVEN parity of ECC check codes of all data bits that are logic one. For example, if the 64 bit data is 0x45. The binary equivalent is 01000101. From Table 146, the required check codes are 00001101 (bit[6]), 01000011 (bit[2]) and 00010011 (bit[0]). Bitwise XOR of this check codes (even parity) result in ECC value of 01011101.

For error checking, GT–64240A reads 64-bits of data and 8-bits of ECC. It calculates ECC based on the 64-bit data and then compares it against the received ECC. The result of this comparison (bitwise XOR between received ECC and calculated ECC) is called the syndrome.

If the syndrome is 00000000, both the received data and ECC are correct.

If the syndrome is any other value, the GT–64240A assumes either the received data or the received ECC are in error.

If the syndrome contains a single ‘1’, there is a single bit error in the ECC byte. For example, if the received data is 0x45, the calculated ECC is 01011101, as explained before. If the received ECC is 01010101, the resulting syndrome is 00001000. Table 146 shows that this syndrome corresponds to check bit 3. The GT–64240A does not report or correct this type of error.

If the syndrome contains three or five ‘1’s, it indicates that there is at least one data bit error. For example, if the received data is 0x45, the calculated ECC is 01011101, as explained before. If the received ECC is 00011110, the resulting syndrome is 01000011. This syndrome includes three ‘1’s and it corresponds to data bit 2 as shown in Table 146. In this case, the GT–64240A corrects the data by inverting data bit 2 (the corrected data is 0x41).

If the result syndrome contains two ‘1’s, it indicates that there is a double-bit error.

If the result syndrome contains four ‘1’s, it indicates a 4-bit error located in four consecutive bits of a nibble.

If the result syndrome contains five ‘1’s, and no four of the ‘1’s are contained in check bits [7:4] or check bits [3:0] (which means it does not correspond to any data bit of the table), it indicates a triple-bit error within a nibble.

NOTE: These types of errors cannot be corrected. The GT–64240A reports an error but will not change the data.

6.2.2 SDRAM Interface Operation

On SDRAM reads, the GT–64240A reads the ECC byte with the data, calculates the ECC byte, and compares it against the read ECC byte. In case of a single bit error, it corrects the error and drives the correct data to the initiating interface. In case of two errors detection (or 3 or 4 errors that resides in the same nibble), it only reports an error, see section 6.2.3.

On a write transaction, the GT–64240A calculates the new ECC and writes it to the ECC bank, with the data that is written to the data bank. Since the ECC calculation is based on a 64-bit data width, if the write transaction is smaller than 64 bits, the GT–64240A runs a read modify write (RMW) sequence. It reads the full 64-bit data, merges the incoming data with the read data, and writes the new data back to SDRAM bank with new ECC byte.

NOTE: If identifying a non-correctable error during the read portion of the RMW sequence, the GT–64240A writes the data back to DRAM with a non-correctable ECC byte (it calculates a new ECC byte and than flips two bits). This behavior guarantees that the error is still visible if there is a future read from this DRAM location.

RMW is performed on 64-bit data basis. In case of a burst to DRAM, only data which not all of its byte enables are active require RMW. For example, a burst write from a 32-bit PCI bus of five 32-bit words to address 0x0 in DRAM, results in burst write of three 64-bit words to DRAM, in which only the third data has byte enable inactive (be = 0xf0). In this case, only the third data requires RMW.

The GT–64240A also supports forcing bad ECC written to the ECC bank for debug purposes. If this mode is enabled, rather than calculating the ECC to be written to the ECC bank, it drives a fixed ECC byte configured in SDRAM ECC Control register, Table 144 on page 131.

SDRAM interface also contains a 32-bit ECC error counter that counts the number of corrected, single bit errors that are detected. Use software to reset the ECC error counter.

6.2.3 ECC Error Report

In case of ECC error detection, the GT–64240A asserts an interrupt (if not masked), and latches the:

- Address in the ECC Error Address register.
- 64-bit read data in the ECC Error Data register.
- Read ECC byte in the SDRAM ECC register.
- Calculated ECC byte in the Calculated ECC register.

NOTE: For more information about these registers, see Section 5.15.3 “SDRAM Error Report Registers” on page 130.

The GT–64240A reports an ECC error whenever it detects but cannot correct an error (2, 3, or 4 bits errors).

The GT–64240A also reports on single bit errors (correctable errors), based on the setting of the ECC threshold, bits [23:16], in the ECC Control register, see Table 144 on page 131.

- If the threshold is set to ‘0’, there is no report on single bit errors.
- If set to ‘1’, GT–64240A reports each single bit error.
- If set to ‘n’, GT–64240A reports each ‘n’ single bit error.

NOTE: In case of multiple errors detection, the address, data, and ECC are latched in the corresponding registers only for the first error. Latching of new data into these registers is enabled only when reading ECC Error Address register. The interrupt handler must read this register last.

6.3 Parity Support for Devices

There is no dedicated logic in the GT–64240A to support parity on the device bus. If devices parity checking is required, use external logic. In order to generate an interrupt in case of bad device parity detection, use the GPP inputs (see Section 18.3 “GPP Interrupts” on page 477).

6.4 PCI Parity Support

The GT–64240A implements all parity features required by the PCI spec, including PAR, PAR64*, PERR*, and SERR* generation and checking.

As an initiator, the GT–64240A generates even parity on PAR signals for write transaction’s address and data phases. It samples PAR on data phase of read transactions.

NOTE: If the GT–64240A detects bad parity and the Status and Command Configuration register’s PErrEn bit is set (see Table 291 on page 234), it asserts PERR*.

As a target, the GT–64240A generates even parity on PAR signals for a read transaction’s data phase. It samples PAR on the address phase and data phase of write transactions.

In all of the parity errors conditions, the GT–64240A generates an interrupt (if not masked) and latches the:

- Address in the PCI Error Address register
- Data in PCI Error Data register
- Command, byte-enable, and parity in the PCI Error Command register

If the PCI Status and Command configuration register’s SErrEn bit is set to ‘1’ and enabled via the SERR Mask register (see Table 274 on page 225), the GT–64240A may also assert SERR*. If any of the parity errors conditions occurs, SERR* is asserted.

NOTE: In case of multiple errors detection, address, data and parity are latched in the corresponding registers only for the first error. Latching of new data into these registers is enabled only when reading PCI Error Address (Low) register. The interrupt handler must read this register last.

6.5 Communication Ports Data Integrity

The GT–64240A supports CRC on the Ethernet and MPSC ports. For full details, see Section 13. “10/100Mb Ethernet Unit” on page 339 and Section 14. “Multi Protocol Serial Controller (MPSC)” on page 397.

6.6 Parity/ECC Errors Propagation

Although each interface includes the required logic to detect and report parity/ECC errors, this is sometimes inadequate, due to the latency of interrupt routines.

For example, bad parity is detected on a PCI write to SDRAM. In the time required for the CPU interrupt handler to handle the interrupt, the bad data may be read by the CPU.

To guarantee this scenario does not occur, propagate the bad PCI parity to SDRAM as a non-correctable ECC error. This guarantees that once the CPU reads this data, it recognizes it as erroneous data.

In case of a write access to SDRAM with bad parity indication, the SDRAM interface can force two ECC errors to the ECC bank. If ErrProp bit in the ECC Control register is set to ‘1’, the GT–64240A calculates the new ECC byte and flips two bits before writing it to the ECC bank.

In case of a CPU read from SDRAM that results in ECC error detection (but no correction), or a CPU read from PCI that results in parity error, the GT–64240A generates an interrupt. It also drives Erroneous Data bit (SysCmd[5]) to the CPU. The CPU interface can be also configured to force bad parity in this case. If PerrProp bit in the CPU Configuration register is set to ‘1’, the GT–64240A calculates data parity and flips all the bits when driving it on the CPU bus.

In case of PCI reads from SDRAM that results in ECC error detection (but no correction), or in any case of CPU or IDMA write to PCI with bad ECC/parity indication, the PCI interface can force bad parity on the bus. If PErrProp bit in PCI Command register is set to ‘1’, the GT–64240A calculates data parity and flips the value it drives on PAR.

7. DEVICE CONTROLLER

The device controller supports up to five banks of devices. Each bank's supported memory space can be programmed separately in 1Mbyte quantities up to 512Mbyte of address space, resulting in total device space of 2.5Gbyte.

Each bank has its own parameters register. Bank width can be programmed to 8-, 16-, or 32-bits. Bank timing parameters can be programmed to support different device types (e.g. Sync Burst SRAM, Flash, ROM, I/O Controllers).

The five individual chip selects are typically separated into four individual device banks and one chip select for a boot device. The boot device bank is the same as any of the other banks except that it's default address map matches the MIPS CPU boot address (0x1fc0.0000) and that it's default width is sampled at reset.

The device AD bus is a 32-bit multiplexed address/data bus. During the address phase, the device controller puts an address on the AD bus with a corresponding chip select asserted and DevRW indicated. It deassserts Address Latch Enable (ALE) to latch the address, the chip select, and read/write signals by an external latch (or register).

CS* must then be qualified with CSTiming* to generate the specific device chip select and DevRW* must be qualified with CSTiming* to generate a read or write cycle indication. The CSTiming* signal is active for the entire device access time specified in the device timing parameters register.

During the data phase, the device controller drives data on the AD bus, in case of write cycle, or samples data driven by the device, in case of read cycle. Use Wr[3:0]* as the byte enable signal during a write transaction.

NOTE: The GT–64240A does not support READ byte enables.

The GT–64240A does not support multiple masters on the AD bus or access to the different GT–64240A interfaces via the device bus.

All device controller signals, including CSTiming*, are floated for the entire reset assertion period and an additional five TClk cycles after reset deassertion. Since the device chip select is qualified with CSTiming*, this signal must be pulled up or driven for the five additional cycles by some external logic, to prevent undesired accesses to the device.

7.1 Device Controller Implementation

The device interface consists of 128 bytes of write buffer and 128 bytes of read buffer. It can absorb up to four read plus four write transactions.

On a write transaction to a device, the data is written to the write buffer and then driven to the device bus. As soon as a device access is requested, the device controller drives an address on the AD bus for two cycles and deasserts ALE, so it will be used by external logic to latch the address, chip select, and DevRW* indication.

NOTE: The CS* must be qualified by the CSTiming* signal to generate the device's actual chip select.

On the next cycle after ALE deassertion, the device controller pops data from the write buffer and drives it on the bus. It drives the valid data based on the device timing parameters, see 7.2.

In case the device controller is still serving a previous transaction on the bus, the whole burst write is posted into the write buffer and driven to the device bus when all the previous transactions are completed.

On a read transaction, the device controller samples the read data from the AD bus. The sample window is determined according to the device timing parameters, see 7.2. When the whole read data is placed in the read buffer, it is driven back to the requesting interface.

7.2 Device Timing Parameters

To allow interfacing with very slow devices and fast synchronous SRAMs, each device can be programed to different timing parameters.

7.2.1 TurnOff

The TurnOff parameter defines the number of TClk cycles that the GT–64240A does not drive the AD bus after the completion of a device read. This prevents contentions on the device bus after a read cycle from a slow device. The minumum setting is 0x1.

7.2.2 Acc2First

The Acc2First parameter defines the number of TClk cycles from the assertion of ALE to the cycle that the first read data is sampled by GT–64240A. The minumum setting is 0x3.

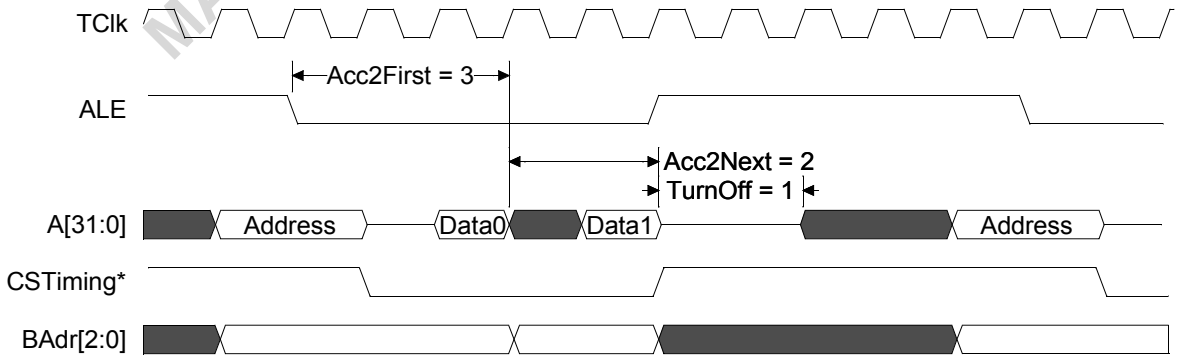
NOTE: Extend this parameter by extending the Ready* pin, see 7.4.

7.2.3 Acc2Next

The Acc2Next parameter defines the number of TClk cycles between the cycle that samples the first read data by GT–64240A to the cycle that samples the next data (in burst accesses). Extend this parameter can be extended by the Ready* pin, see 7.4. The minumum setting is 0x1.

Figure 24 shows a device read timing parameters example.

Figure 24: Device Read Parameters Example



7.2.4 ALE2Wr

The ALE2Wr parameter defines the number of TClk cycles from ALE deassertion cycle to Wr[3:0]* assertion. The minumum setting is 0x3.

7.2.5 WrLow

The WrLow parameter defines the number of TClks that Wr[3:0]* is active (low). Extend this parameter by the Ready* pin, see 7.4. BAdr and Data are kept valid for the whole WrLow period. This parameter defines the setup time of address and data to Wr rise. The minumum setting is 0x1.

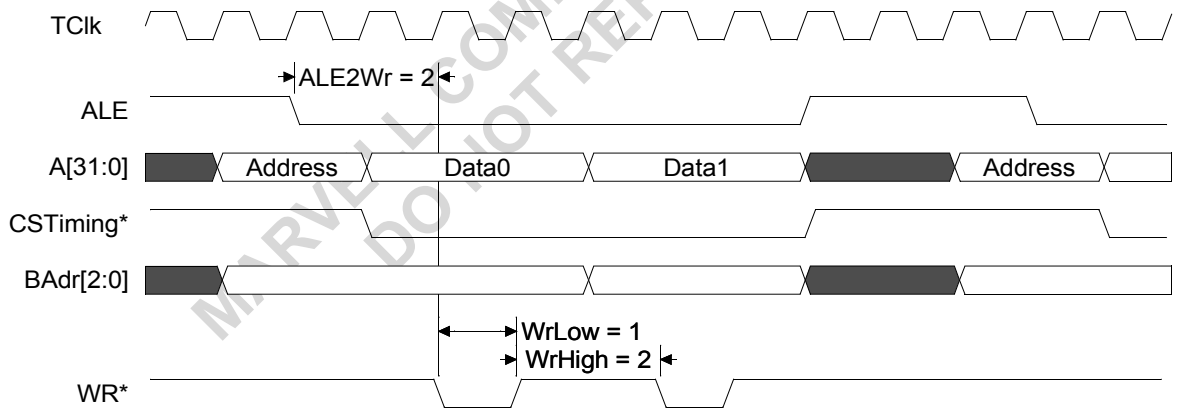
7.2.6 WrHigh

The WrHigh parameter defines the number of TClk cycles that Wr[3:0]* is kept inactive (high) between data beats of a burst write. BAdr and Data are kept valid (don't toggle) for WrHigh-1 period, with the exceptions of WrHigh values of '0' or '1'. This parameter defines the hold time of address and data after Wr rise. The minumum setting is 0x0.

NOTE: Programing WrHigh to '0' is only used for zero wait states burst access (e.g. sync burst SRAM access). It is only allowed when WrLow is set to 1.

Figure 25 shows a device write timing parameters example.

Figure 25: Device Write Parameters Example

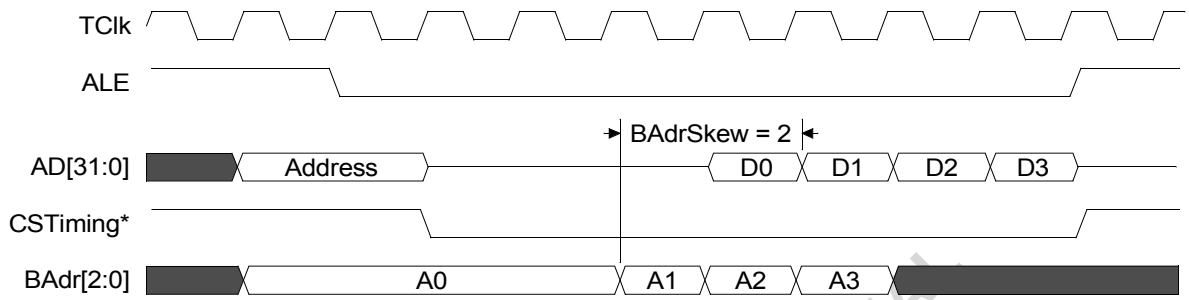


7.2.7 BAdrSkew

The GT–64240A also supports early toggle of burst address during read access. The Device Bank Parameters register's BAdrSkew bits [29:28] (see Table 152 on page 149) defines the number of TClk cycles from BAdr toggle to read data sample. This parameter is usefull for SyncBurst SRAM type of devices, where the address precedes the read data by one (Flow Through SRAM) or two (Pipelined SRAM) cycles.

Figure 26 shows a BAdrSkew usage example.

Figure 26: Pipeline Sync Burst SRAM Read Example



7.3 Data Pack/Unpack and Burst Support

The device controller supports 8-, 16-, or 32-bit wide devices. Specify the device width in the DevWidth[21:20] field of each device parameters register.

The device controller supports up to 32 byte burst to a 32-bit wide device, and up to 8 bytes burst to 8- or 16-bit wide device. The burst address is supported by a dedicated three bit BAdr[2:0] bus. This bus must be connected directly to the device address bus (not like the latched address on the multiplexed AD bus). The device controller supports pack/unpack of data between the device (8-, 16-, or 32-bit wide) and the initiator (PCI, CPU, DMA).

An attempt to access a device with a non-supported burst results in an interrupt assertion.

NOTE: Since bursts to 8- and 16-bit devices are limited to eight bytes, never place these devices in a CPU cacheable region (that requires bursts of 32 bytes). Also, it is only possible to read these devices from a PCI's non-prefetchable region.

Since bursts to 32-bit devices are limited to 32 bytes, DMA or PCI accesses to such devices must not exceed 32 bytes. This means that the PCI Mburst must be set to 32 bytes (see Table 247 on page 217); the IDMA BurstLimit must not exceed 32 bytes (see Table 437 on page 301); the Ethernet SDMA BSZ Burst is limited to 4 64bit words (see Table 563 on page 382); and, the MPSC's SDMA BSZ is limited to 4 64bit words (Table 626 on page 462.

The device controller does not support non-sequential byte enables to 8 or 16-bit wide devices (e.g. write of 32-bit word to 8-bit wide device with byte enable 1010).

On burst read access to a 32-bit device, the device controller can return read data to the requester as soon as first 64-bit data is available, or only when the whole burst data is available. If the Device Interface Control register's RdTrig bit [16] is set to '1', data is returned to the requester, only when the whole burst data is valid (store and forward policy). This is useful when interfacing with a device that has long wait states between data beats, to avoid wasting the GT–64240A cross bar bandwidth. If RdTrig is set to '0', data is returned as soon as packed 64-bit data is valid.

7.4 Ready* Support

Ready* input is used to extend the programable device timing parameters. This is usefull for two cases:

- Interfacing a very slow device, which has access time greater than the maximum programable values.
- Interfacing a device with a non-deterministic access time (access time depends on other systm events and activity).

Ready* can extend the following timing parameters:

- Acc2First
- Acc2Next
- WrLow

During a read access, the device controller is first counting TClk cycles based on Acc2First programable parameters (see Table 152 on page 149). If at the time Acc2First expires and Ready* input is not asserted, the device controller keeps waiting until Ready* is sampled asserted, and only then samples first read data. Similarly, if Acc2Next expires and Ready* is not asserted, the device controller waits until Ready* is sampled asserted, and only then samples next read data. On a write access, if at the time WrLow is expired, Ready* input is not asserted, it keeps driving write data until Ready* is sampled asserted Figure 27, Figure 28, and Figure 29 show examples of the Ready* operation.

NOTE: If Ready* is not used, Ready* pin must be tied low.

If the WrLow or WrHigh timing parameter is set to ’0’, Ready* is not supported during a write access

When interfacing a device with a non-deterministic access time, timing parameters must be set to the minimum values, and the actual access time is controlled via the Ready* pin

To prevent system hang due to a lack of Ready* assertion, the GT–64240A implements a programable timer that allows termination of a device access even without Ready* assertion. If during a device access the timeout timer expires, the device controller completes the transaction as if Ready* was asserted and generates an interrupt. Setting the timer to 0x0 disables it, and the device controller waits for Ready* forever.

NOTE: The timer is used only for preventing system hang due to a lack of Ready* pin assertion. If expired (which means a system hardware problem), the device controller completes the transaction ignoring Ready*. This might result in bad data read/write from/to the device. The timer must be programed to a number that must never be exceeded in normal operation.

Figure 27: Ready* Extending Acc2First

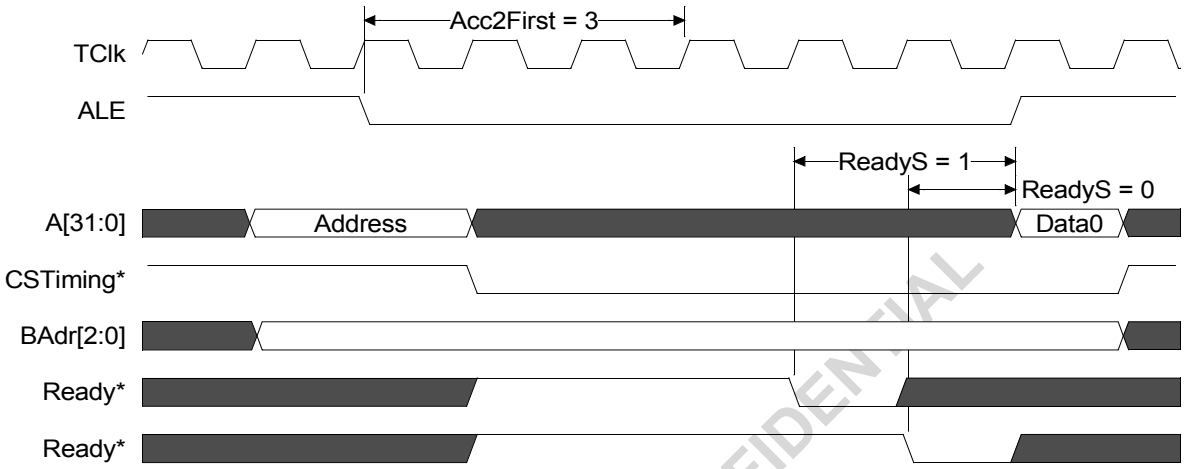


Figure 28: Ready* Extending Acc2Next

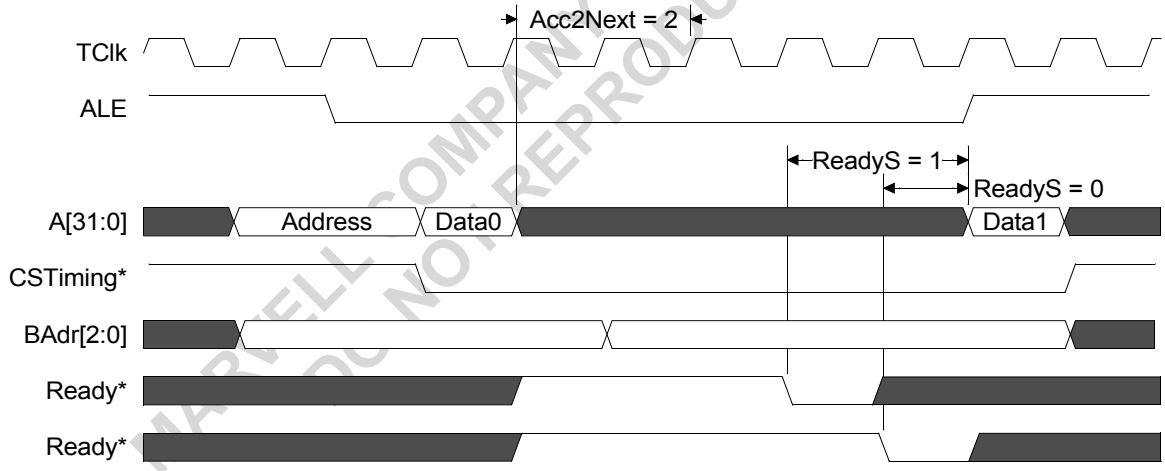
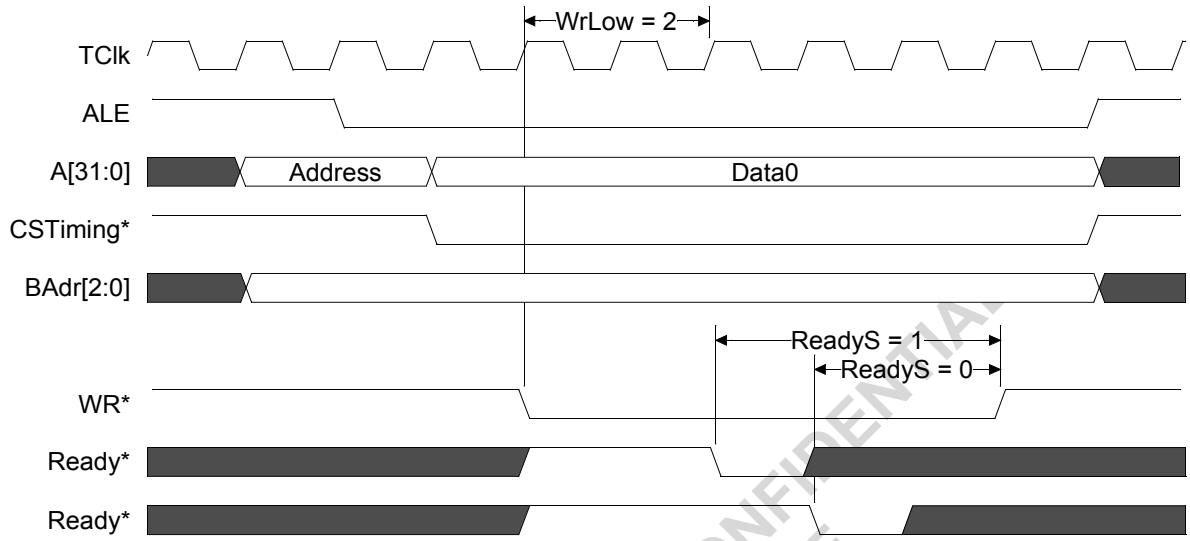


Figure 29: Ready* Extending WrLow Parameter



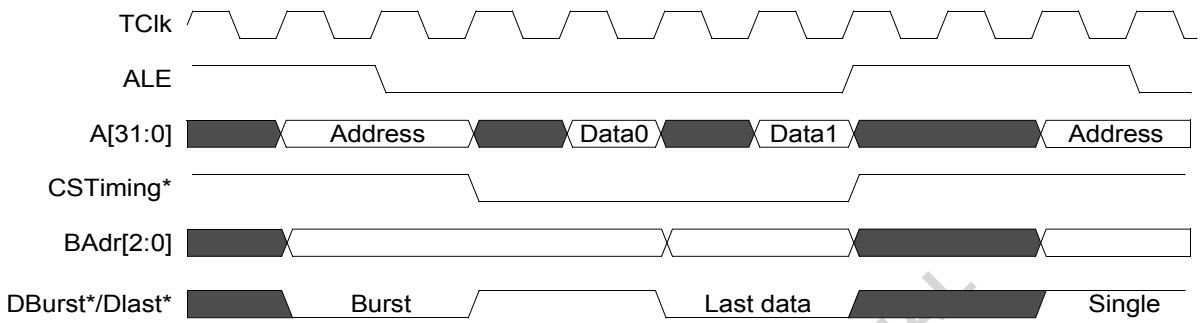
The Device Interface Control register's ReadyS bit [19] determines the Ready* input sample window, see Table 157 on page 151. If set to '1', the device controller samples read data two cycles after Ready* assertion on a read access, and de-asserts Wr* two cycles after Ready* assertion on a write access. If set to '0', the device controller samples read data one cycle after Ready* assertion, and toggles Wr* one cycle after Ready* assertion, as shown in the above figures.

NOTE: Ready* input setup time, is defined in Section 27. "AC Timing" on page 538, for the case of ReadyS set to '1'. The input setup is 1.5ns greater in the case of ReadyS set to '0'.

7.5 Additional Device Interface Signaling

To make it easy to glue external logic on the device bus, the GT–64240A supports burst and last indication via MPP lines. DBurst*/DLast* is driven low on the address phase (need to be latched via ALE*) to indicate a burst access and is driven low on the last data phase to indicate the last data transfer. Figure 30 shows an example.

Figure 30: DBurst*/Dlast* Example



7.6 Error Report

In case of a device access error condition, the Device Interrupt Cause register registers an interrupt. Also, the address of the device access is registered in the Device Error Address register.

7.7 Interfacing With 8/16/32-Bit Devices

To connect the devices correctly, follow the pin connection information listed in the following tables.

Table 147: 8-bit Devices

Connection	Connect...	To...
Device Address	BAdr[2:0] AD[27:2] ALE Latch Outputs	Device Address Bits [2:0] Address Latch Inputs Address LE Device Address Bits [28:3]
Device Data	AD[7:0]	Device Data Bits [7:0]
Device Control Pins	ALE AD[1] AD[0] AD[31:28]	Control latch LE Becomes DevRW* Becomes BootCS* Becomes CS[3:0]*
Write Strobes	Wr[0]*	Device Data Bits[7:0] Write Strobe

Table 148: 16-bit Devices

Connection	Connect...	To...
Device Address	BAdr[2:0] AD[27:3] ALE Latch Outputs	Device Address Bits[2:0] Address Latch Inputs Address LE Device Address Bits [27:3]
Device Data	AD[15:0]	Device Data Bits [15:0]
Device Control Pins	ALE AD[1] AD[0] AD[31:28]	Control latch LE Becomes DevRW* Becomes BootCS* Becomes CS[3:0]*
Write Strobes	Wr[0]* Wr[1]*	Device Data Bits[7:0] Write Strobe Device Data Bits[15:8] Write Strobe

Table 149: 32-bit Devices

Connection	Connect...	To...
Device Address	BAdr[2:0] AD[27:4] ALE Latch Outputs	Device Address Bits [2:0] Address Latch Inputs Address LE Device Address Bits [26:3]
Device Data	AD[31:0]	Device Data Bits [31:0]
Device Control Pins	ALE AD[1] AD[0] AD[31:28]	Control latch LE Becomes DevRW* Becomes BootCS* Becomes CS[3:0]*
Write Strobes	Wr[0]* Wr[1]* Wr[2]* Wr[3]*	Device Data Bits[7:0] Write Strobe Device Data Bits[15:8] Write Strobe Device Data Bits[23:16] Write Strobe Device Data Bits[31:24] Write Strobe

7.8 Device Interface Registers

Table 150: Device Control Register Map

Register	Offset	Page
Device Bank0 Parameters	0x45c	page 149
Device Bank1 Parameters	0x460	page 150

Table 150: Device Control Register Map (Continued)

Register	Offset	Page
Device Bank2 Parameters	0x464	page 150
Device Bank3 Parameters	0x468	page 151
Boot Device Parameters	0x46c	page 151
Device Interface Control	0x4c0	page 151
Device Interface Crossbar Control (Low)	0x4c8	page 152
Device Interface Crossbar Control (High)	0x4cc	page 152
Device Interface Crossbar Timeout	0x4c4	page 153

Table 151: Device Interrupts Register Map

Register	Offset	Page
Device Interrupt Cause	0x4d0	page 153
Device Interrupt Mask	0x4d4	page 153
Device Error Address	0x4d8	page 154

7.8.1 Device Control Registers

Table 152: Device Bank0 Parameters, Offset: 0x45c

Bits	Field Name	Function	Initial Value
2:0	TurnOff	The number of cycles in a read access between the deassertion of CSTiming* to a new device bus cycle.	0x7
6:3	Acc2First	The number of cycles in a read access between the assertion of ALE to the cycle that the first data is sampled by the GT–64240A.	0xf
10:7	Acc2Next	The number of cycles in a burst read access between the cycle that the first data is sampled by the GT–64240A to the cycle that the next data is sampled.	0xf
13:11	ALE2Wr	The number of cycles in a write access from the ALE deassertion to the assertion of Wr*.	0x7
16:14	WrLow	The number of cycles in a write access that the Wr* signal is kept active. **NOTE:** If WrLow is set to ’0’, Ready* is not supported.	0x7
19:17	WrHigh	The number of cycles in a burst write access that the Wr* signal is kept deasserted. **NOTE:** If WrHighis set to ’0’, Ready* is not supported.	0x7

Table 152: Device Bank0 Parameters, Offset: 0x45c (Continued)

Bits	Field Name	Function	Initial Value
21:20	DevWidth	Device Width 00 - 8 bits 01 - 16 bits 10 - 32 bits 11 - Reserved	0x2 For the boot device width, these bits are sampled by AD[15:14] at reset.
22	TurnOffExt	TurnOff Extention The MSB of the TurnOff parameter.	0x1
23	Acc2FirstExt	Acc2First Extention The MSB of the Acc2First parameter.	0x1
24	Acc2NextExt	Acc2Next Extention The MSB of the Acc2Next parameter.	0x1
25	ALE2WrExt	ALE2Wr Extention The MSB of the ALE2Wr parameter.	0x1
26	WrLowExt	WrLow Extention The MSB of the WrLow parameter.	0x1
27	WrHighExt	WrHigh Extention The MSB of the WrHigh parameter.	0x1
29:28	BAdrSkew	Cycles gap between BAdr toggle to read data sample. Useful when interfacing Sync Burst SRAM 0x0 - No gap (default setting) 0x1 - One cycle gap 0x2 - Two cycle gaps 0x3 - Reserved	0x0
31:30	Reserved	Reserved.	0xf

Table 153: Device Bank1 Parameters, Offset: 0x460

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Device Bank0.	0xffefffff

Table 154: Device Bank2 Parameters, Offset: 0x464

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Device Bank0.	0xffefffff

Table 155: Device Bank3 Parameters, Offset: 0x468

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Device Bank0.	0xffefffff

Table 156: Boot Device Bank Parameters, Offset: 0x46c

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Device Bank0.	0xff?fffff[1]

1. The boot device width (bits[21:20]) are sampled by AD[15:14] at reset.

Table 157: Device Interface Control, Offset: 0x4c0

Bits	Field Name	Function	Initial Value
15:0	Timeout	Timeout Timer Preset Value. If the device access is not completed within this preset value's period (due to a lack of Ready* assertion), the device controller completes the transaction as if Ready* was asserted and asserts an interrupt. **NOTE:** If set to 0x0, the device controller waits for Ready* assertions forever.	0xffff
16	RdTrig	Read Trigger Control 0 - Drives the read data to the requesting unit only after the last data arrives from the device. 1 - Drives the read data to the requesting unit as soon as the first 64-bit data arrives from the device.	0x1
17	Reserved	Must be 1	0x1
18	ReadyS	Ready* input sampling window 0 - Read data is sampled one cycle after Ready* is asserted. Wr* is deasserted one cycle after Ready* is asserted. 1 - Read data is sampled two cycles after Ready* is asserted. Wr* is deasserted two cycles after Ready* is asserted.	0x1
31:19	Reserved	Reserved.	0x0

Table 158: Device Interface Crossbar Control (Low), Offset: 0x4c8

Bits	Field Name	Function	Initial Value
3:0	Arb0	Slice 0 of the device controller “pizza” arbiter. 0x0 - Reserved 0x1 - NULL request 0x2 - CPU access 0x3 - PCI_0 access 0x4 - PCI_1 access 0x5 - Comm unit access 0x6 - IDMA channels 0/1/2/3 access 0x7 - IDMA channels 4/5/6/7 access 0x8 - 0xf - Reserved	0x2
7:4	Arb1	Slice 1 of the device controller “pizza” arbiter.	0x3
11:8	Arb2	Slice 2 of the device controller “pizza” arbiter.	0x4
15:12	Arb3	Slice 3 of the device controller “pizza” arbiter.	0x5
19:16	Arb4	Slice 4 of the device controller “pizza” arbiter.	0x6
23:20	Arb5	Slice 5 of the device controller “pizza” arbiter.	0x7
27:24	Arb6	Slice 6 of the device controller “pizza” arbiter.	0x1
31:28	Arb7	Slice 7 of the device controller “pizza” arbiter.	0x1

Table 159: Device Interface Crossbar Control (High), Offset: 0x4cc

Bits	Field Name	Function	Initial Value
3:0	Arb8	Slice 8 of the device controller “pizza” arbiter.	0x2
7:4	Arb9	Slice 9 of the device controller “pizza” arbiter.	0x3
11:8	Arb10	Slice 10 of the device controller “pizza” arbiter.	0x4
15:12	Arb11	Slice 11 of the device controller “pizza” arbiter.	0x5
19:16	Arb12	Slice 12 of the device controller “pizza” arbiter.	0x6
23:20	Arb13	Slice 13 of the device controller “pizza” arbiter.	0x7
27:24	Arb14	Slice 14 of the device controller “pizza” arbiter.	0x1
31:28	Arb15	Slice 15 of the device controller “pizza” arbiter.	0x1

Table 160: Device Interface Crossbar Timeout, Offset: 0x4c4
NOTE: Reserved for Galileo Technology usage.

Bits	Field Name	Function	Initial Value
7:0	Timeout	CrossBar Arbiter Timeout Preset Value	0xff
15:8	Reserved	Reserved.	0x0
16	TimeoutEn	CrossBar Arbiter Timer Enable 0 - Enable 1 - Disable	0x1
31:17	Reserved	Reserved.	0x0

7.8.2 Device Interrupts

Table 161: Device Interrupt Cause, Offset: 0x4d0[1]

Bits	Field name	Function	Initial Value
0	DBurstErr	Burst violation An attempt to burst more data than device controller is capable of handling.	0x0
1	DRdyErr	Ready Timer Expired.	0x0
26:2	Reserved	Reserved.	0x0
27	Sel	Specifies the error event currently being reported in the Error Address register. 0 - DBurstErr 1 - DRdyErr Read Only.	0x0
31:28	Reserved	Reserved.	0x0

1. All cause bits are clear only. They are set upon error condition cleared upon a value write of '0'. Writing a value of '1' has no affect.

Table 162: Device Interrupt Mask, Offset: 0x4d4

Bits	Field name	Function	Initial Value
0	DBurstErr	If set to '1', enables DBurstErr interrupt.	0x0
1	DRdyErr	If set to '1', enables DRdyErr interrupt.	0x0
31:2	Reserved	Reserved.	0x0

Table 163: Device Error Address, Offset: 0x4d8

Bits	Field name	Function	Initial Value
31:0	Addr	Latched Address Upon Device Error Condition After the address is latched, no new address is latched (due to additional error condition) until the register is being read.	0x0

8. PCI INTERFACE

The GT–64240A supports two 64-bit PCI interfaces, compliant to PCI specification rev. 2.2.

NOTE: When configured as a 32-bit bus, the GT–64240A drives PAD0/1[63:32], CPE0/1[7:4], PAR640/1 pins; a pull-up is not required.

8.1 PCI Master Operation

When the CPU, IDMA, Communication unit, or the other PCI interface initiates a bus cycle to the PCI, the PCI master translates the cycle into the appropriate PCI bus transaction. The transaction address is the same as the initiator cycle address, unless address remapping is used.

The GT–64240A PCI master supports the following transactions:

- Memory Read
- Memory Write
- Memory Read Line
- Memory Read Multiple
- Memory Write & Invalidate
- I/O Read
- I/O Write
- Configuration Read
- Configuration Write
- Interrupt Acknowledge
- Special Cycle
- Dual Address Cycles

The GT–64240A PCI master generates a Memory Write and Invalidate transaction if:

- The transaction accessing the PCI memory space requests a data transfer size equal to multiples of the PCI cache line size.
- The start address is cache line aligned.
- the PCI Status and Command register's MemWrInv bit is set, see Table 291 on page 234

The GT–64240A PCI master generates a Memory Read Line transaction if:

- The transaction accessing the PCI memory space requests a data transfer size equal to multiples of the PCI cache line size.
- The start address is cache line aligned.

A Memory Read Multiple transaction is carried out when the transaction accessing the PCI memory space requests a data transfer that crosses the PCI cache line size boundary.

NOTE: The GT–64240A supports only cache line size of eight (8 32-bit words). Setting the PCI cache line register to any other value is treated as if cache line size is set to '0'.

Dual Address Cycles (DAC) transaction is carried out if the requested address is beyond 4Gbyte (address bits[63:32] are not '0').

The master consists of 512 bytes of posted write data buffer and 512 bytes of read buffer. It can absorb up to four write transactions plus four read transactions. The PCI master posted write buffer in the GT–64240A permits the CPU to complete CPU-to-PCI memory writes even if the PCI bus is busy. The posted data is written to the target PCI device when the PCI bus becomes available. The read buffer absorbs the incoming data from PCI. Read and Write buffers implementation guarantees that there are no wait states inserted by the master

NOTE: IRDY* is never deasserted in the middle of a transaction.

8.1.1 PCI Master Write Operation

On a write transaction, data from the initiator unit is first written to the master write buffer and then driven on the PCI bus. The master does not need to wait for the write buffer to be full. It starts driving data on the bus when the first data is written into the write buffer or only when the whole burst is placed in the buffer. This depends on the MWrTrig bit setting in the PCI Command register, see Table 233 on page 206.

On consecutive write transactions, the transactions are placed into the queue. When the first transaction is done, the master initiates the transaction for the next transaction in the queue.

The master supports combining memory writes. This is especially useful for long DMA transfers, where a long burst write is required. If combining is enabled through the MWrCom bit in PCI Command register, the master combines consecutive write transactions, if possible. For combining memory writes to occur, the following conditions must exist:

- Combining is enabled through the PCI Command register's MWrCom bit, see Table 233 on page 206.
- The start address of the second transaction matches the address of data n+1 of the first transaction.
- While the first transaction is still in progress, the request for the new transaction occurs.

The master supports fast back-to-back transactions. If there is a pending new transaction in the middle of a transaction in progress, the master starts the new transaction after the first transaction ends, without inserting dead cycle. For the master to issue a fast back-to-back transaction, the following conditions must exist:

- Fast back-to-back is enabled (bit[9] of Status and Command register is set to 1), see Table 291 on page 234.
- The first transaction is a write.
- While the first transaction is still in progress, the new transaction request occurs.

8.1.2 PCI Master Read Operation

On a read transaction, when the initiator requests a PCI read access, the PCI master drives the transaction on the bus (after gaining bus mastership). The returned data is written into read buffer. The PCI master drives the read data to the initiating unit as soon as the data arrives from the PCI bus or when the whole burst read is placed in the read buffer. This action depends on the setting of the MRdTrig bit in PCI Command register, see Table 233 on page 206.

NOTE: In case of a CPU burst read cache line read, regardless of RdTrig bit setting, the master absorbs the full burst into the read buffer and only then drives it to the CPU interface unit in sub-lock order.

The master also supports combining read transactions. This is especially useful for long DMA transfers, where a long burst read is required, and the PCI target drives long burst data without inserting wait states. If combining is enabled through MRdCom bit in PCI Command register, the master combines consecutive read transactions. For combining read transactions to occur, the following conditions must exist:

- Combining is enabled.
- The start address of the second transaction matches the address of data n+1 of the first transaction.
- While the first transaction is still in progress, the request for the new transaction occurs.

8.2 PCI Master Termination

If there is no target response to the initiated transaction within four clock cycles (five clocks in case of DAC transaction), the master issues a Master Abort event. The master deasserts FRAME* and on the next cycle deassert IRDY*. Also, the Interrupt Cause register's MMAbort bit is set and an interrupt is generated, if not masked.

The master supports several types of target termination:

- Retry
- Disconnect
- Target Abort

If a target terminated a transaction with Retry, the GT–64240A master re-issues the transaction. In default, the master retries a transaction until it is being served. When the master reaches this count value, it stops the retries and a bit is set in the Interrupt Cause register.

If a target terminates a transaction with Disconnect, the master re-issues the transaction from the point it was disconnected. For example, if the master attempts to burst eight 32-bit dwords starting at address 0x18, and the target Disconnects the transaction after the fifth data transfer, the master re-issues the transaction with address 0x2C to burst the left three dwords.

NOTE: To limit the number of retry attempts for transactions using Retry or Disconnect, set the RetryCtr in the PCI Timeout and Retry register to a desired count value, see Table 235 on page 210

If a target abnormally terminates a transaction with a Target Abort, the master does not attempt to re-issue the transaction. A bit in the Interrupt Cause register is set and an is interrupt generated, if not masked.

8.3 PCI Bus Arbitration

The GT–64240A supports both external arbiter or internal arbiter configuration through the PCI Arbiter Control register's EN bit [31], see Table 238 on page 212. If the bit is set to '1', the GT–64240A internal PCI bus arbiter is enabled.

NOTE: The internal PCI arbiter REQ*/GNT* signals are multiplexed on the MPP pins. For the internal arbiter to work, the MPP pins must first be configured to their appropriate functionality, see Section 19.1 "MPP Multiplexing" on page 480. Additionally, since the MPP default configuration is general purpose input, pull-ups must be set on all GNT* signals.

Since the internal PCI arbiter is disabled by default (the MPP pins function as general purpose inputs), changing the configuration can only be done by the CPU or through serial ROM initialization. The configuration cannot be done by an external PCI master (since an external master will not gain PCI bus arbitration).

8.3.1 PCI Master Bus Arbitration

Whenever there is a pending request for a PCI access, the PCI master requests bus ownership through the REQ* pin. As soon as the PCI master gains bus ownership (GNT* asserted), it issues the transaction. If no additional pending transactions exist, it deasserts REQ* the same cycle it asserts FRAME*. If parked on the bus, the master does not request the bus at all.

The GT–64240A implements the Latency Timer Configuration register as defined in PCI spec. The timer defines number of clock cycles starting from FRAME* assertion that the master is allowed to keep bus ownership, if not granted any more. If the Latency Timer is expired, and the master is not granted (GNT* not asserted), the master terminates the transaction properly on the next data transfer (TRDY* assertion). It re-issues the transaction from the point it was stopped, similar to the case of disconnect.

One exception is Memory Write and Invalidate command. In this case, the master quits the bus only after next cache line boundary, as defined in PCI spec.

8.3.2 Internal PCI Arbiter

The GT–64240A integrates two PCI arbiters, one per each PCI interface. Each arbiter can handle up to six external agents plus one internal agent (PCI_0/1 master).

The PCI arbiters implements a Round Robin (RR) arbitration mechanism.

The PCI arbiter performs a default parking on the last agent granted.

To overcome problems that happen with some PCI devices that do not handle parking properly, use the PCI Arbiter Control register's PD bits [21:14] as an option to disable parking on a per PCI master basis, see Table 238 on page 212.

NOTE: In addition to disabling parking to avoid issues with some problematic devices, it is required to disable parking on any unused request/grant pair. This is to avoid possible parking on non existent PCI masters. For example, if only three external agents are connected to PCI_0 arbiter, then PD[6:4] must be set to '1'.

8.4 PCI Master Configuration Cycles

The GT–64240A translates CPU read and write cycles into configuration cycles using the PCI configuration mechanism #1 (per the PCI spec). Mechanism #1 defines:

- A way to translate the CPU cycles into configuration cycles on the PCI bus
- A way to access the GT–64240A's internal configuration registers.

The GT–64240A contains two registers to support configuration accesses: PCI Configuration Address (Table 271 on page 224) and PCI Configuration Data (Table 272 on page 225). The mechanism for accessing configuration space is to write a value into the PCI Configuration Address register that specifies the:

- PCI bus number
- Device number on the bus
- Function number within the device
- Configuration register within the device/function being accessed

A subsequent read or write to the PCI Configuration Data register causes the GT–64240A to translate that Configuration Address value to the requested cycle on the PCI bus or internal configuration space.

The BusNum and DevNum fields of PCI P2P Configuration register affects the type of configuration access, see Table 245 on page 216.

If the BusNum field in the Configuration Address register equals the P2P Configuration register's BusNum field, but the DevNum fields do not match, a Type0 access is performed. This type of access addresses a device attached to the local PCI bus.

If the BusNum field in the Configuration Address register does not match the P2P Configuration register's BusNum field, a Type1 access is performed. This type of access addresses a device attached to a remote PCI bus.

The GT–64240A performs address stepping for the PCI configuration cycles. Address stepping allows for the use of the high-order PCI AD signals as IdSel signals through resistive coupling.[1]

Table 164 shows DevNum to IdSel mapping (type 0 configuration access).

Table 164: DevNum to IdSel Mapping

DevNum[15:11]	AD[31:11]
00001	0000.0000.0000.0000.0000.1
00010	0000.0000.0000.0000.0001.0
00011	0000.0000.0000.0000.0010.0
00100	0000.0000.0000.0000.0100.0
-- -- --	-- -- --
10101	1000.0000.0000.0000.0000.0
00000, 10110 - 11111	0000.0000.0000.0000.0000.0

A special cycle is generated if all of the following apply:

- The PCI Configuration Address register's BusNum field equals the P2P configuration register's BusNum field.
- The DevNum field is 0x1f.
- The function number is 0x7.
- The register offset is 0x0.

The CPU accesses the GT–64240A's internal configuration registers when the fields DevNum and BusNum fields in the Configuration Address register match the corresponding fields in the P2P Configuration register. The GT–64240A configuration registers are also accessed from the PCI bus when the GT–64240A is a target responding to PCI configuration read and write cycles.

1. "Resistive Coupling" also means "hook a resistor from ADx to IdSel" on a given device.

NOTES: The ConfigEn bit in the Configuration Address register must be set before the Configuration Data register is read or written. An attempt by the CPU to access a configuration register without this bit set results in PCI master behaving as if it performed a master abort - no PCI transaction is driven on the bus, nothing is returned for write transactions, and the internal register value is returned for write transactions.

A CPU access to the GT–64240A PCI_1 configuration register is done via the PCI_1 Configuration Address and Configuration Data registers. This is not compatible with the GT-64120 and GT-64130 devices.

The P2P Configuration register's BusNum and DevNum fields do not exist in the GT-64120 and GT-64130 devices. By default, the values of these fields are 0x0 which results in behavior identical to these devices.

8.5 PCI Target Address Decoding

The PCI target interface uses a one stage decoding process as described in Section 3.2 "PCI Address Decoding" on page 46. For an exact list of base address registers and size registers, see Section 8.19.1 "PCI Slave Address Decoding Registers" on page 192.

PCI interface supports 14 regular address windows plus 11 64-bit addressing windows. Each window is defined by the base and size registers. Each window can decode up to 4Gbyte space.

The PCI target interface also supports address remapping to any of the resources. This is especially useful when one needs to reallocate some PCI address range to a different location on memory. More over, it enables P2P access to a PCI agent located above the 4Gbyte space.

The PCI target interface contains a High P2P Remap registers that defines the upper 32-bit PCI address. If the register is set to 0, the PCI access from one PCI interface to another results in a SAC (Single Address Cycle) transaction. If the register is set to any other value, the PCI master issues a DAC transaction with the high 32 address bits set according to the value of the High P2P Remap register.

8.5.1 SDRAM and Device BARs

The GT–64240A contains four BARs for PCI access to SDRAM and five BARs for access to Devices. An address match in any of these BARs results in an access to the target chip select. There is no further sub decoding, as used to be in the GT-64120/GT-64130.

NOTE: Unlike the GT-64120/GT-64130, there are no Swap BARs in GT–64240A. Byte swapping is controlled via the Access Control registers. For more details, see Section 8.6 "PCI Access Protection" on page 162.

8.5.2 Internal Space Address Decoding

PCI_0/1 accesses the GT–64240A internal registers using memory or I/O transactions.

There is a dedicated BAR for PCI_0/1. No size registers exist for the internal space BARs. This means each BAR has a fixed internal space of 64Kbyte. This implies that on address decode of an internal BAR, all address bits[31:16] must match the BAR's bits.

NOTE: The PCI specification defines that an I/O mapped BAR may not consume more than 256bytes I/O space. This implies that GT–64240A I/O Mapped Internal BAR is not PCI compliant. By default, this BAR is disabled. Enable this BAR through the BAR Enable register, see Section 8.5.6.

8.5.3 Expansion ROM Address Decoding

Expansion ROM is enabled through reset configuration. For the PCI slave to respond to a PCI address hit in the expansion ROM space, the system software must first set the Configuration Command register's Target Memory Enable bit [1] to '1' and bit[0] of expansion ROM BAR to '1', as defined in PCI specification.

With the Expansion ROM enabled through the reset configuration of AD [17:16], the GT–64240A configuration space includes an expansion ROM BAR at offset 0x30 of function0 configuration space as specified in the PCI specification. Like the other BARs, there are expansion ROM size and remap registers. Address decoding is done the same way as for the other devices. A hit in the expansion ROM BAR results in an access to CS[3] or BootCS, depending on the setting of the PCI Address Decode Control register's ExpRomDev bit, seeTable 232 on page 205.

With the Expansion ROM disabled, the GT–64240A does not support expansion ROM BAR, offset 0x30 in the configuration space is reserved.

8.5.4 P2P BARs

The GT–64240A supports basic P2P functionality.

The GT–64240A contains two memory BARs plus an I/O BAR to support access between the two PCI interfaces. A PCI address hit in one of the P2P Memory BARs results in transferring the transaction to the other PCI interface Memory space. An address hit in the I/O P2P BAR results in transferring the transaction to the other PCI interface I/O space.

8.5.5 64-bit Addressing BARs

The GT–64240A supports 64-bit addressing through Dual Access Cycle (DAC) transactions. It contains 11 64-bit BARs. There are:

- Four SDRAM DAC BARs
- Five Device DAC BARs
- Two P2P DAC BARs

If the upper 32-bits of the BAR are not 0x0 (meaning the BAR maps an address space located above 4Gbyte), only addresses of PCI DAC transactions are compared against the 64-bit BAR. If the upper 32-bits of the BAR are 0, it acts as a regular 32-bit BAR, and only addresses of PCI SAC transactions are checked against it.

Each 64-bit BARs have their own size registers. However, their size registers can map up to 4Gbyte per each BAR.

NOTE: The GT–64240A does not support larger address windows than 4Gbyte per each BAR. It does support the location of the address window in offsets that are higher than the 4Gbyte space.

8.5.6 Base Address Registers Enable

Only if bit[0] of the Configuration Command register (Target I/O Enable) is set to ‘1’ does the PCI slave responds to an address hit in the I/O BARs. It responds to an address hit in any of the other BARs only if bit[1] of Configuration Command register (Target Memory Enable) is set to ‘1’.

To disable a specific BAR space, the GT–64240A includes a 27-bit BAR Enable register - bit per BAR. Setting a bit to ‘1’ disables the corresponding BAR. A disabled BAR is treated as a reserved register (read only 0). PCI access match to a disabled BAR is ignored and no DEVSEL* asserted.

8.5.7 Loop Back Access

By default, the PCI slave does not respond to any PCI transaction initiated by the PCI master. However, if the PCI Command register’s LPEn bit is set to ‘1’, the slave responds to the PCI master transactions, if targeted to the slave address space.

NOTE: This loop back feature is only used for system debug. Do not use in normal operation.

8.6 PCI Access Protection

The PCI slave interface supports configurable access control. It is possible to define up to eight address ranges to different configurations. Each region can be configured to:

- Write protection
- Access protection
- Byte swapping
- Read prefetch

Three registers define each address window - Base (low and high) and Top. The minimum address range of each window is 1Mbyte. An address received from the PCI, in addition to the address decoding and remapping process, is compared against the eight Access Control base/top registers. Bits[63:32] of DAC cycle address are checked to be equal to the Base high register. Bits[31:20] of the address are checked to be between the lower and upper addresses defined by bits[11:0] of Base and Top registers. If an address matches one of the windows, GT–64240A handles the transaction according to transaction type and the attributes programed in the Access Control register.

Each region contains two protection bits:

- Access protection
  Any PCI access to this region is forbidden.
- Write protection
  Any PCI write access to this region is forbidden.

If an access violation occurs:

- The PCI slave interface terminates the transaction with Target Abort.
- The transaction address is latched in PCI Slave Error Address register.
- The PCI AddrErr bit in the interrupt cause register is set.

NOTE: The GT–64240A internal registers space is not protected, even if the access protection windows contain this space.

The other attributes of the Access Control registers are discussed in Section 8.8 and Section 8.13.

8.7 P2P Configuration Transactions

The GT–64240A supports not only memory and I/O P2P transactions between the two PCI interfaces, but also propagation of configuration cycles.

Each PCI interface may respond to a type 1 configuration transaction, according to the settings of the PCI P2P Configuration register's 2ndBusL and 2ndBusH fields, see Table 245 on page 216. These fields specify the buses resides on the other PCI interface. Upon detecting of PCI configuration type 1 transaction, the PCI target interface decodes the bus number driven on the AD bus (bits[23:16]). If the bus number is within the range of the other PCI interface (including 2ndBusL and 2ndBusH boundaries), the transaction is propagated to the other PCI interface.

NOTE: By default, the 2ndBusL field is greater than 2ndBusH. This means that propagating a type 1 configuration transaction is disabled.

In case the type 1 configuration is claimed (DEVSEL* asserted), the transaction type driven by the PCI master is determined according to the device number, function number, and register offset driven on the AD bus (bits 15:11, 10:8 and 7:2 respectively), as follows:

1. If the received bus number is identical to the other PCI interface bus number, it converts the transaction

to type 0.

2. If the received bus number differs from the other PCI interface bus number, it keeps the transaction as type 1.
3. If the received bus number is identical to the other PCI interface bus number, the device number is ‘11111, the function number is ‘111, and the register offset is 0x0. It drives a Special Cycle.

NOTE: Although the GT–64240A supports all types of P2P cycles, it is not P2P Bridge Specification compliant. It does not implement all required bridge configuration registers, nor keeps all P2P transactions ordering rules.

Unlike a P2P bridge that has a primary and secondary interfaces, in the GT–64240A the P2P functionality is identical in both directions.

8.8 PCI Target Operation

The GT–64240A responds to the following PCI cycles as a target device:

- Memory Read
- Memory Write
- Memory Read Line
- Memory Read Multiple
- Memory Write and Invalidate
- I/O Read
- I/O Write
- Configuration Read
- Configuration Write
- DAC Cycles

The GT–64240A does not act as a target for Interrupt Acknowledge and Special cycles (these cycles are ignored). The GT–64240A does not support Exclusive Accesses. It treats Locked transactions as regular transactions (it does not support LOCK* pin).

The slave consists of 512 bytes of posted write data buffer that can absorb up to 4 write transactions, and 8 read prefetch buffers, 128 bytes each, to support up to 8 delayed reads.

8.8.1 PCI Write Operation

All PCI writes are posted. Data is first written into the posted write buffer and later written to the target device.

The slave supports unlimited burst writes. The write logic separates the long PCI bursts to fixed length bursts towards the target device. Program the internal burst length to four, eight, or 16 64-bit words through the PCI Access Control registers’s MBurst bits [21:20], see Table 247 on page 217. Whenever this burst limit is reached, the slave generates a write transaction toward the target device, while continuing to absorb incoming data from the PCI. The PCI burst writes have no wait states (TRDY* is never deasserted). In case the slave transaction queue is full, a new write transaction is retried. This depends on target device capability to absorb the write data (target device bandwidth and arbitration scheme).

The slave posting writes logic also aligns bursts that do not start on a 32/64/128-byte boundary, depending on the WrBurst setting, for more efficient processing by the target units. For example, if MBurst is set to maximum bursts of eight 64-bit words, and a PCI long burst write transaction starts at address 0x18, the slave issues a write transaction of five 64-bit words to the target unit and continues with a new transaction to address 0x40.

NOTE: If the PCI address does not match any of the PCI Access Control registers address windows, the default burst write size is four 64-bit words.

The PCI slave treats Memory Write and Memory Write and Invalidate commands the same way.

If the region is marked as cache coherent, MBurst must be set to four 64-bit words.

8.8.2 PCI Read Operation

All PCI reads can be configured to be handled as non prefetched, prefetched or aggressive prefetched, and also to be handled as delayed transactions or not. Also, it is possible to program the amount of prefetched data. These read attributes are programable per transaction type (read/read-line/read-multiple) and per address range, as defined by the PCI Access Control registers (see Table 247 on page 217).

If an address range is marked as non-prefetchable (PrefetchDis bit in the PCI Access Control register), a PCI read to this region results in a single word read from the target device. An attempt to burst from a non-prefetchable region results in a disconnect after the first data. It is recommended to mark a region as non-prefetchable, only if prefetch reads from this area are destructive (e.g. target device is a FIFO).

In case of a prefetchable region, the size of the burst read requested from the target device can be programed to four, eight, or 16 64-bit words, through PCI Access Control register's MBurst bits. If the typical PCI read transaction is long, it is recommended to set this bit to long bursts. However, setting this bit to long bursts implies that the target unit (SDRAM interface unit for example) is busy for many cycles and not able to serve requests from other interfaces (CPU for example).

NOTE: If the region is marked as cache coherent, MBurst must be set to four 64-bit words.

The PCI slave interface supports two prefetch modes, selected via the RdPrefetch, RdLinePrefetch, and RdMulPrefetch bits in the PCI Access Control register - regular prefetch and aggressive prefetch.

In regular prefetch mode, the target device is requested for a single burst transaction (burst size depends on the setting in the RdBurst field). If by the time all of the burst data was driven on the PCI bus and the PCI read transaction is still alive (implying a longer burst is required), the slave terminates the transaction with disconnect and the initiating master must re-issue the remaining transaction. If the typical PCI reads behave this way (requiring more than a single target device burst), it is recommended to use the aggressive prefetch mode.

In the aggressive prefetch mode, the target is requested for two bursts in advance (similar to aggressive prefetch in GT-64120 and GT-64130 devices). If the read transaction on the PCI is still active by the time the first burst is driven on the PCI bus, the slave prefetches an additional burst (a third one) while driving the second burst on the PCI bus.

NOTE: The PCI slave treats Memory Read, Memory Read Line, and Memory Read Multiple commands the same, unless using different RdPrefetch, RdLinePrefetch, RdMulPrefetch settings. These settings enable "smart" PCI masters that generate different PCI read commands to have regular prefetch for one type of command, and aggressive prefetch for another type.

If not using delayed reads, the slave drives read data on the PCI bus (TRDY* asserted) as soon as data arrives from the target unit. The slave does not wait for the read buffer to be full. In case of a burst read from a slow tar-

get device, the slave might need to insert wait states (TRDY* deasserted) between the data phases, according to the data rate from the target.

As mentioned above, the PCI slave prefetch read data, based on the setting of the MBurst parameter. However, there is one exception. If the initiating PCI master, asserts FRAME* for a single clock cycle (which implies it request a single data), the PCI slave will read a single data from the target interface, regardless of MBurst setting

NOTE: If the PCI address does not match any of PCI Access Control registers address windows, the default burst read size is four 64-bit words, and is treated as a non-delayed read. Also, read prefetching is determined according to the value of the corresponding Base Address Register prefetch bit.

With a PCI burst access that uses a start address outside the range of all the Access Control address windows, the PCI slave cannot recognize when the burst is crossing one of the Access Control windows. So, if using the Access Control registers, it is recommended that they cover the whole PCI slave address space. Conversely, if a PCI burst start address is within an access region and then crosses the region boundary, the PCI slave disconnects.

8.8.3 PCI Delayed Reads

Delayed reads are configurable through the PCI Access Control register's DReadEn bit [13]. Delayed reads are typically useful in multi-PCI masters environments. In these environments, PCI bus efficiency is critical and there is a need to minimize wait states on the bus. When using delayed reads, there are no wait states (not to first data nor to consecutive data). The bus is released quickly, allowing other PCI masters gain bus arbitration.

The slave supports up to eight pending delayed reads. When a read transaction is marked as a delayed read, the slave issues a STOP* immediately (retry termination), but internally continues the transaction towards the target device. When the data is received from the target, it is written to one of the eight read buffers. Any attempt to retry the original transaction before the read buffer is full (the whole burst is written into the buffer) results in STOP*. When the read buffer is full, a retry of the original transaction results in data driven immediately on the bus.

If by the time all burst data is driven on the PCI bus and the PCI read transaction is still alive (implying that a longer burst is required), the slave terminates the transaction with disconnect and the initiating master must re-issue the remaining transaction. If the typical PCI reads behave this way (requiring more than a single target device burst), it is recommended to use the aggressive prefetch mode.

NOTE: If a region is marked as non-prefetchable, a read access to this region is treated as a single data delayed read, even if the region is not marked to be used with delayed reads.

If a region is marked to be used with aggressive prefetch, a read access to this region is treated as a delayed read (prefetch of two read buffers), even if region is not marked to be used with delayed reads.

8.8.4 PCI Slave Read Buffers

The slave handles a queue of available read buffers.

For every incoming read transaction, the slave allocates a new read buffer. The read buffer is where the returned data from the target is stored. When the buffer data is flushed to the PCI bus (completion of the read transaction), the buffer is invalidated and is free to be re-used.

If all eight read buffers are full and a new read buffer is required (a new read transaction), the incoming read transaction is retried.

To prevent dead locks due to “stuck” buffers (valid buffers that are never being accessed), the GT–64240A supports a Discard Timer register, see Table 236 on page 211. Each read buffer has its timer initiated to the Discard Timer value. When the address is isuued on the PCI, the buffer timer starts counting down. If the buffer timer reaches ‘0’ before being accessed, the buffer is invalidated. Setting the Discard Timer register to ‘0’ prevents the slave from invalidating the read buffers.

8.8.5 PCI Access to Internal Registers

PCI writes to internal registers are posted as any other PCI write to memory, with the exception of writes to the PCI interface unit’s internal registers. These writes are non-posted – the slave asserts TRDY* only when data is actually written to the internal register. This implementation guarantees that there is never a race condition between the PCI transaction changing address mapping (Base Address registers) and the following transactions.

Burst writes to internal registers are not supported. An attempted burst to internal registers results in a disconnect after 1st TRDY*.

PCI reads from internal registers are treated as reads from a non-prefetchable region (single 32-bit word read), regardless of PCI Access Control registers settings. An attempt of burst read from internal registers results in a disconnect after the first data.

8.8.6 PCI I/O Access

The GT–64240A PCI slave only supports I/O read and write accesses to its internal registers, and to the other PCI interface (P2P bridging) if there are multiple PCI interfaces. Both cases are treated the same. An I/O write is treated as a single 32-bit word write. An I/O read is treated as a single 32-bit non-prefetchable read.

8.8.7 PCI Configuration Access

The GT–64240A PCI slave supports configuration read and write access to the GT–64240A configuration space, and to the other PCI interface (P2P bridging) if there are multiple PCI interfaces. Both cases are treated the same. A configuration write is treated as a 32-bit word non posted write. A configuration read is treated as 32-bit word non prefetchable read.

8.9 PCI Target Termination

The GT–64240A PCI slave supports the three types of target termination events specified in PCI specification – Target Abort, Retry and Disconnect.

Target Abort is activated in the following cases:

- I/O transaction with address bits [1:0] not consistent with byte enables.
- Address parity error.
- Violation of PCI access protection setting.

In any of these cases, the PCI slave latches the address in the PCI Slave Error Address register and sets an interrupt bit in the Interrupt Cause register.

If the PCI slave cannot complete a transaction in a “reasonable time”, it might terminate a transaction with Retry or Disconnect. All conditions of Retry and Disconnect are described bellow.

8.9.1 Timeout Termination

The GT–64240A includes two 8-bit timeout registers (see Table 235 on page 210) – timeout0 for Retry termination and timeout1 for Disconnect termination (same as in GT-64120 and GT-64130 devices). Timeout0 defines the maximum allowed wait-states between FRAME* assertion and first TRDY* assertion. Timeout1 defines the maximum wait-states between consecutive TRDYs (in case of a burst). By default, these registers are initialized to 0xf and 0x7, as required by PCI spec. However, it is possible to program these registers to longer numbers to support access to slow devices.

Retry or Disconnect termination due to timeout expired might happen if:

- Timeout0 expired before first read data received from the target device. Relevant only for non-delayed reads.
- Timeout1 expired before next read data of a burst read received from the target device. Relevant only for non-delayed reads.
- Timeout0 expired before non-posted write completes.

NOTE: Timeout0 must be greater than ‘5’.

8.9.2 Non-Timeout Termination Conditions

There are more conditions of immediate Retry termination (without waiting for timeout):

- Delayed reads.
- Slave transaction queue is full.
- A new read transaction, and there is no available read buffer.
- A new synch barrier transaction while there is a pending unresolved previous sync barrier.

Also, there are some additional disconnect cases:

- A burst access with start address bits[1:0] different than ‘00.
- A burst access that reaches BAR boundary or Access Control window boundary.
- A delayed read completion that requires more than one buffer.

8.10 Initialization Retry

Some applications require programing of the PCI configuration registers in advance of other bus masters accessing them. In a PC add-in card application, for example, the Device ID, BAR size requirements, etc., must be set before the BIOS attempts to configure the card. The GT–64240A provides a mechanism that directs the PCI target interface to Retry all of the transactions until this configuration is complete. This prevents race conditions between the local processor and the BIOS.

If Initialization Retry is enabled at reset, the PCI slave Retries any transaction targeted to the GT–64240A’s space. The GT–64240A remains in this retry mode until the CPU configuration register’s StopRetry bit is set. This mode is useful in all of the applications in which the local CPU programs the PCI configuration registers.

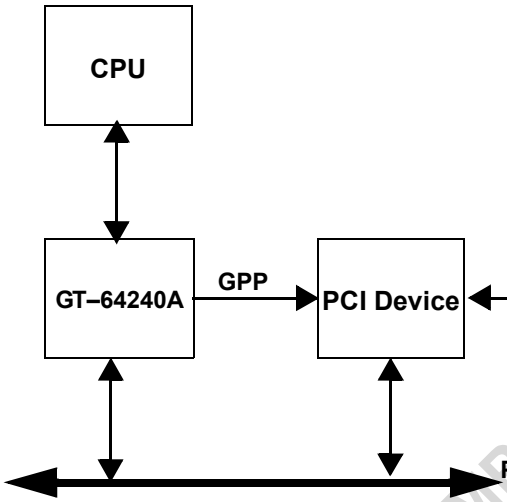
If serial ROM initialization is enabled, any PCI access to the GT–64240A is terminated with Retry. This lasts until the end of the initialization. See Section 24. “Reset Configuration” on page 522 for more details.

8.11 Synchronization Barrier

The GT–64240A supports a sync barrier mechanism. This mechanism is a hardware hook to help software synchronize between the CPU and PCI activities. The GT–64240A supports sync barrier in both directions - CPU-to-PCI and PCI-to-CPU.

Figure 31 shows an example of the PCI sync barrier application.

Figure 31: CPU Sync Barrier Example



Assume the CPU sends a packet to some PCI device and then notifies this device (via one of the GPP pins) that it has a packet waiting to handle. Since the packet may still reside in the GT–64240A CPU interface write buffer or in the PCI master write buffer, the PCI device must first perform a sync barrier action, to make sure the packet is no longer in the GT–64240A buffers.

The PCI slave “synchronization barrier” cycles are Configuration Reads. If there is no posted data within the CPU interface write buffer and PCI master write buffer, the cycle ends normally. If after a timeout0 period there is still posted data in the buffers, the cycle is terminated with Retry. Until the original cycle ends, any new “synchronization barrier” cycles are terminated with Retry. The PCI slave only handles a single pending sync barrier transaction at a time.

NOTE: The PCI device that initiated the sync barrier transaction, must keep retrying the transaction until it completes. If the transaction is terminated, and never retried, any new sync barrier attempt results in a retry termination (since the PCI slave can support only a single outstanding sync barrier transaction at a time). In order to prevent dead locks due to missing sync barrier completion, the sync barrier mechanism is protected by the discard timer, similar to the delayed read bluffers, see Section 8.8.4.

An alternative method for generating the PCI slave sync barrier is reading from the PCI Sync Barrier Virtual register, see Table 244 on page 216. When reading this register from PCI, the PCI slave checks if the write buffers to be empty, and only when there is no posted write data in the buffers, completes the transaction on the PCI bus. The returned data is indeterministic.

Setting the PCI Control register’s SBD bit to ‘1’ disables sync barrier action on configuration reads. This allows the user to perform configuration reads to the GT–64240A without suffering from sync barrier latency.

8.12 Clocks Synchronization

The PCI interface clock (Clk) is designed to run asynchronously with respect to the memory clock (TClk) and CPU clock (SysClk). Also, the two PCI interfaces can run asynchronously to each other.

The PCI interface includes synchronization logic that synchronizes between the Clk and TClk clock domains, enabling these two clocks to run asynchronously.

NOTE: Unlike the GT-64120 and GT-64130, the GT–64240A has no special synch modes, for different frequency ranges. The only restriction is that the TClk frequency must be greater than the Clk frequency.

8.13 Data Endianess

The GT–64240A supports interfacing with both Little and Big Endian orientation CPUs. Although the PCI specification defines the PCI bus only as a Little Endian bus, the GT–64240A supports also interfacing Big Endian PCI devices.

Endianess conversion is supported in both directions - access to PCI via the PCI master interface and access from PCI via the PCI slave interface.

Both PCI master and slave supports byte and word swapping. The swapping is refered to a 64-bit words (as this is the GT–64240A internal data path width). Table 165 shows an example of the data 0x0011223344556677.

Table 165: Data Swap Control

Swap Control	Swapping Granularity	Swapped Data							
00	Byte	77	66	55	44	33	22	11	00
01	Non	00	11	22	33	44	55	66	77
10	Byte and Word	33	22	11	00	77	66	55	44
11	Word	44	55	66	77	00	11	22	33

The right swapping setting depends on the PCI bus width (32/64) and endian orientation (Big/Little), as well as the CPU bus endianess orientation.

Table 166: 32-bit PCI Byte and Word Swap Settings

	Little Endian PCI agent	Big Endian PCI agent
Big endian CPU bus	Byte swapping	Word swapping

Table 167: 64-bit PCI Byte and Word Swap Settings

	Little Endian PCI agent	Big Endian PCI agent
Big endian CPU bus	Byte swapping	No swapping

8.13.1 PCI Slave Data Swapping

For maximum endianess flexibility, it is possible to configure each of the eight address ranges defined by the PCI Access Control registers to different data swapping. This feature enables different PCI masters with different endianess conventions to interface with the GT–64240A.

The GT–64240A still preserves the GT-64120/130 devices data swapping mechanism for software compatibility. If the PCI Command register's SwapEn bit is cleared (default), the PCI slave handles data according to the setting of the PCI Command register's SByteSwap [16] and SWordSwap bit [11] (see Table 233 on page 206), as in the GT-64120/130 devices.

The GT–64240A internal registers always maintain Little Endian data. By default, it is assumed that data driven on the PCI bus is in Little Endian convention, and there is no data swapping on PCI access to the internal registers. However, the GT–64240A supports data swapping also on the PCI access to internal registers via the PCI Command register's SIntSwap bits [26:24].

8.13.2 PCI Master Data Swapping

Very similar to the slave data swapping mechanism, the PCI master also supports data swapping on any access to the PCI bus.

It also supports flexible swapping control, determined by the initiator, on an address window basis. This feature enables the CPU, IDMAs, and Communication units to interface different PCI targets with different endianess conventions.

The GT–64240A still preserves the GT-64120/130 devices fixed data swapping for software compatibility. If the PCI Command register's SwapEn bit is cleared (default), the PCI master handles data according to the setting of the PCI Command register's MByteSwap and MWordSwap bits, see Table 233 on page 206.

See the following sections for further details about transaction initiator endianess configuration:

- For CPU details: Section 4.11 "CPU Endian Support" on page 71.
- For IDMA details: Section 10.7 "Big and Little Endian Support" on page 285.
- For Communication unit details: Section 12.3 "Big and Little Endian Support" on page 322.

8.14 64-bit PCI Interface

The GT–64240A supports a 64-bit PCI interface. To operate as a 64-bit device, the REQ640* pin must be sampled LOW on RST* rise as required by PCI spec (Hold time of REQ640* in respect to RST* rise is '0').

When the GT–64240A is configured to 64-bit PCI, both master and target interfaces are configured to execute 64-bit transactions, whenever it is possible.

NOTE: Since PCI0 interface supports CompactPCI Hot Swap Ready, P64EN0* pin is used to detect a 64-bit PCI bus rather than REQ640*. If not using CompactPCI, connectPCI0 REQ640* to P64EN0* pin.

8.14.1 PCI Master 64-bit Interface

The PCI master interface always attempts to generate 64-bit transactions (asserts REQ640*), except for I/O or configuration transaction or when the required data is no greater than 64-bits. If the transaction target does not respond with ACK640*, the master completes the transaction as a 32-bit transaction.

The PCI master also avoids from generating a 64-bit transaction, if the requested address is not 64-bit aligned, and the PCI Command register's M64Allign bit [18] is set cleared, see Table 233 on page 206. For example the requested address is 0x4, the master issues a 64-bit transaction (assert REQ640*) with byte enables 0x0f. If the target does not respond with ACK640*, the transaction becomes a 32-bit transaction, with the first data phase driven with byte enable 0xf. Although it is fully compliant with the PCI specification, some target devices do not tolerate this behavior. Use the M64Allign bit to prevent this problem.

When a PCI burst running in 64-bit mode is disconnected, and the amount of data the master needs to drive is not greater than 64-bit, it completes the disconnected transaction as a 32-bit master (does not assert REQ640*). This behavior has a small penalty in case the target device is capable of accepting the transaction as a 64-bit transaction. More over, it might be problematic when the target is a 64-bit Big Endian target. As mentioned before, the byte swapping setting depends not only on the endianess nature of both initiator and target but also on bus width. Changing bus width in the middle of a transaction targeted to a Big Endian device results in an incorrect data transfer.

NOTE: See the Galileo Technology Technical Bulletin TB-51 for more information on 64-bit Big Endian PCI bus.

If the targeted device on the PCI bus is a 64-bit device that ALWAYS responds with ACK640* to 64-bit transaction, the PCI master can be configured to always assert REQ640*, even if the amount of data needs to be transferred is less than or equal to 64-bit. Each initiating interface (CPU, IDMA, Communication units) has programable bits, that forces the PCI master to issue 64-bit transactions. When running in this mode, correct endianess is guaranteed, even when interfacing a 64-bit Big Endian device on the PCI bus.

NOTES: Forcing REQ640* is allowed only when the target PCI device responds with ACK640* to a 64-bit transactions. If the target device is not of that type and REQ64* is forced, a PCI violation occurs and the system might hang.

The PCI bus is defined as a Little Endian bus. Placing Big Endian devices on the bus is not compliant with the PCI specification. This feature of forcing REQ640* is implemented to support 64-bit Big Endian devices on the PCI bus. The hook of forcing REQ640* is not fully compliant with the PCI specification, and must be used carefully.

8.14.2 PCI Slave 64-bit interface

The PCI target interface always responds with ACK640* to a 64-bit transaction, except for accesses to configuration space, internal registers, I_2O space, or I/O transaction.

8.15 64-bit Addressing

Both PCI master and slave support 64-bit addressing cycles.

Both CPU and DMAs support 64-bit remapping registers towards the PCI interface. If the CPU or one of the DMAs initiate a PCI transaction with an address higher than 4Gbyte (which means that the upper 32-bit address is not 0), the master initiates a DAC transaction. This means the transaction address phase takes two clock cycles.

On the first cycle, the master drives a '1101' value on C/BE[3:0]* and the lower 32-bit address on AD[31:0]. On the next cycle it drives the required command on C/BE[3:0]* and the upper 32-bit address on AD[31:0].

If the PCI interface is configured to 64-bit bus, the master drives on the first cycle the required command on C/BE[7:4]* and the upper 32-bit address on AD[63:32]. This is useful when the target is also a 64-bit addressing capable device. In this case, the target starts address decoding on the first cycle, without waiting for the second address cycle.

On a DAC transaction, target address decode time is one cycle longer than in SAC transaction. Thus, the master issues a master abort on a DAC transaction only after five clock cycles, rather than four clocks in the case of SAC.

As a target, GT–64240A responds to DAC transactions if the address matches one of it's 64-bit BARs. In this case, the slave starts address decoding only after 2nd cycle (when the whole 64-bit address is available). This implies that DEVSEL* asserts three clock cycles after FRAME* rather than two clocks in the case of SAC transaction.

The PCI slave 64-bit BARs are not necessarily used only for DAC transactions. If the upper 32-bit of the BAR are set to '0', it acts as a 32-bit BAR responds to SAC transactions.

NOTE: The PCI specification restricts the PCI masters from issuing DAC transactions if the address is bellow 4Gbyte space.

8.16 PCI Parity and Error Support

The GT–64240A implements all parity features required by the PCI specification. This includes PAR, PERR*, and SERR* generation and checking, also PAR64 in case of 64-bit PCI configuration.

It also supports propagation of errors between the different interfaces. For example, a PCI read from SDRAM with ECC error detection may be configured to be driven on the PCI bus with bad PAR indication.

The PCI interface also supports other error conditions indications, such as access violation and illegal PCI bus behavior, see Section 8.6 "PCI Access Protection" on page 162 and Section 8.9 "PCI Target Termination" on page 167 for more details.

The PCI parity support is detailed in Section 6. "Address and Data Integrity" on page 133.

8.17 Configuration Space

The PCI slave supports Type 00 configuration space header as defined in PCI specification. The GT–64240A is a multi-function device and the header is implemented in all eight functions as shown in Figure 32 and Figure 33. The configuration space is accessible from the CPU or PCI buses.

If IDSEL* is active and it is a type 0 configuration transaction, the slave responds to configuration read/write. Many of functions 1-7 registers are aliased to function 0 registers. For example, access to Vendor ID register in function 1 actually accesses Vendor ID register of function 0.

The GT–64240A acts as multi function device regardless of multi-function bit setting (bit[7] in Header Type) - it responds to configuration access to any of the eight functions.

Each of the two PCI interfaces implements the configuration header. Each PCI can also access the other PCI's configuration space, but with offset increment of 0x80. For example, the PCI_0 Vendor ID is accessed at offset 0x0 from PCI_0, but at offset 0x80 from PCI_1 bus. Or, the PCI_1 Vendor ID is accessed at offset 0x0 from PCI_1, but at offset 0x80 from PCI_0 bus. This is especially required for PC environment where BIOS expects to see the Base Address Registers at specific offsets.

NOTE: Although the GT–64240A supports P2P transactions, it does not contain the required P2P device configuration header and is not P2P spec compliant.

8.17.1 Plug and Play Base Address registers Sizing

Systems adhering to the plug and play configuration standard determine the size of a base address register's decode range by first writing 0xFFFF.FFFF to the BAR, then reading back the value contained in the BAR. Any bits that were unchanged (i.e. read back a zero) indicate that they cannot be set and are not part of the address comparison. With this information the size of the decode region can be determined.

The GT–64240A responds to BAR sizing requests based on the values programmed into the Bank Size Registers (see Table 294 on page 237). Whenever a BAR is being read, the returned data is the BAR's value masked by it's corresponding size register. For example, if SCS[0] BAR is programed to 0x3FF0.0000 and SCS[0] Size register is programed to 0x03FF.FFFF, the PCI read of SCS[0] BAR will result in data of 0x3C00.0000.

The Size registers can be loaded automatically after RESET as part of the GT–64240A serial ROM initialization, see Section 24. "Reset Configuration" on page 522 for more details.

Figure 32: PCI Configuration Space Header

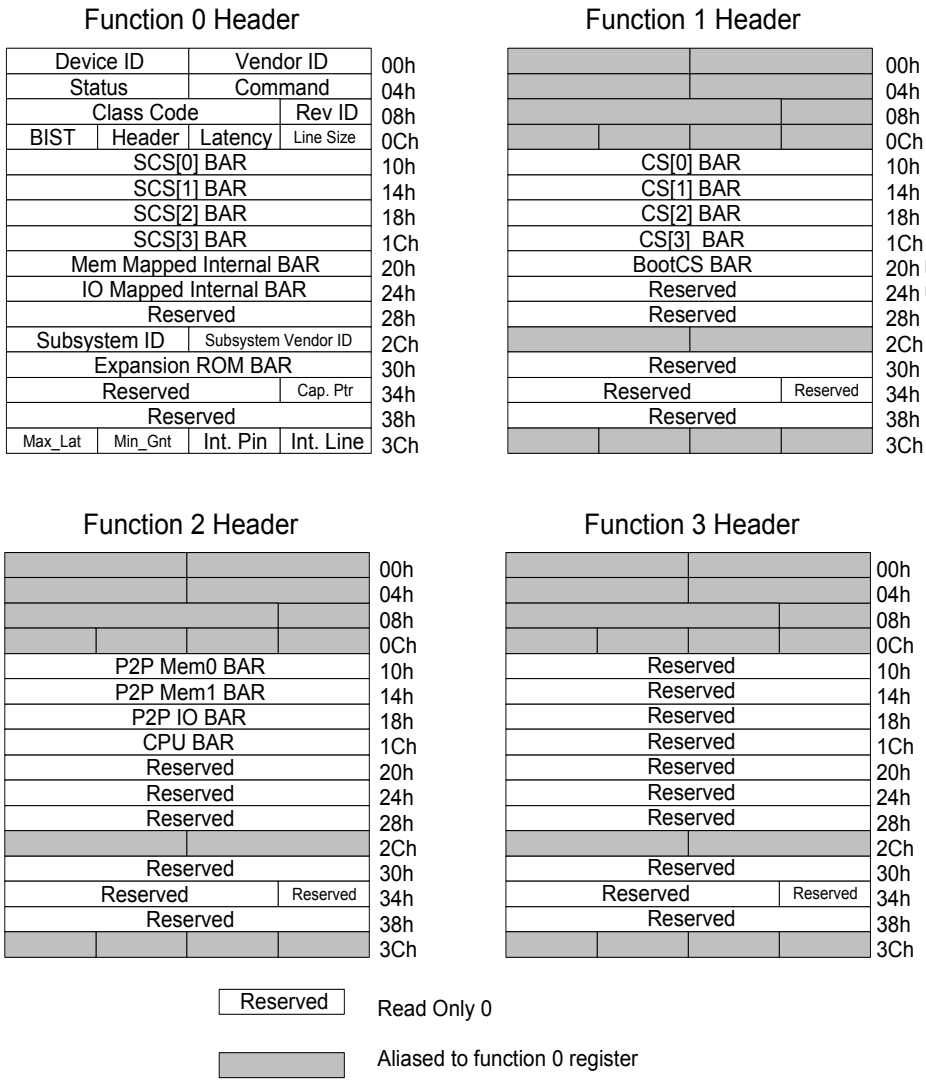
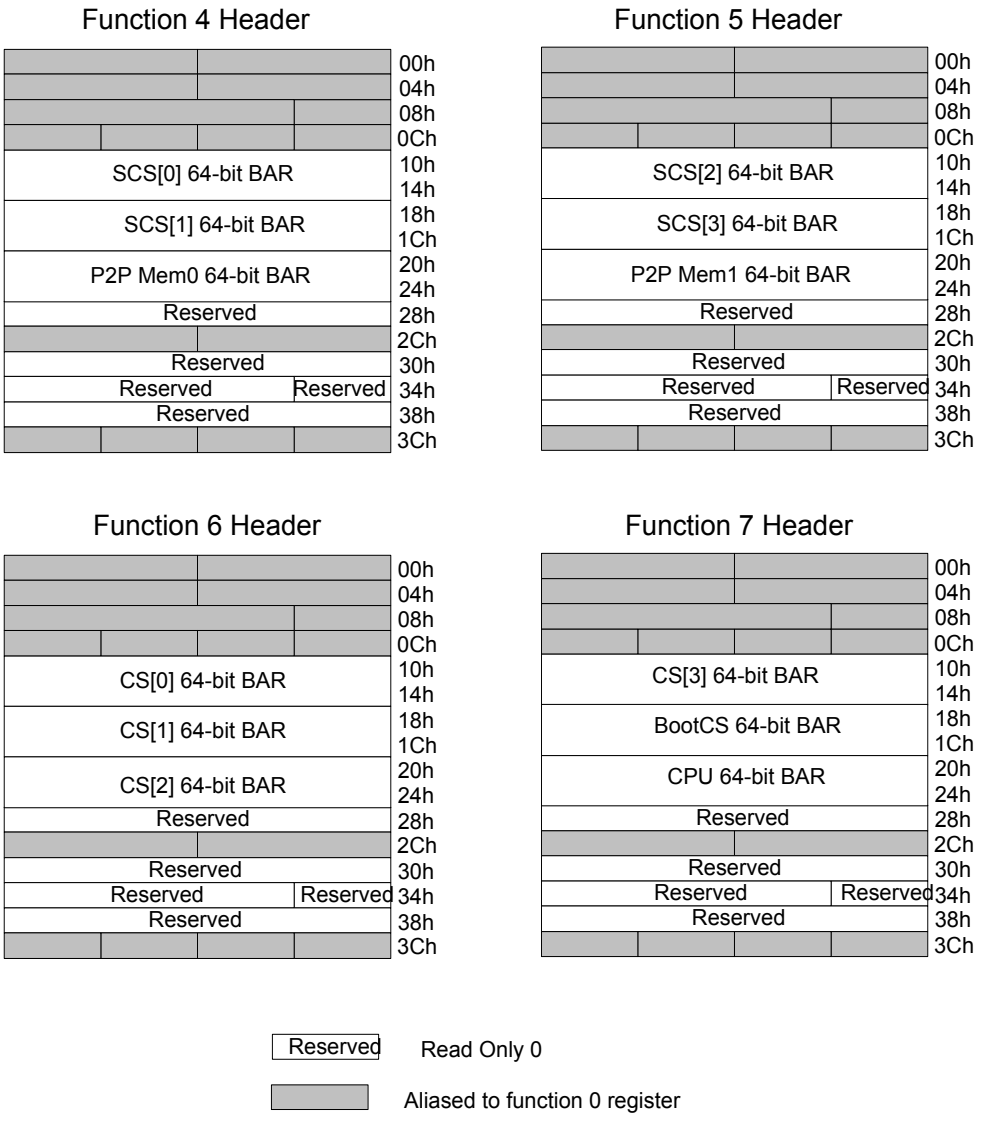


Figure 33: PCI Configuration Space Header[1]



1. Function 7 CPU 64-bit BAR is a Reserved register

8.18 [1]PCI Special Features

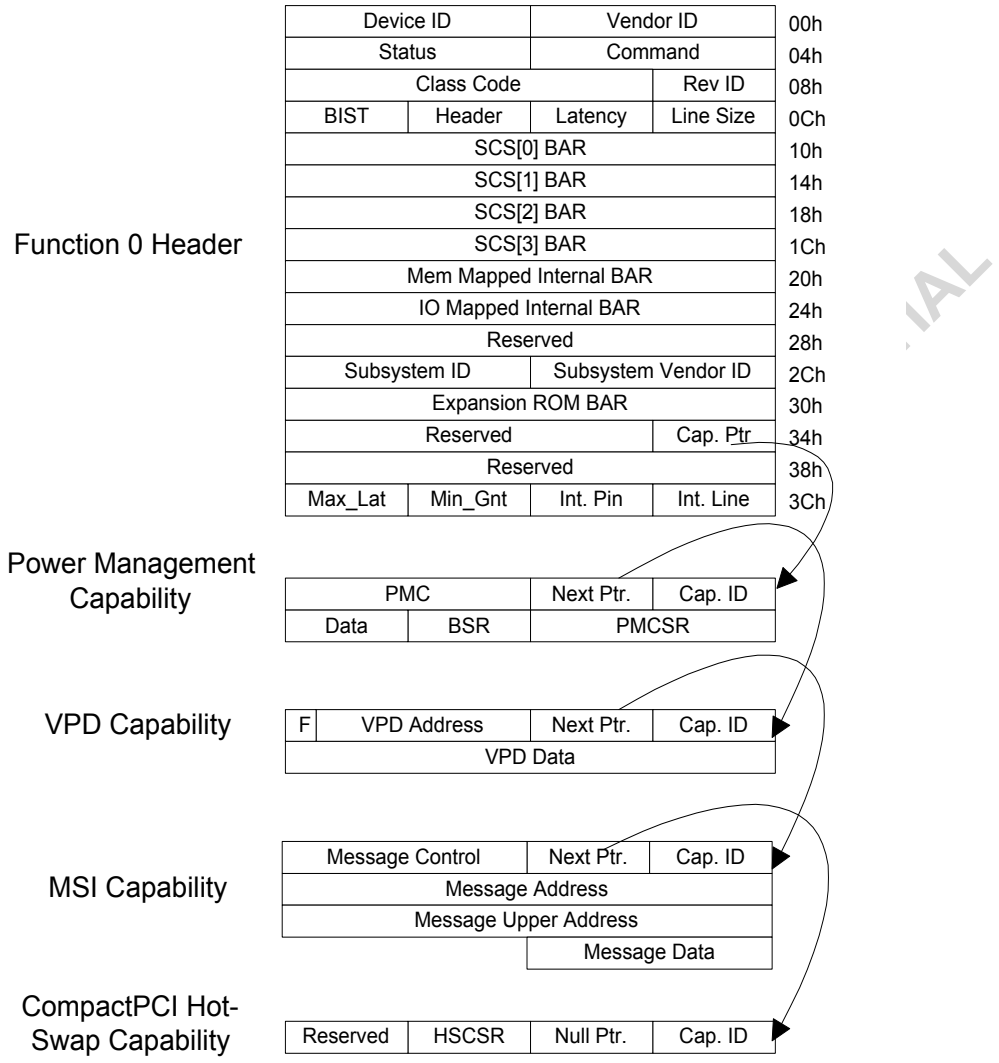
The GT–64240A supports the following special PCI features:

- Built In Self Test (BIST)
- Vital Product Data (VPD)
- Message Signaled Interrupt (MSI)
- Power Management
- Compact PCI Hot Swap

The VPD, MSI, PMG, and HotSwap features are configured through Capability List, as shown in Figure 34.

1. Function 7 CPU 64-bit BAR is a Reserved register

Figure 34: GT–64240A Capability List



8.18.1 Power Management

The GT–64240A implements the required configuration registers defined by the PCI specification for supporting system Power Management as well as PME* pin. The registers are implemented on both PCI_0 and PCI_1 interfaces. This implementation is fully compliant with the specification.

NOTE: For full details on system Power Management implementation, see the PCI specification.

The Power Management capability structure consists of the following fields:

- Capability structure ID. The ID of PMG capability is 0x1.
- Pointer to next capability structure.

- Power Management Capability.
- Power Management Status and Control.

The Power Management registers are accessible from the CPU or PCI. Whenever PCI_0 or PCI_1 updates the PCI Power Management Control and Status register's Power State bits [1:0] (see Table 305 on page 242), the PCI Interrupt Cause register's PM interrupt bit is set and an interrupt to the CPU or PCI is generated, if not masked by interrupt mask registers.

PME* is an open drain output. When the CPU sets PME_Status bit to '1' in the PMCSR register, the GT–64240A asserts PME*. It keeps asserting PME* as long as the bit is set, and the PME_En bit is set to '1' in the PMCSR register. The PCI clears the PME_Status by writing '1', causing the deassertion of PME*.

The PME0* and PME1* pins are multiplexed on the GT–64240A MPP pins. If PME* support is required, first program the MPP pins to the appropriate configuration. See Section 19. "Pins Multiplexing" on page 480 for details.

NOTE: The GT–64240A does not support it's own power down. It only supports a software capability to power down the CPU or other on board devices.

8.18.2 Vital Product Data (VPD)

VPD is information that uniquely identifies hardware elements of a system. VPD provides the system with information such as part number, serial number or any other information.

The PCI specification defines a method of accessing VPD. The GT–64240A VPD implementation is fully compliant with the spec. For full details on the VPD's structure, see the PCI specification.

The VPD's capability structure consists of the following fields:

- Capability structure ID. The ID of VPD capability is 0x3.
- Pointer to next capability structure.
- VPD Address. The 15-bit address of the accessed VPD structure.
- Flag. Used to indicate data transfer between VPD Data register and memory.
- VPD Data. The 32-bit VPD data written to memory or read from memory.

The GT–64240A supports a VPD located in CS[3]* or BootCS* Device. PCI access to this VPD results in access to CS[3] or BootCS*, depending on the PCI Address Decode Control register's VPDDev bit setting (see Table 232 on page 205). Although the PCI specification defines the address to be accessed, as the VPD Address field in the VPD capability list item (15-bit address), the GT–64240A supports remapping of the 17 high bits by setting the PCI Address Decode Control register's VPDHighAddr field to the required address.

For PCI VPD write, the PCI write VPD data first, then writes the VPD address with Flag bit set to '1'. As a response, the slave writes the VPD data to the VPD device (CS[3] or BootCS) to the required address and clears the Flag bit as soon as the write is done.

For a PCI VPD read, the PCI writes VPD address with the Flag bit set to '0'. As response, the slave reads the VPD device from the required address, places the data in the VPD data field, and sets the Flag bit to '1'. The VPD read is treated as a non-prefetched nor delayed read transaction.

8.18.3 Message Signaled Interrupt (MSI)

The MSI feature enables a device to request an interrupt service without using interrupts. The device requests a service by writing a system specified message to a system specified address. The system software initializes the message destination and message during device configuration. The GT–64240A MSI implementation is fully compliant with the PCI specification. It supports a single interrupt message.

The MSI capability structure consists of the following fields:

- Capability structure ID. The ID of MSI capability is 0x5.
- Pointer to next capability structure.
- Message Control.
- Message Address. 32-bit message low address.
- Message Upper Address. 32-bit message high address (in case 64-bit addressing is supported)
- Message data. 15-bit of message data.

Message Control word consists of the following fields:

- bit[0] - MSI Enable. If set to 1, MSI is enabled, and the GT–64240A drives interrupt messages rather than asserting the PCI INT* pin
- bits[3:1] - Multiple Message Capable. Defines the number of DIFFERENT MSI messages the GT–64240A can drive.
- bits[6:4] - Multiple Message Enable. Defines the number of DIFFERENT MSI messages the system allocates for the GT–64240A.
- bit[7] - 64-bit address capable. Enables 64-bit addressing messages.

As soon as PCI enables MSI (set MSI enable bit), GT–64240A will no longer assert interrupts on the PCI bus. Instead, the PCI master will drive a memory write transaction on the PCI bus, with address as specified in Message Address field and data as specified in the Message Data field.

If the Message Upper Address field is set to ‘0’, the master drives a DWORD write, else it drives a DAC DWORD write.

Unlike the PCI INT*, a level sensitive interrupt that is active as long as there are active non-masked interrupts bits set, MSI is an edge like interrupt. However, to prevent the PCI interrupt handler from missing any new interrupt events, the GT–64240A continues to drive new MSI messages while pending, non-masked interrupts exist.

The MSI Timeout register defines the time gap (TClk cycles) between sequential MSI requests. A timer starts counting with each new MSI request. If it reaches ‘0’ and there is still a pending non-masked interrupt, a new MSI request is triggered. If the PCI interrupt handler clears one of the Interrupt Cause register bits and there is still a pending interrupt, the GT–64240A immediately issues a new MSI without waiting for the timeout to expire.

Setting the MSI Timeout register to ‘0’ disables the timer functionality (as if it was programed to infinity). In this case, the PCI interrupt handler must confirm that there are no interrupt event is missed.

NOTE: When programing the MSI Timeout register to a small value, the PCI master transaction queue is repeatedly filled with MSI requests. This prevents CPU or DMA access to the PCI until the PCI interrupt handler clears the interrupt cause bit(s).

8.18.4 Hot Swap

The GT–64240A is CompactPCI Hot-Swap ready compliant. It implements the required configuration registers defined by CompactPCI Hot-Swap specification as well as three required pins.

NOTE: CompactPCI Hot-Swap is supported only on the PCI_0 interface.

The ComactPCI Hot Swap capability structure consists of the following fields:

- Capability structure ID. The ID of HS capability is 0x6.
- Pointer to next capability structure.
- Hot Swap Status and Control.

Hot Swap Status and Control register (HS_CSR) is accessible from both CPU and PCI. This register bits give status of board insertion/extraction as defined in the spec. HS_CSR bits are:

- EIM - ENUM* Interrupt Mask. If set to ‘1’, the GT–64240A won’t assert ENUM* interrupt.
- LOO - LED On/Off. If set to ‘1’ LED is on.
- REM - Removal. Indicates board is about to be extracted.
- INS - Insertion. Indicates board has been inserted.

The GT–64240A supports four Hot-Swap ready required pins:

- HS - Handle Switch input pin. Indicates insertion or extraction of board. A ‘0’ value indicates the handle is open.
- LED - LED control output pin. A ‘1’ value turns the on board LED on.
- ENUM* - open drain output. Asserted upon board insertion or extraction (if not masked by EIM bit).
- 64EN* - PCI 64-bit enable input. Replaces the REQ64* sample on reset deassertion.

NOTE: If not using the GT–64240A in a hot-swap board, the REQ64* pin must be connected to 64EN*.

Board extraction consists of the following steps:

1. The operator opens board ejector handle. As a result, HS goes LOW, indicating board is about to be extracted.
2. As a result, the REM bit is set and the ENUM* pin is asserted, if not masked by EIM bit.
3. The System Hot Swap software detects ENUM* assertion. Checks the REM bits in all Hot-Swappable boards. Identifies the board about to be extracted and clears the REM bit (by writing a ‘1’ value).
4. The GT–64240A acknowledges the system software by stop asserting the ENUM* pin.
5. The Hot Swap software might re-configure the rest of the boards, and when ready, it sets the LOO bit, indicating board is allowed to be removed.
6. As a result, GT–64240A drive LED pin to ‘1’, the on board LED is turned on indicating that the operator may remove the board.

Board insertion consists of the following steps:

1. Board is inserted. It is powered from Early Power and it’s reset is asserted from Local PCI Rst*. The on board LED is turned on by hardware (not as a result of LOO bit state).
2. Local PCI Rst* is deasserted, causing LED to turn off, indicating that the operator may lock the ejector handle.
3. The operator locks the handle. As a result, HS goes HIGH, indicating board is inserted and locked.
4. As a result, INS bit is set and ENUM* is asserted, notifying Hot-Swap software that a board has been

inserted.

5. System Hot Swap software detects ENUM* assertion, checks INS bits in all Hot-Swappable boards, identifies the inserted board and clears INS bit (by writing a value of 1).
6. GT–64240A acknowledges system software by stop asserting ENUM* pin. Now software may re-configure all the boards.

NOTE: For full details on Hot-Swap process and board requirements, see the CompactPCI Hot-Swap specification.

To support HotSwap Ready requirements, the GT–64240A implements a 64EN* input pin. When hot inserting a board, REQ64* cannot be sampled with local reset deassertion in order to identify 64-bit PCI bus, since REQ64* is an active signal on the bus. For this reason, the 64EN* signal is provided. The GT–64240A samples this pin rather than the REQ64* on reset deassertion (local reset) to determine whether it works in a 64-bit PCI environment.

In addition, the GT–64240A supports the following hot swap device requirements:

- All PCI outputs floats when RST* is asserted.
- All GT–64240A PCI state machines are kept in their idle state while RST* is asserted.
- The GT–64240A PCI interface maintains it's idle state until PCI bus is in an IDLE state. If reset is deasserted in the middle of a PCI transaction, the PCI interface stays in it's idle state until the PCI bus is back in idle.
- The GT–64240A has no assumptions on clock behavior prior to it's setup to the rising edge of RST#.
- The GT–64240A is tolerant of the 1V precharge voltage during insertion.
- The GT–64240A can be powered from Early Vcc.

8.18.5 BIST (Built In Self Test)

The GT–64240A supports BIST functionality as defined by the PCI specification. It does not run its own self test. Instead, it enables the PCI to trigger CPU software self test.

The BIST Configuration register is located at offset 0xf of function 0 configuration header. It consists of the following fields:

- BIST Capable bit (bit[7]). If BIST is enabled through reset initialization, it is set to '1'. This bit is read only from the PCI.
- Start BIST bit (bit[6]). Set to '1' by the PCI to trigger CPU software self test. Cleared by the CPU upon test finish.
- Bits[5:4] - Reserved.
- Completion Code (bits[3:0]). Written by the self test software upon test finish. Any value other than '0' stands for test fail.

Upon PCI triggering of BIST (writing '1' to bit[6]), the CPU interrupt is asserted (if not masked) and the CPU interrupt handler must run the system self test. When the test is completed, the CPU software must clear bit[6] and write the completion code.

The PCI specification requires that BIST is completed in two seconds. It is the BIST software responsibility to meet this requirement. If bit[6] is not cleared by two seconds, the PCI BIOS may treat it as BIST failure.

NOTE: The GT–64240A does not runs its own self test. The BIST register implementation is just a software hook for the CPU to run a system self test.

8.19 PCI Interface Registers

The same set of registers are duplicated for both PCI_0 and PCI_1. The only difference is that PCI_0 and PCI_1 registers are located at different offsets.

The PCI_1 interface contains the same set of INTERNAL registers as PCI_0 interface. However, unless specified otherwise, the PCI_1 registers offsets are PCI_0 registers offsets + 0x080. For example, the PCI_0 SCS[0] Size register is located at offset 0xC08. The PCI_1 SCS[0] Size register is located at offset 0xC88.

All PCI CONFIGURATION registers are located at their standard offset in the configuration header, as defined in the PCI spec, when accessed from their corresponding PCI bus. For example, if a master on PCI_0 performs a PCI configuration cycle on PCI's Status and Command Register, the register is located at 0x004. Likewise, if a master on PCI_1 performs a PCI configuration cycle on PCI_1's Status and Command Register, the register is located at 0x004.

On the other hand, if a master on PCI_0 performs a PCI configuration cycle on PCI_1's Status and Command Register, the register is located at 0x084. Likewise, if a master on PCI_1 performs a PCI configuration cycle on PCI's Status and Command Register, the register is located at 0x084.

A CPU access to the GT–64240A's PCI_0 configuration registers is performed via the PCI_0 Configuration Address and PCI_0 Configuration Data registers (internal registers offset 0xcf8 and 0xcfc respectively). A CPU access to the GT–64240A's PCI_1 configuration registers is performed via the PCI_1 Configuration Address and PCI_1 Configuration Data registers (internal registers offset 0xc78 and 0xc7c respectively).

NOTE: A CPU access to GT–64240A PCI_1 configuration registers is not compatible with GT-64120 and GT-64130 devices.

Table 168: PCI Slave Address Decoding Register Map

Register	Offset		Page
	PCI_0	PCI_1	
PCI SCS[0]* BAR Size	0xC08	0xC88	page 192
PCI SCS[1]* BAR Size	0xD08	0xD88	page 193
PCI SCS[2]* BAR Size	0xC0C	0xC8C	page 193
PCI SCS[3]* BAR Size	0xD0C	0xD8C	page 193
PCI CS[0]* BAR Size	0xC10	0xC90	page 193
PCI CS[1]* BAR Size	0xd10	0xd90	page 193
PCI CS[2]* BAR Size	0xd18	0xd98	page 194
PCI CS[3]* BAR Size	0xc14	0xc94	page 194
PCI Boot CS* BAR Size	0xd14	0xd94	page 194
PCI P2P Mem0 BAR Size	0xd1c	0xd9c	page 194
PCI P2P Mem1 BAR Size	0xd20	0xda0	page 194
PCI P2P I/O BAR Size	0xd24	0xda4	page 195
PCI DAC SCS[0]* BAR Size	0xe00	0xe80	page 195

Table 168: PCI Slave Address Decoding Register Map (Continued)

Register	Offset PCI_0	Offset PCI_1	Page
PCI DAC SCS[1]* BAR Size	0xe04	0xe84	page 195
PCI DAC SCS[2]* BAR Size	0xe08	0xe88	page 195
PCI DAC SCS[3]* BAR Size	0xe0c	0xe8c	page 195
PCI DAC CS[0]* BAR Size	0xe10	0xe90	page 196
PCI DAC CS[1]* BAR Size	0xe14	0xe94	page 196
PCI DAC CS[2]* BAR Size	0xe18	0xe98	page 196
PCI DAC CS[3]* BAR Size	0xe1c	0xe9c	page 196
PCI DAC Boot CS* BAR Size	0xe20	0xea0	page 196
PCI DAC P2P Mem0 BAR Size	0xe24	0xe94	page 197
PCI DAC P2P Mem1 BAR Size	0xe28	0xe98	page 197
PCI Expansion ROM BAR Size	0xd2c	0xd9c	page 197
PCI Base Address Registers' Enable	0xc3c	0xcbc	page 197
PCI SCS[0]* Base Address Remap	0xc48	0xcc8	page 200
PCI SCS[1]* Base Address Remap	0xd48	0xdc8	page 200
PCI SCS[2]* Base Address Remap	0xc4c	0xccc	page 200
PCI SCS[3]* Base Address Remap	0xd4c	0xdcc	page 200
PCI CS[0]* Base Address Remap	0xc50	0xcd0	page 200
PCI CS[1]* Base Address Remap	0xd50	0xdd0	page 201
PCI CS[2]* Base Address Remap	0xd58	0xdd8	page 201
PCI CS[3]* Address Remap	0xc54	0xcd4	page 201
PCI Boot CS* Address Remap	0xd54	0xdd4	page 201
PCI P2P Mem0 Base Address Remap (Low)	0xd5c	0xddc	page 201
PCI P2P Mem0 Base Address Remap (High)	0xd60	0xde0	page 202
PCI P2P Mem1 Base Address Remap (Low)	0xd64	0xde4	page 202
PCI P2P Mem1 Base Address Remap (High)	0xd68	0xde8	page 202
PCI P2P I/O Base Address Remap	0xd6c	0xdec	page 202
PCI DAC SCS[0]* Base Address Remap	0xf00	0xf80	page 202
PCI DAC SCS[1]* Base Address Remap	0xf04	0xf84	page 203
PCI DAC SCS[2]* Base Address Remap	0xf08	0xf88	page 203

Table 168: PCI Slave Address Decoding Register Map (Continued)

Register	Offset PCI_0	Offset PCI_1	Page
PCI DAC SCS[3]* Base Address Remap	0xf0C	0xf8C	page 203
PCI DAC CS[0]* Base Address Remap	0xf10	0xf90	page 203
PCI DAC CS[1]* Base Address Remap	0xf14	0xf94	page 203
PCI DAC CS[2]* Base Address Remap	0xf18	0xf98	page 204
PCI DAC CS[3]* Base Address Remap	0xf1c	0xf9c	page 204
PCI DAC BootCS* Base Address Remap	0xf20	0xfa0	page 204
PCI DAC P2P Mem0 Base Address Remap (Low)	0xf24	0xfa4	page 204
PCI DAC P2P Mem0 Base Address Remap (High)	0xf28	0xfa8	page 204
PCI DAC P2P Mem1 Base Address Remap (Low)	0xf2c	0xfac	page 204
PCI DAC P2P Mem1 Base Address Remap (High)	0xf30	0xfb0	page 205
PCI Expansion ROM Base Address Remap	0xf38	0xfb8	page 205
PCI Address Decode Control	0xd3c	0xdbc	page 205

Table 169: PCI Control Register Map

Register	Offset PCI_0	Offset PCI_1	Page
PCI Command	0xc00	0xc80	page 206
PCI Mode	0xd00	0xd80	page 209
PCI Timeout & Retry	0xc04	0xc84	page 210
PCI Read Buffer Discard Timer	0xd04	0xd84	page 211
PCI MSI Trigger Timer	0xc38	0xcb8	page 211
PCI Arbiter Control	0x1d00	0x1d80	page 212
PCI Interface Crossbar Control (Low)	0x1d08	0x1d88	page 213
PCI Interface Crossbar Control (High)	0x1d0c	0x1d8c	page 213
PCI Interface Crossbar Timeout **NOTE:** Reserved for Galileo Technology usage.	0x1d04	0x1d84	page 214

Table 169: PCI Control Register Map (Continued)

Register	Offset PCI_0	Offset PCI_1	Page
PCI Read Response Crossbar Control (Low)	0x1d18	0x1d98	page 215
PCI Read Response Crossbar Control (High)	0x1d1c	0x1d9c	page 216
PCI Sync Barrier Virtual Register	0x1d10	0x1d90	page 216
PCI P2P Configuration	0x1d14	0x1d94	page 216
PCI P2P Swap Control	0x1d54	0x1dd4	page 217
PCI Access Control Base 0 (Low)	0x1e00	0x1e80	page 217
PCI Access Control Base 0 (High)	0x1e04	0x1e84	page 219
PCI Access Control Top 0	0x1e08	0x1e88	page 219
PCI Access Control Base 1 (Low)	0x1e10	0x1e80	page 219
PCI Access Control Base 1 (High)	0x1e14	0x1e94	page 219
PCI Access Control Top 1	0x1e18	0x1e98	page 220
PCI Access Control Base 2 (Low)	0x1e20	0x1ea0	page 220
PCI Access Control Base 2 (High)	0x1e24	0x1ea4	page 220
PCI Access Control Top 2	0x1e28	0x1ea8	page 220
PCI Access Control Base 3 (Low)	0x1e30	0x1eb0	page 220
PCI Access Control Base 3 (High)	0x1e34	0x1eb4	page 221
PCI Access Control Top 3	0x1e38	0x1eb8	page 221
PCI Access Control Base 4 (Low)	0x1e40	0x1ec0	page 221
PCI Access Control Base 4 (High)	0x1e44	0x1ec4	page 221
PCI Access Control Top 4	0x1e48	0x1ec8	page 222
PCI Access Control Base 5 (Low)	0x1e50	0x1ed0	page 222
PCI Access Control Base 5 (High)	0x1e54	0x1ed4	page 222
PCI Access Control Top 5	0x1e58	0x1ed8	page 222
PCI Access Control Base 6 (Low)	0x1e60	0x1ee0	page 223
PCI Access Control Base 6 (High)	0x1e64	0x1ee4	page 223
PCI Access Control Top 6	0x1e68	0x1ee8	page 223
PCI Access Control Base 7 (Low)	0x1e70	0x1ef0	page 223
PCI Access Control Base 7 (High)	0x1e74	0x1ef4	page 224
PCI Access Control Top 7	0x1e78	0x1ef8	page 224

Table 170: PCI Configuration Access Register Map

Register	Offset PCI_0	Offset PCI_1	Page
PCI Configuration Address	0xcf8	0xc78	page 224
PCI Configuration Data Virtual Register	0xcfc	0xc7c	page 225
PCI Interrupt Acknowledge Virtual Register	0xc34	0xcb4	page 225

Table 171: PCI Error Report Register Map

Register	Offset PCI_0	Offset PCI_1	Page
PCI SErr Mask	0xc28	0xca8	page 225
PCI Error Address (Low)	0x1d40	0x1dc0	page 227
PCI Error Address (High)	0x1d44	0x1dc4	page 227
PCI Error Data (Low)	0x1d48	0x1dc8	page 227
PCI Error Data (High)	0x1d4C	0x1dcc	page 227
PCI Error Command	0x1d50	0x1dd0	page 228
PCI Error Cause	0x1d58	0x1dd8	page 228
PCI Error Mask	0x1d5c	0x1ddc	page 231

Table 172: PCI Configuration, Function 0, Register Map

Register	Offset From CPU or PCI_0	Offset from PCI_1	Page
PCI_0 Device and Vendor ID	0x00	0x80	page 233
PCI_1 Device and Vendor ID	0x80	0x00	page 233
PCI_0 Status and Command	0x04	0x84	page 234
PCI_1 Status and Command	0x84	0x04	page 234
PCI_0 Class Code and Revision ID	0x08	0x88	page 236
PCI_1 Class Code and Revision ID	0x88	0x08	page 236
PCI_0 BIST, Header Type, Latency Timer, and Cache Line	0x0c	0x8c	page 236
PCI_1 BIST, Header Type, Latency Timer, and Cache Line	0x8c	0x0c	page 236

Table 172: PCI Configuration, Function 0, Register Map (Continued)

Register	Offset		Page
	From CPU or PCI_0	from PCI_1	
PCI_0 SCS[0]* Base Address	0x10	0x90	page 237
PCI_1 SCS[0]* Base Address	0x90	0x10	page 237
PCI_0 SCS[1]* Base Address	0x14	0x94	page 238
PCI_1 SCS[1]* Base Address	0x94	0x14	page 238
PCI_0 SCS[2]* Base Address	0x18	0x98	page 238
PCI_1 SCS[2]* Base Address	0x98	0x18	page 238
PCI_0 SCS[3]* Base Address	0x1c	0x9c	page 238
PCI_1 SCS[3]* Base Address	0x9c	0x1c	page 238
PCI_0 Internal Registers Memory Mapped Base Address	0x20	0xa0	page 238
PCI_1 Internal Registers Memory Mapped Base Address	0xa0	0x20	page 238
PCI_0 Internal Registers I/O Mapped Base Address	0x24	0xa4	page 239
PCI_1 Internal Registers I/O Mapped Base Address	0xa4	0x24	page 239
PCI_0 Subsystem ID and Subsystem Vendor ID	0x2c	0xac	page 239
PCI_1 Subsystem ID and Subsystem Vendor ID	0xac	0x2c	page 239
PCI_0 Expansion ROM Base Address	0x30	0xb0	page 240
PCI_1 Expansion ROM Base Address	0xb0	0x30	page 240
PCI_0 Capability List Pointer	0x34	0xb4	page 240
PCI_1 Capability List Pointer	0xb4	0x34	page 240
PCI_0 Interrupt Pin and Line	0x3c	0xbc	page 241
PCI_1 Interrupt Pin and Line	0xbc	0x3c	page 241
PCI_0 Power Management Capability	0x40	0xc0	page 241
PCI_1 Power Management Capability	0xc0	0x40	page 241
PCI_0 Power Management Status and Control	0x44	0xc4	page 242
PCI_1 Power Management Status and Control	0xc4	0x44	page 242
PCI_0 VPD Address	0x48	0xc8	page 243
PCI_1 VPD Address	0xc8	0x48	page 243
PCI_0 VPD Data	0x4c	0xcc	page 244
PCI_1 VPD Data	0xcc	0x4c	page 244
PCI_0 MSI Message Control	0x50	0xd0	page 244

Table 172: PCI Configuration, Function 0, Register Map (Continued)

Register	Offset		Page
	From CPU or PCI_0	from PCI_1	
PCI_1 MSI Message Control	0xd0	0x50	page 244
PCI_0 MSI Message Address	0x54	0xd4	page 245
PCI_1 MSI Message Address	0xd4	0x54	page 245
PCI_0 MSI Message Upper Address	0x58	0xd8	page 245
PCI_1 MSI Message Upper Address	0xd8	0x58	page 245
PCI_0 Message Data	0x5c	0xdc	page 245
PCI_1 Message Data	0xdc	0x5c	page 245
PCI_0 CompactPCI Hot Swap Capability	0x60	0xe0	page 246
PCI_1 CompactPCI Hot Swap Capability	0xe0	0x60	page 246

Table 173: PCI Configuration, Function 1, Register Map

Register	Offset		Page
	From CPU or PCI_0	from PCI_1	
PCI_0 CS[0]* Base Address	0x10	0x90	page 246
PCI_1 CS[0]* Base Address	0x90	0x10	page 246
PCI_0 CS[1]* Base Address	0x14	0x94	page 247
PCI_1 CS[1]* Base Address	0x94	0x14	page 247
PCI_0 CS[2]* Base Address	0x18	0x98	page 247
PCI_1 CS[2]* Base Address	0x98	0x18	page 247
PCI_0 CS[3]* Base Address	0x1c	0x9c	page 247
PCI_1 CS[3]* Base Address	0x9c	0x1c	page 247
PCI_0 Boot CS* Base Address	0x20	0xa0	page 247
PCI_1 Boot CS* Base Address	0xa0	0x20	page 247

Table 174: PCI Configuration, Function 2, Register Map

Register	Offset		Page
	From CPU or PCI_0	from PCI_1	
PCI_0 P2P Mem0 Base Address	0x10	0x90	page 248
PCI_1 P2P Mem0 Base Address	0x90	0x10	page 248
PCI_0 P2P Mem1 Base Address	0x14	0x94	page 248
PCI_1 P2P Mem1 Base Address	0x94	0x14	page 248
PCI_0 P2P I/O Base Address	0x18	0x98	page 248
PCI_1 P2P I/O Base Address	0x98	0x98	page 248

Table 175: PCI Configuration, Function 4, Register Map

Register	Offset		Page
	From CPU or PCI_0	from PCI_1	
PCI_0 DAC SCS[0]* Base Address (Low)	0x10	0x90	page 249
PCI_1 DAC SCS[0]* Base Address (Low)	0x90	0x10	page 249
PCI_0 DAC SCS[0]* Base Address (High)	0x14	0x94	page 249
PCI_1 DAC SCS[0]* Base Address (High)	0x94	0x14	page 249
PCI_0 DAC SCS[1]* Base Address (Low)	0x18	0x98	page 249
PCI_1 DAC SCS[1]* Base Address (Low)	0x98	0x18	page 249
PCI_0 DAC SCS[1]* Base Address (High)	0x1c	0x9c	page 250
PCI_1 DAC SCS[1]* Base Address (High)	0x9c	0x1c	page 250
PCI_0 DAC P2P Mem0 Base Address (Low)	0x20	0xa0	page 250
PCI_1 DAC P2P Mem0 Base Address (Low)	0xa0	0x20	page 250
PCI_0 DAC P2P Mem0 Base Address (High)	0x24	0xa4	page 250
PCI_1 DAC P2P Mem0 Base Address (High)	0xa4	0x24	page 250

Table 176: PCI Configuration, Function 5, Register Map

Register	Offset From CPU or PCI_0	Offset From PCI_1	Page
PCI_0 DAC SCS[2]* Base Address (Low)	0x10	0x90	page 251
PCI_1 DAC SCS[2]* Base Address (Low)	0x90	0x10	page 251
PCI_0 DAC SCS[2]* Base Address (High)	0x14	0x94	page 251
PCI_1 DAC SCS[2]* Base Address (High)	0x94	0x14	page 251
PCI_0 DAC SCS[3]* Base Address (Low)	0x18	0x98	page 251
PCI_1 DAC SCS[3]* Base Address (Low)	0x98	0x18	page 251
PCI_0 DAC SCS[3]* Base Address (High)	0x1c	0x9c	page 252
PCI_1 DAC SCS[3]* Base Address (High)	0x9c	0x1c	page 252
PCI_0 DAC P2P Mem1 Base Address (Low)	0x20	0xa0	page 252
PCI_1 DAC P2P Mem1 Base Address (Low)	0xa0	0x20	page 252
PCI_0 DAC P2P Mem1 Base Address (High)	0x24	0xa4	page 252
PCI_1 DAC P2P Mem1 Base Address (High)	0xa4	0x24	page 252

Table 177: PCI Configuration, Function 6, Register Map

Register	Offset From CPU or PCI_0	Offset From PCI_1	Page
PCI_0 DAC CS[0]* Base Address (Low)	0x10	0x90	page 253
PCI_1 DAC CS[0]* Base Address (Low)	0x90	0x10	page 253
PCI_0 DAC CS[0]* Base Address (High)	0x14	0x94	page 253
PCI_1 DAC CS[0]* Base Address (High)	0x94	0x14	page 253
PCI_0 DAC CS[1]* Base Address (Low)	0x18	0x98	page 253
PCI_1 DAC CS[1]* Base Address (Low)	0x98	0x18	page 253
PCI_0 DAC CS[1]* Base Address (High)	0x1c	0x9c	page 254
PCI_1 DAC CS[1]* Base Address (High)	0x9c	0x1c	page 254
PCI_0 DAC CS[2]* Base Address (Low)	0x20	0xa0	page 254
PCI_1 DAC CS[2]* Base Address (Low)	0xa0	0x20	page 254

Table 177: PCI Configuration, Function 6, Register Map (Continued)

Register	Offset		Page
	From CPU or PCI_0	From PCI_1	
PCI_0 DAC CS[2]* Base Address (High)	0x24	0xa4	page 254
PCI_1 DAC CS[2]* Base Address (High)	0xa4	0x24	page 254

Table 178: PCI Configuration, Function 7, Register Map

Register	Offset		Page
	From CPU or PCI_0	from PCI_1	
PCI_0 DAC CS[3]* Base Address (Low)	0x10	0x90	page 255
PCI_1 DAC CS[3]* Base Address (Low)	0x90	0x10	page 255
PCI_0 DAC CS[3]* Base Address (High)	0x14	0x94	page 255
PCI_1 DAC CS[3]* Base Address (High)	0x94	0x14	page 255
PCI_0 DAC Boot CS* Base Address (Low)	0x18	0x98	page 255
PCI_1 DAC Boot CS* Base Address (Low)	0x98	0x18	page 255
PCI_0 DAC Boot CS* Base Address (High)	0x1c	0x9c	page 256
PCI_1 DAC Boot CS* Base Address (High)	0x9c	0x1c	page 256

8.19.1 PCI Slave Address Decoding Registers

Table 179: PCI SCS[0] BAR Size

- PCI_0 Offset: 0xc08
- PCI_1 Offset: 0xc88

Bits	Field Name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	BARSize	SCS[0]* BAR Address Bank Size Must be programed from LSB to MSB as sequence of '1s' followed by sequence of '0s'. BAR size is in 4Kbyte resolution. For example, a 0x00FF.F000 size register value represents a BAR size of 16Mbyte.	0x007ff

Table 180: PCI SCS[1]* BAR Size

- PCI_0 Offset: 0xd08
- PCI_1 Offset: 0xd88

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 181: PCI SCS[2]* BAR Size

- PCI_0 Offset: 0xc0c
- PCI_1 Offset: 0xc8c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 182: PCI SCS[3]* BAR Size

- PCI_0 Offset: 0xd0c
- PCI_1 Offset: 0xd8c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 183: PCI CS[0]* BAR Size

- PCI_0 Offset: 0xc10
- PCI_1 Offset: 0xc90

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 184: PCI CS[1]* BAR Size

- PCI_0 Offset: 0xd10
- PCI_1 Offset: 0xd90

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 185: PCI CS[2]* BAR Size

- PCI_0 Offset: 0xd18
- PCI_1 Offset: 0xd98

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x00fff000

Table 186: PCI CS[3]* BAR Size

- PCI_0 Offset: 0xc14
- PCI_1 Offset: 0xc94

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 187: PCI Boot CS* BAR Size

- PCI_0 Offset: 0xd14
- PCI_1 Offset: 0xd94

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 188: PCI P2P Mem0 BAR Size

- PCI_0 Offset: 0xd1c
- PCI_1 Offset: 0xd9c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size	0x01fff000

Table 189: PCI P2P Mem1 BAR Size

- PCI_0 Offset: 0xd20
- PCI_1 Offset: 0xda0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x01fff000

Table 190: PCI P2P I/O BAR Size

- PCI_0 Offset: 0xd24
- PCI_1 Offset: 0xda4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x01fff000

Table 191: PCI DAC SCS[0] BAR Size

- PCI_0 Offset: 0xe00
- PCI_1 Offset: 0xe80

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 192: PCI DAC SCS[1] BAR Size

- PCI_0 Offset: 0xe04
- PCI_1 Offset: 0xe84

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 193: PCI DAC SCS[2] BAR Size

- PCI_0 Offset: 0xe08
- PCI_1 Offset: 0xe88

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 194: PCI DAC SCS[3] BAR Size

- PCI_0 Offset: 0xe0c
- PCI_1 Offset: 0xe8c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 195: PCI DAC CS[0]* BAR Size

- PCI_0 Offset: 0xe10
- PCI_1 Offset: 0xe90

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 196: PCI DAC CS[1]* BAR Size

- PCI_0 Offset: 0xe14
- PCI_1 Offset: 0xe94

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 197: PCI DAC CS[2]* BAR Size

- PCI_0 Offset: 0xe18
- PCI_1 Offset: 0xe98

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x00fff000

Table 198: PCI DAC CS[3]* BAR Size

- PCI_0 Offset: 0xe1c
- PCI_1 Offset: 0xe9c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 199: PCI DAC BootCS* BAR Size

- PCI_0 Offset: 0xe20
- PCI_1 Offset: 0xea0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 200: PCI DAC P2P Mem0 BAR Size

- PCI_0 Offset: 0xe24
- PCI_1 Offset: 0xea4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x01fff000

Table 201: PCI DAC P2P Mem1 BAR Size

- PCI_0 Offset: 0xe28
- PCI_1 Offset: 0xea8

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x01fff000

Table 202: PCI Expansion ROM BAR Size

- PCI_0 Offset: 0xd2c
- PCI_1 Offset: 0xdac

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 203: PCI Base Address Registers Enable

- PCI_0 Offset: 0xc3c
- PCI_1 Offset: 0xcbc

Bits	Field Name	Function	Initial Value
0	SCS0En	SCS[0]* BAR Enable 0 - Enable 1 - Disable	0x0
1	SCS1En	SCS[1]* BAR Enable 0 - Enable 1 - Disable	0x0
2	SCS2En	SCS[2]* BAR Enable 0 - Enable 1 - Disable	0x0
3	SCS3En	SCS[3]* BAR Enable 0 - Enable 1 - Disable	0x0

Table 203: PCI Base Address Registers Enable (Continued)

- PCI_0 Offset: 0xc3c
- PCI_1 Offset: 0xcbc

Bits	Field Name	Function	Initial Value
4	CS0En	CS[0]* BAR Enable 0 - Enable 1 - Disable	0x0
5	CS1En	CS[1]* BAR Enable 0 - Enable 1 - Disable	0x0
6	CS2En	CS[2]* BAR Enable 0 - Enable 1 - Disable	0x0
7	CS3En	CS[3]* BAR Enable 0 - Enable 1 - Disable	0x0
8	BootCSEn	BootCS* BAR Enable 0 - Enable 1 - Disable	0x0
9	IntMemEn[1]	Memory Mapped Internal Registers BAR Enable 0 - Enable 1 - Disable	0x0
10	IntIOEn	I/O Mapped Internal Registers BAR Enable 0 - Enable 1 - Disable	0x1
11	P2PMem0En	P2P Mem0 BAR Enable 0 - Enable 1 - Disable	0x1
12	P2PMem1En	P2P Mem1 BAR Enable 0 - Enable 1 - Disable	0x1
13	P2PIOEn	P2P IO BAR Enable 0 - Enable 1 - Disable	0x1
14	Reserved	Must be 1.	0x1
15	DSCS0En	DAC SCS[0]* BAR Enable 0 - Enable 1 - Disable	0x1

Table 203: PCI Base Address Registers Enable (Continued)

- PCI_0 Offset: 0xc3c
- PCI_1 Offset: 0xcbc

Bits	Field Name	Function	Initial Value
16	DSCS1En	DAC SCS[1]* BAR Enable 0 - Enable 1 - Disable	0x1
17	DSCS2En	DAC SCS[2]* BAR Enable 0 - Enable 1 - Disable	0x1
18	DSCS3En	DAC SCS[3]* BAR Enable 0 - Enable 1 - Disable	0x1
19	DCS0En	DAC CS[0]* BAR Enable 0 - Enable 1 - Disable	0x1
20	DCS1En	DAC CS[1]* BAR Enable 0 - Enable 1 - Disable	0x1
21	DCS2En	DAC CS[2]* BAR Enable 0 - Enable 1 - Disable	0x1
22	DCS3En	DAC CS[3]* BAR Enable 0 - Enable 1 - Disable	0x1
23	DBootCSEn	DAC BootCS* BAR Enable 0 - Enable 1 - Disable	0x1
24	DP2PMem0En	DAC P2P Mem0 BAR Enable 0 - Enable 1 - Disable	0x1
25	DP2PMem1En	DAC P2P Mem1 BAR Enable 0 - Enable 1 - Disable	0x1
26	Reserved	Must be 1.	0x1
31:27	Reserved	Reserved.	0x1f

1. The GT–64240A prevents disabling both memory mapped and I/O mapped BARs (bits 9 and 10 cannot simultaneously be set to 1).

Table 204: PCI SCS[0]* Base Address Remap

- PCI_0 Offset: 0xc48
- PCI_1 Offset: 0xcc8

Bits	Field Name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	SCS0Remap	SCS[0]* BAR Remap Address	0x0

Table 205: PCI SCS[1]* Base Address Remap

- PCI_0 Offset: 0xd48
- PCI_1 Offset: 0xdc8

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x00800000

Table 206: PCI SCS[2]* Base Address Remap

- PCI_0 Offset: 0xc4c
- PCI_1 Offset: 0xccc

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x01000000

Table 207: PCI SCS[3]* Base Address Remap

- PCI_0 Offset: 0xd4c
- PCI_1 Offset: 0xdcc

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x01800000

Table 208: PCI CS[0]* Base Address Remap

- PCI_0 Offset: 0xc50
- PCI_1 Offset: 0xcd0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1c000000

Table 209: PCI CS[1]* Base Address Remap

- PCI_0 Offset: 0xd50
- PCI_1 Offset: 0xdd0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1c800000

Table 210: PCI CS[2]* Base Address Remap

- PCI_0 Offset: 0xd58
- PCI_1 Offset: 0xdd8

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1d000000

Table 211: PCI CS[3]* Base Address Remap

- PCI_0 Offset: 0xc54
- PCI_1 Offset: 0xcd4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1f00000

Table 212: PCI BootCS* Base Address Remap

- PCI_0 Offset: 0xd54
- PCI_1 Offset: 0xdd4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1f800000

Table 213: PCI P2P Mem0 Base Address Remap (Low)

- PCI_0 Offset: 0xd5c
- PCI_1 Offset: 0xddc

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	PCI_0: 0x22000000 PCI_1: 0x12000000

Table 214: PCI P2P Mem0 Base Address Remap (High)

- PCI_0 Offset: 0xd60
- PCI_1 Offset: 0xde0

Bits	Field Name	Function	Initial Value
31:0	P2P0Remap	P2P Mem0 BAR Remap Address.	0x0

Table 215: PCI P2P Mem1 Base Address Remap (Low)

- PCI_0 Offset: 0xd64
- PCI_1 Offset: 0xde4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	PCI_0: 0x24000000 PCI_1: 0xf2000000

Table 216: PCI P2P Mem1 Base Address Remap (High)

- PCI_0 Offset: 0xd68
- PCI_1 Offset: 0xde8

Bits	Field Name	Function	Initial Value
31:0	P2P1Remap	P2P Mem1 BAR Address Remap.	0x0

Table 217: PCI P2P I/O Base Address Remap

- PCI_0 Offset: 0xd6c
- PCI_1 Offset: 0xdec

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	PCI_0: 0x20000000 PCI_1: 0x10000000

Table 218: PCI DAC SCS[0]* Base Address Remap

- PCI_0 Offset: 0xf00
- PCI_1 Offset: 0xf80

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x0

Table 219: PCI DAC SCS[1]* Base Address Remap

- PCI_0 Offset: 0xf04
- PCI_1 Offset: 0xf84

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x00800000

Table 220: PCI DAC SCS[2]* Base Address Remap

- PCI_0 Offset: 0xf08
- PCI_1 Offset: 0xf88

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x01000000

Table 221: PCI DAC SCS[3]* Base Address Remap

- PCI_0 Offset: 0xf0c
- PCI_1 Offset: 0xf8c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x01800000

Table 222: PCI DAC CS[0]* Base Address Remap

- PCI_0 Offset: 0xf10
- PCI_1 Offset: 0xf90

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1c000000

Table 223: PCI DAC CS[1]* Base Address Remap

- PCI_0 Offset: 0xf14
- PCI_1 Offset: 0xf94

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1c800000

Table 224: PCI DAC CS[2]* Base Address Remap

- PCI_0 Offset: 0xf18
- PCI_1 Offset: 0xf98

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1d000000

Table 225: PCI DAC CS[3]* Base Address Remap

- PCI_0 Offset: 0xf1c
- PCI_1 Offset: 0xf9c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1f000000

Table 226: PCI DAC BootCS* Base Address Remap

- PCI_0 Offset: 0xf20
- PCI_1 Offset: 0xfa0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1f800000

Table 227: PCI DAC P2P Mem0 Base Address Remap (Low)

- PCI_0 Offset: 0xf24
- PCI_1 Offset: 0xfa4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	PCI_0: 0x26000000 PCI_1: 0xf4000000

Table 228: PCI DAC P2P Mem0 Base Address Remap (High)

- PCI_0 Offset: 0xf28
- PCI_1 Offset: 0xfa8

Bits	Field Name	Function	Initial Value
31:0	P2P0Remap	DAC P2P Mem0 BAR Address Remap.	0x0

Table 229: PCI DAC P2P Mem1 Base Address Remap (Low)

- PCI_0 Offset: 0xf2c
- PCI_1 Offset: 0xfac

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	PCI_0: 0x28000000 PCI_1: 0xf6000000

Table 230: PCI DAC P2P Mem1 Base Address Remap (High)

- PCI_0 Offset: 0xf30
- PCI_1 Offset: 0xfb0

Bits	Field Name	Function	Initial Value
31:0	P2P1Remap	DAC P2P Mem1 BAR Address Remap.	0x0

Table 231: PCI Expansion ROM Base Address Remap

- PCI_0 Offset: 0xf38
- PCI_1 Offset: 0xfb8

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1f000000

Table 232: PCI Address Decode Control

- PCI_0 Offset: 0xd3c
- PCI_1 Offset: 0xdbc

Bits	Field Name	Function	Initial Value
0	RemapWrDis	Address Remap Registers Write Disable 0 - Writes to a BAR result in updating the corresponding remap register with the BAR's new value. 1 - Writes to a BAR have no affect on the corresponding Remap register value.	0x0
1	ExpRomDev	Expansion ROM Device 0 - CS[3]* 1 - BootCS*	0x0
2	VPDDev	VPD Device 0 - CS[3]* 1 - BootCS*	0x0

Table 232: PCI Address Decode Control (Continued)

- PCI_0 Offset: 0xd3c
- PCI_1 Offset: 0xdbc

Bits	Field Name	Function	Initial Value
3	MsgAcc	Messaging registers access 0 - Messaging unit registers are accessible on lowest 4Kbyte of SCS[0] BAR space. 1 - Messaging unit registers are only accessible as part of the GT–64240A internal space. **NOTE:** If using I20 queues, must be set to '0'.	0x1
7:4	Reserved	Reserved.	0x0
24:8	VPDHighAddr	VPD High Address bits [31:15] of VPD the address.	0x0
31:25	Reserved	Reserved.	0x0

8.19.2 PCI Control Registers

Table 233: PCI Command

- PCI_0 Offset: 0xc00
- PCI_1 Offset: 0xc80

Bits	Field Name	Function	Initial Value
0	MByteSwap	PCI Master Byte Swap When set to '0', the GT–64240A PCI master swaps the bytes of the incoming and outgoing PCI data (swap the 8 bytes of a long-word). **NOTE:** GT-64120 and GT-64130 compatible.	AD[4] sampled at reset.
1	Reserved	Read Only.	0x0
2	Reserved	Must be 0.	0x0
3	Reserved	Read Only.	0x0
4	MWrCom	PCI Master Write Combine Enable When set to '1', write combining is enabled.	0x1
5	MRdCom	PCI Master Read Combine Enable When set to '1', read combining is enabled.	0x1

Table 233: PCI Command (Continued)

- PCI_0 Offset: 0xc00
- PCI_1 Offset: 0xc80

Bits	Field Name	Function	Initial Value
6	MWrTrig	PCI Master Write Trigger 0 - Accesses the PCI bus only when the whole burst is written into the master write buffer. 1 - Accesses the PCI bus when the first data is written into the master write buffer.	0x1
7	MRdTrig	PCI Master Read Trigger 0 - Returns read data to the initiating unit only when the whole burst is written into master read buffer. 1 - Returns read data to the initiating unit when the first read data is written into master read buffer.	0x0
8	MRdLine	PCI Master Memory Read Line Enable 0 - Disable 1 - Enable	0x1
9	MRdMul	PCI Master Memory Read Multiple Enable 0 - Disable 1 - Enable	0x1
10	MWordSwap	PCI Master Word Swap When set to '1', the GT–64240A PCI master swaps the 32-bit words of the incoming and outgoing PCI data. **NOTE:** GT-64120 and GT-64130 compatible.	0x0
11	SWordSwap	PCI Slave Word Swap When set to '1', the GT–64240A PCI slave swaps the 32-bit words of the incoming and outgoing PCI data. **NOTE:** GT-64120 and GT-64130 compatible.	0x0
12	Reserved	Reserved.	0x1
13	SBDis	PCI Slave Sync Barrier Disable When set to '1', the PCI configuration read transaction will stop act as sync barrier transaction.	0x0
14	Reserved	Must be 0	0x0
15	MReq64	PCI Master REQ64* Enable 0 - Disable 1 - Enable	0x1
16	SByteSwap	PCI Slave Byte Swap When set to '0', the GT–64240A PCI slave swaps the bytes of the incoming and outgoing PCI data (swap the 8 bytes of a long-word). **NOTE:** GT-64120 and GT-64130 compatible.	Sampled at reset on AD[4]

Table 233: PCI Command (Continued)

- PCI_0 Offset: 0xc00
- PCI_1 Offset: 0xc80

Bits	Field Name	Function	Initial Value
17	MDACEn	PCI Master DAC Enable 0 - Disable The PCI master never drives the DAC cycle. 1 - Enable In case the upper 32-bit address is not '0', the PCI master drives the DAC cycle.	0x1
18	M64Allign	PCI Master REQ64* assertion on non-aligned 0 - Disable The master asserts REQ64* only if the address is 64-bit aligned. 1 - Enable The master asserts REQ64* even if the address is not 64-bit aligned.	0x1
19	PErrProp	Parity/ECC Errors Propagation Enable 0 - Disable The PCI interface always drives correct parity on the PAR signal. 1 - Enable In case of slave read bad ECC from SDRAM, or master write with bad parity/ECC indication from the initiator, the PCI interface drives bad parity on the PAR signal.	0x0
20	SSwapEn	PCI Slave Swap Enable 0 - PCI slave data swapping is determined via SByteSwap and SWordSwap bits (bits 16 and 11), as in the GT-64120/130. 1 - PCI slave data swapping is determined via PCISwap bits [25:24] in the PCI Access Control registers. **NOTE:** Even if the SSwapEn bit is set to '1'and the PCI address does not match any of the Access Control registers, slave data swapping works according to SByteSwap and SWordSwap bits.	0x0
21	MSwapEn	PCI Master Swap Enable 0 - PCI master data swapping is determined via MByteSwap and MWordSwap bits (bits 0 and 10), as in the GT-64120/130. 1 - PCI master data swapping is determined via PCISwap bits in CPU to PCI Address Decoding registers.	0x0

Table 233: PCI Command (Continued)

- PCI_0 Offset: 0xc00
- PCI_1 Offset: 0xc80

Bits	Field Name	Function	Initial Value
22	MIntSwapEn	PCI Master Configuration Transactions Data Swap Enable 0 - Disable The PCI master configuration transaction to the PCI bus is always in Little Endian convention. 1 - Enable The PCI master configuration transaction to the PCI bus is determined according to the setting of MSwapEn bit. **NOTE:** Reserved for Galileo Technology usage.	0x0
23	LBEn	PCI "Loop Back" Enable 0 - Disable The PCI slave does not respond to transactions initiated by the PCI master. 1 - Enable The PCI slave does respond to transactions initiated by the PCI master, if targeted to the slave (address match). **NOTE:** Reserved for Galileo Technology usage.	0x0
26:24	SIntSwap	PCI Slave data swap control on PCI accesses to the GT–64240A internal and configuration registers. Bits encoding are the same as bits[26:24] in PCI Access Control registers.	Bits[26:25]: 0x0 Bit[24]: sampled at reset on AD[4]
27	Reserved	Must be 0.	0x0
31:28	Reserved	Read only.	0x0

Table 234: PCI Mode

- PCI_0 Offset: 0xd00
- PCI_1 Offset: 0xd80

Bits	Field Name	Function	Initial Value
0	PciID	PCI Interface ID Read Only 0.	PCI_0: 0x0 PCI_1: 0x1
1	Reserved	Reserved.	0x0
2	Pci64	64-bit PCI Interface When set to '1', the PCI interface is configured to a 64-bit interface. Read Only.	Sampled at reset. PCI_0: 64EN* PCI_1: REQ641*

Table 234: PCI Mode (Continued)

- PCI_0 Offset: 0xd00
- PCI_1 Offset: 0xd80

Bits	Field Name	Function	Initial Value
7:3	Reserved	Reserved.	0x0
8	ExpRom	Expansion ROM Enable When set to '1', the expansion ROM BAR is enabled. Read Only from PCI.	Sampled at reset. PCI_0: AD[17] PCI_1: AD[18]
9	VPD	VPD Enable When set to '1', VPD is supported. Read Only from PCI.	0x1
10	MSI	MSI Enable When set to '1', MSI is supported. Read Only from PCI.	0x1
11	PMG	Power Management Enable When set to '1', PMG is supported. Read Only from PCI.	0x1
12	HotSwap	CompactPCI Hot Swap Enable When set to '1', HotSwap is supported. Read Only from PCI.	0x1
13	BIST	BIST Enable If set to '1', BIST is enabled. Read only from PCI.	0x1
30:14	Reserved	Reserved.	0x0
31	PRst	PCI Interface Reset Indication Set to '0' as long as the RST* pin is asserted. Read Only.	Reset initialization.

Table 235: PCI Timeout and Retry

- PCI_0 Offset: 0xc04
- PCI_1 Offset: 0xc84

Bits	Field Name	Function	Initial Value
7:0	Timeout0	Specifies the number of PClk cycles the GT–64240A slave holds the PCI bus before terminating a transaction with RETRY.	0x0f
15:8	Timeout1	Specifies the number of PClk cycles the GT–64240A slave holds the PCI bus before terminating a transaction with DISCONNECT.	0x07

Table 235: PCI Timeout and Retry (Continued)

- PCI_0 Offset: 0xc04
- PCI_1 Offset: 0xc84

Bits	Field Name	Function	Initial Value
23:16	RetryCtr	Retry Counter Specifies the number of retries of the GT–64240A Master. The GT–64240A generates an interrupt when this timer expires. A 0x00 value means a “retry forever”.	0x0
31:24	Reserved	Reserved.	0x0

Table 236: PCI Read Buffer Discard Timer

- PCI_0 Offset: 0xd04
- PCI_1 Offset: 0xd84

Bits	Field Name	Function	Initial Value
15:0	Timer	Specifies the number of PClk cycles the GT–64240A slave keeps an non-accessed read buffers (non completed delayed read) before invalidating the buffer.	0xffff
23:16	RdBufEn	Slave Read Buffers Enable Each bit corresponds to one of the eight read buffers. If set to ‘1’, buffer is enabled.	0xff
31:24	Reserved	Reserved.	0x0

Table 237: PCI MSI Trigger Timer

- PCI_0 Offset: 0xc38
- PCI_1 Offset: 0xcb8

Bits	Field Name	Function	Initial Value
15:0	Timer	Specifies the number of TClk cycles between consecutive MSI requests.	0xffff
31:16	Reserved	Reserved.	0x0

Table 238: PCI Arbiter Control

- PCI_0 Offset: 0x1d00
- PCI_1 Offset: 0x1d80

Bits	Field Name	Function	Initial Value
0	Reserved	Must be '0'.	0x0
1	BDEn	Broken Detection Enable If set to '1', broken master detection is enabled. A master is said to be broken if it fails to respond to grant assertion within a window specified in BV (bits [6:3]).	0x0
2	Reserved	Reserved.	0x0
6:3	BV	Broken Value This value sets the maximum number of cycles that the arbiter waits for a PCI master to respond to its grant assertion. If a PCI master fails to assert FRAME* within this time, the PCI arbiter aborts the transaction and performs a new arbitration cycle and a maskable interrupt is generated. Must be greater than 0. **NOTE:** The PCI arbiter waits for the current transaction to end before starting to count the wait-for-broken cycles. Must be greater than '1' for masters that performs address stepping (such as the GT–64240A PCI master), since they require GNT* assertion for two cycles.	0x0
13:7	Reserved	Reserved.	0x0
20:14	PD[6:0]	Parking Disable Use these bits to disable parking on any of the PCI masters. When a PD bit is set to '1', parking on the associated PCI master is disabled. **NOTE:** The arbiter parks on the last master granted unless disabled through the PD bit. Also, if PD bits are all '1', the PCI arbiter parks on the internal PCI master.	0x0
30:21	Reserved	Reserved.	0x0
31	EN	Enable Setting this bit to '1' enables operation of the arbiter.	0x0

Table 239: PCI Interface Crossbar Control (Low)

- PCI_0 Offset: 0x1d08
- PCI_1 Offset: 0x1d88

Bits	Field Name	Function	Initial Value
3:0	Arb0	Slice 0 of the PCI master “pizza” arbiter. 0x0,0x1 - Reserved 0x2 - CPU access 0x3 - PCI_0: NULL request PCI_1: PCI_0 access 0x4 - PCI_0: PCI_1 access PCI_1: NULL request 0x5 - Comm unit access 0x6 - IDMA channels 0/1/2/3 access 0x7 - IDMA channels 4/5/6/7 access 0x8 - 0xf - Reserved	0x2
7:4	Arb1	Slice 1 of PCI master “pizza” arbiter.	PCI_0: 0x4 PCI_1: 0x3
11:8	Arb2	Slice 2 of PCI master “pizza” arbiter.	0x5
15:12	Arb3	Slice 3 of PCI master “pizza” arbiter.	0x6
19:16	Arb4	Slice 4 of PCI master “pizza” arbiter.	0x7
23:20	Arb5	Slice 5 of PCI master “pizza” arbiter.	PCI_0: 0x3 PCI_1: 0x4
27:24	Arb6	Slice 6 of PCI master “pizza” arbiter.	PCI_0: 0x3 PCI_1: 0x4
31:28	Arb7	Slice 7 of PCI master “pizza” arbiter.	PCI_0: 0x3 PCI_1: 0x4

Table 240: PCI Interface Crossbar Control (High)

- PCI_0 Offset: 0x1d0c
- PCI_1 Offset: 0x1d8c

Bits	Field Name	Function	Initial Value
3:0	Arb8	Slice 8 of PCI master “pizza” arbiter.	0x2
7:4	Arb9	Slice 9 of PCI master “pizza” arbiter.	PCI_0: 0x4 PCI_1: 0x3
11:8	Arb10	Slice 10 of PCI master “pizza” arbiter.	0x5
15:12	Arb11	Slice 11 of PCI master “pizza” arbiter.	0x6
19:16	Arb12	Slice 12 of PCI master “pizza” arbiter.	0x7

Table 240: PCI Interface Crossbar Control (High) (Continued)

- PCI_0 Offset: 0x1d0c
- PCI_1 Offset: 0x1d8c

Bits	Field Name	Function	Initial Value
23:20	Arb13	Slice 13 of PCI master “pizza” arbiter.	PCI_0: 0x3 PCI_1: 0x4
27:24	Arb14	Slice 14 of PCI master “pizza” arbiter.	PCI_0: 0x3 PCI_1: 0x4
31:28	Arb15	Slice 15 of PCI master “pizza” arbiter.	PCI_0: 0x3 PCI_1: 0x4

Table 241: PCI Interface Crossbar Timeout

- PCI_0 Offset: 0x1d04
- PCI_1 Offset: 0x1d84

NOTE: Reserved for Galileo Technology usage.

Bits	Field Name	Function	Initial Value
7:0	Timeout	Crossbar Arbiter Timeout Preset Value	0xff
15:8	Reserved	Reserved	0x0
16	TimeoutEn	Crossbar Arbiter Timer Enable 0 - Enable 1 - Disable	0x1
31:17	Reserved	Reserved.	0x0

Table 242: PCI Read Response Crossbar Control (Low)

- PCI_0 Offset: 0x1d18
- PCI_1 Offset: 0x1d98

Bits	Field Name	Function	Initial Value
3:0	Arb0	Slice 0 of PCI slave “pizza” arbiter. 0x0 - SDRAM read data 0x1 - Device read data 0x2 - Reserved 0x3 - PCI_0: NULL PCI_1: PCI_0 read data 0x4 - PCI_0: PCI_1 read data PCI_1: NULL 0x5 - Comm unit internal registers read data 0x6 - IDMA 0/1/2/3 internal registers read data 0x7 - IDMA 4/5/6/7 internal registers read data 0x8 - 0xf - Reserved	0x0
7:4	Arb1	Slice 1 of PCI slave “pizza” arbiter.	0x1
11:8	Arb2	Slice 2 of PCI slave “pizza” arbiter.	0x2
15:12	Arb3	Slice 3 of PCI slave “pizza” arbiter.	PCI_0: 0x4 PCI_1: 0x3
19:16	Arb4	Slice 4 of PCI slave “pizza” arbiter.	0x5
23:20	Arb5	Slice 5 of PCI slave “pizza” arbiter.	0x6
27:24	Arb6	Slice 6 of PCI slave “pizza” arbiter.	0x7
31:28	Arb7	Slice 7 of PCI slave “pizza” arbiter.	PCI_0: 0x3 PCI_1: 0x4

Table 243: PCI Read Response Crossbar Control (High)

- PCI_0 Offset: 0x1d1c
- PCI_1 Offset: 0x1d9c

Bits	Field Name	Function	Initial Value
3:0	Arb8	Slice 8 of PCI slave “pizza” arbiter.	0x0
7:4	Arb9	Slice 9 of PCI slave “pizza” arbiter.	0x1
11:8	Arb10	Slice 10 of PCI slave “pizza” arbiter.	0x2
15:12	Arb11	Slice 11 of PCI slave “pizza” arbiter.	PCI_0: 0x4 PCI_1: 0x3
19:16	Arb12	Slice 12 of PCI slave “pizza” arbiter.	0x5
23:20	Arb13	Slice 13 of PCI slave “pizza” arbiter.	0x6
27:24	Arb14	Slice 14 of PCI slave “pizza” arbiter.	0x7
31:28	Arb15	Slice 15 of PCI slave “pizza” arbiter.	PCI_0: 0x3 PCI_1: 0x4

Table 244: PCI Sync Barrier Virtual Register

- PCI_0 Offset: 0x1d10
- PCI_1 Offset: 0x1d90

Bits	Field Name	Function	Initial Value
31:0	SyncReg	Sync Barrier Virtual Register PCI read from this register results in PCI slave sync barrier action. The returned data is un-deterministic. Read Only.	0x0

Table 245: PCI P2P Configuration

- PCI_0 Offset: 0x1d14
- PCI_1 Offset: 0x1d94

Bits	Field Name	Function	Initial Value
7:0	2ndBusL	Secondary PCI Interface Bus Range Lower Boundary	0xff
15:8	2ndBusH	Secondary PCI Interface Bus Range Upper Boundary	0x0
23:16	BusNum	The PCI bus number to which the PCI interface is connected.	0x0
28:24	DevNum	The PCI interface’s device number.	0x0
31:29	Reserved	Reserved.	0x0

Table 246: PCI P2P Swap Control

- PCI_0 Offset: 0x1d54
- PCI_1 Offset: 0x1dd4

Bits	Field Name	Function	Initial Value
2:0	M0Sw	P2P Mem0 BAR Swap Control	0x1
3	M0Req64	P2P Mem0 BAR Force REQ64	0x1
6:4	M1Sw	P2P Mem1 BAR Swap Control	0x1
7	M1Req64	P2P Mem1 BAR Force REQ64	0x1
10:8	DM0Sw	P2P DAC Mem0 BAR Swap Control	0x1
11	DM0Req64	P2P DAC Mem0 BAR Force REQ64	0x1
14:12	DM1Sw	P2P DAC Mem1 BAR Swap Control	0x1
15	DM1Req64	P2P DAC Mem1 BAR Force REQ64	0x1
18:16	IOSw	P2P I/O BAR Swap Control	0x1
19	Reserved	Reserved.	0x1
22:20	CfgSw	P2P Configuration Swap Control	0x1
31:19	Reserved	Reserved.	0x0

Table 247: PCI Access Control Base 0 (Low)

- PCI_0 Offset: 0x1e00
- PCI_1 Offset: 0x1e80

Bits	Field Name	Function	Initial Value
11:0	Addr	Base Address Corresponds to address bits[31:20].	0xfff
12	PrefetchEn	Read Prefetch Enable 0 - Prefetch disabled. The PCI slave reads a single word. 1 - Prefetch enabled.	0x1
13	DReadEn	Delayed Reads Enable 0 - Disable 1 - Enable	0x0
14	Reserved	Must be 0	0x0
15	Reserved	Reserved.	0x0

Table 247: PCI Access Control Base 0 (Low) (Continued)

- PCI_0 Offset: 0x1e00
- PCI_1 Offset: 0x1e80

Bits	Field Name	Function	Initial Value
16	RdPrefetch	PCI Read Aggressive Prefetch Enable 0 - Disable 1 - Enable The PCI slave prefetches two bursts in advance	0x0
17	RdLinePrefetch	PCI Read Line Aggressive Prefetch Enable 0 - Disable 1 - Enable PCI slave prefetch two bursts in advance.	0x0
18	RdMulPrefetch	PCI Read Multiple Aggressive Prefetch Enable 0 - Disable 1 - Enable PCI slave prefetch two bursts in advance.	0x0
19	Reserved	Reserved.	0x0
21:20	MBurst	PCI Max Burst Specifies the maximum burst size for a single transaction between a PCI slave and the other interfaces 00 - 4 64-bit words 01 - 8 64-bit words 10 - 16 64-bit words 11 - Reserved	0x0
23:22	Reserved	Reserved.	0x0
25:24	PCISwap	Data Swap Control 00 - Byte Swap 01 - No swapping 10 - Both byte and word swap 11 - Word swap	0x1
26	Reserved	Must be 0.	0x0
27	Reserved	Reserved.	0x0
28	AccProt	Access Protect 0 - PCI access is allowed. 1 - Region is not accessible from PCI.	0x0
29	WrProt	Write Protect 0 - PCI write is allowed. 1 - Region is not writeable from PCI	0x0
31:30	Reserved	Reserved.	0x0

Table 248: PCI Access Control Base 0 (High)

- PCI_0 Offset: 0x1e04
- PCI_1 Offset: 0x1e84

Bits	Field Name	Function	Initial Value
31:0	Addr	Base Address High Corresponds to address bits[63:32].	0x0

Table 249: PCI Access Control Top 0

- PCI_0 Offset: 0x1e08
- PCI_1 Offset: 0x1e88

Bits	Field Name	Function	Initial Value
11:0	Addr	Top Address Corresponds to address bits[31:20].	0x0
31:12	Reserved	Reserved.	0x0

Table 250: PCI Access Control Base 1 (Low)

- PCI_0 Offset: 0x1e10
- PCI_1 Offset: 0x1e90

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 247 on page 217	0x1001fff

Table 251: PCI Access Control Base 1 (High)

- PCI_0 Offset: 0x1e14
- PCI_1 Offset: 0x1e94

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 248 on page 219.	0x0

Table 252: PCI Access Control Top 1

- PCI_0 Offset: 0x1e18
- PCI_1 Offset: 0x1e98

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base Top 0. See Table 249 on page 219.	0x0

Table 253: PCI Access Control Base 2 (Low)

- PCI_0 Offset: 0x1e20
- PCI_1 Offset: 0x1ea0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 247 on page 217.	0x1001fff

Table 254: PCI Access Control Base 2 (High)

- PCI_0 Offset: 0x1e24
- PCI_1 Offset: 0x1ea4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 248 on page 219.	0x0

Table 255: PCI Access Control Top 2

- PCI_0 Offset: 0x1e28
- PCI_1 Offset: 0x1ea8

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0. See Table 249 on page 219.	0x0

Table 256: PCI Access Control Base 3 (Low)

- PCI_0 Offset: 0x1e30
- PCI_1 Offset: 0x1eb0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 247 on page 217.	0x1001fff

Table 257: PCI Access Control Base 3 (High)

- PCI_0 Offset: 0x1e34
- PCI_1 Offset: 0x1eb4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 248 on page 219.	0x0

Table 258: PCI Access Control Top 3

- PCI_0 Offset: 0x1e38
- PCI_1 Offset: 0x1eb8

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0. See Table 249 on page 219.	0x0

Table 259: PCI Access Control Base 4 (Low)

- PCI_0 Offset: 0x1e40
- PCI_1 Offset: 0x1ec0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 247 on page 217.	0x1001fff

Table 260: PCI Access Control Base 4 (High)

- PCI_0 Offset: 0x1e44
- PCI_1 Offset: 0x1ec4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 248 on page 219.	0x0

Table 261: PCI Access Control Top 4

- PCI_0 Offset: 0x1e48
- PCI_1 Offset: 0x1ec8

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0. See Table 249 on page 219.	0x0

Table 262: PCI Access Control Base 5 (Low)

- PCI_0 Offset: 0x1e50
- PCI_1 Offset: 0x1ed0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 247 on page 217.	0x1001fff

Table 263: PCI Access Control Base 5 (High)

- PCI_0 Offset: 0x1e54
- PCI_1 Offset: 0x1ed0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 248 on page 219.	0x0

Table 264: PCI Access Control Top 5

- PCI_0 Offset: 0x1e58
- PCI_1 Offset: 0x1ed8

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0. See Table 249 on page 219.	0x0

Table 265: PCI Access Control Base 6 (Low)

- PCI_0 Offset: 0x1e60
- PCI_1 Offset: 0x1ee0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 247 on page 217.	0x1001fff

Table 266: PCI Access Control Base 6 (High)

- PCI_0 Offset: 0x1e64
- PCI_1 Offset: 0x1ee4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 248 on page 219.	0x0

Table 267: PCI Access Control Top 6

- PCI_0 Offset: 0x1e68
- PCI_1 Offset: 0x1ee8

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0. See Table 249 on page 219.	0x0

Table 268: PCI Access Control Base 7 (Low)

- PCI_0 Offset: 0x1e70
- PCI_1 Offset: 0x1ef0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 247 on page 217.	0x1001fff

Table 269: PCI Access Control Base 7 (High)

- PCI_0 Offset: 0x1e74
- PCI_1 Offset: 0x1ef4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 248 on page 219.	0x0

Table 270: PCI Access Control Top 7

- PCI_0 Offset: 0x1e78
- PCI_1 Offset: 0x1ef8

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0. See Table 249 on page 219.	0x0

8.19.3 PCI Configuration Access Registers

Table 271: PCI Configuration Address

- PCI_0 Offset: 0xcf8
- PCI_1 Offset: 0xc78

Bits	Field Name	Function	Initial Value
1:0	Reserved	Read Only.	0x0
7:2	RegNum	Register number.	0x00
10:8	FunctNum	Function number.	0x0
15:11	DevNum	Device number.	0x00
23:16	BusNum	Bus number.	0x00
30:24	Reserved	Read Only.	0x0
31	ConfigEn	When set, an access to the Configuration Data register is translated into a Configuration or Special cycle on the PCI bus.	0x0

Table 272: PCI Configuration Data

- PCI_0 Offset: 0xcfc
- PCI_1 Offset: 0xc7c

Bits	Field Name	Function	Initial Value
31:0	ConfigData	The data is transferred to/from the PCI bus when the CPU accesses this register and the ConfigEn bit in the Configuration Address register is set. A CPU access to this register causes the GT–64240A to perform a Configuration or Special cycle on the PCI bus.	0x000

Table 273: PCI Interrupt Acknowledge

- PCI_0 Offset: 0xc34
- PCI_1 Offset: 0xcb4

Bits	Field Name	Function	Initial Value
31:0	IntAck	A CPU read access to this register forces an interrupt acknowledge cycle on the PCI bus. This register is READ ONLY.	0x0

8.19.4 PCI Error Report Registers

Table 274: PCI SERR* Mask

- PCI_0 Offset: 0xc28
- PCI_1 Offset: 0xca8

NOTE: The GT–64240A asserts SERR* only if SERR* is enabled via the PCI Status and Command register, see Table 291 on page 234.

Bits	Field Name	Function	Initial Value
0	SAPerr	If set to ‘1’, asserts SERR* upon PCI slave detection of bad address parity.	0x0
1	SWrPerr	If set to ‘1’, asserts SERR* upon PCI slave detection of bad write data parity.	0x0
2	SRdPerr	If set to ‘1’, asserts SERR* upon a PERR* response to read data driven by the PCI slave.	0x0
3	Reserved	Reserved.	0x0
4	MAPerr	If set to ‘1’, asserts SERR* upon a PERR* response to an address driven by the PCI master.	0x0
5	MWrPerr	If set to ‘1’, asserts SERR* upon a PERR* response to write data driven by the PCI master.	0x0

Table 274: PCI SERR* Mask (Continued)

- PCI_0 Offset: 0xc28
- PCI_1 Offset: 0xca8

NOTE: The GT–64240A asserts SERR* only if SERR* is enabled via the PCI Status and Command register, see Table 291 on page 234.

Bits	Field Name	Function	Initial Value
6	MRdPerr	If set to '1', asserts SERR* upon bad data parity detection during a PCI master read transaction.	0x0
7	Reserved	Reserved.	0x0
8	MMabort	If set to '1', asserts SERR* upon a PCI master generation of master abort.	0x0
9	MTabort	If set to '1', asserts SERR* upon a PCI master detection of target abort.	0x0
10	Reserved	Reserved.	0x0
11	MRetry	If set to '1', asserts SERR* upon a PCI master reaching retry counter limit.	0x0
15:12	Reserved	Reserved.	0x0
16	SMabort	If set to '1', asserts SERR* upon a PCI slave detection of master abort.	0x0
17	STabort	If set to '1', asserts SERR* upon a PCI slave termination of a transaction with Target Abort.	0x0
18	SAccProt	If set to '1', asserts SERR* upon a PCI slave access protect violation.	0x0
19	SWrProt	If set to '1', asserts SERR* upon a PCI slave write protect violation.	0x0
20	SRdBuf	If set to '1', asserts SERR* if the PCI slave's read buffer, discard timer expires	0x0
21	Arb	If set to '1', asserts SERR* upon the internal PCI arbiter detection of a "broken" PCI master.	0x0
31:22	Reserved	Reserved.	0x0

Table 275: PCI Error Address (Low)

- PCI_0 Offset: 0x1d40
- PCI_1 Offset: 0x1dc0

Bits	Field Name	Function	Initial Value
31:0	ErrAddr	PCI address bits [31:0] are latched upon an error condition. Upon address latch, no new addresses can be registered (due to additional error condition) until the register is being read. Read Only.	0x0

Table 276: PCI Error Address (High)

- PCI_0 Offset: 0x1d44
- PCI_1 Offset: 0x1dc4

NOTE: Upon data sample, no new data is latched until the PCI Error Low Address register is read. This means that PCI Error Low Address register must be the last register read by the interrupt handler.

Bits	Field Name	Function	Initial Value
31:0	ErrAddr	PCI address bits [63:32] are latched upon error condition. Applicable only when running DAC cycles.	0x0

Table 277: PCI Error Data (Low)

- PCI_0 Offset: 0x1d48
- PCI_1 Offset: 0x1dc8

Bits	Field Name	Function	Initial Value
31:0	ErrData	PCI data bits [31:0] are latched upon error condition.	0x0

Table 278: PCI Error Data (High)

- PCI_0 Offset: 0x1d4c
- PCI_1 Offset: 0x1dcc

Bits	Field Name	Function	Initial Value
31:0	ErrData	PCI data bits [63:32] are latched upon error condition. Applicable only when running 64-bit cycles.	0x0

Table 279: PCI Error Command

- PCI_0 Offset: 0x1d50
- PCI_1 Offset: 0x1dd0

NOTE: Upon data sample, no new data is latched until the PCI Error Low Address register is read. This means that PCI Error Low Address register must be the last register read by the interrupt handler.

Bits	Field Name	Function	Initial Value
3:0	ErrCmd	PCI command is latched upon error condition.	0x0
7:4	Reserved	Reserved.	0x0
15:8	ErrBE	PCI byte enable is latched upon error condition.	0x0
16	ErrPAR	PCI PAR is latched upon error condition.	0x0
17	ErrPAR64	PCI PAR64 is latched upon error condition. Applicable only when running 64-bit cycles.	0x0
31:18	Reserved	Reserved.	0x0

Table 280: PCI Interrupt Cause [1,2]

- PCI_0 Offset: 0x1d58
- PCI_1 Offset: 0x1dd8

Bits	Field Name	Function	Initial Value
0	SAPerr	The PCI slave detected bad address parity.	0x0
1	SWrPerr	The PCI slave detected bad write data parity.	0x0
2	SRdPerr	PERR* response to read data driven by PCI slave.	0x0
3	Reserved	Reserved.	0x0
4	MAPerr	PERR* response to address driven by the PCI master.	0x0
5	MWrPerr	PERR* response to write data driven by the PCI master.	0x0
6	MRdPerr	Bad data parity detected during the PCI master read transaction.	0x0
7	Reserved	Reserved.	0x0
8	MMabort	The PCI master generated master abort.	0x0
9	MTabort	The PCI master detected target abort.	0x0
10	MMasterEn	An attempt to generate a PCI transaction while master is not enabled.	0x0
11	MRetry	The PCI master reached retry counter limit.	0x0
15:12	Reserved	Reserved.	0x0
16	SMabort	The PCI slave detects an illegal master termination.	0x0

Table 280: PCI Interrupt Cause (Continued)[1,2]

- PCI_0 Offset: 0x1d58
- PCI_1 Offset: 0x1dd8

Bits	Field Name	Function	Initial Value
17	STabort	The PCI slave terminates a transaction with Target Abort.	0x0
18	SAccProt	A PCI slave access protect violation.	0x0
19	SWrProt	A PCI slave write protect violation.	0x0
20	SRdBuf	A PCI slave read buffer discard timer expired.	0x0
21	Arb	Internal PCI arbiter detection of a “broken” master.	0x0
23:22	Reserved	Reserved.	0x0
24	BIST	PCI BIST Interrupt	0x0
25	PMG	PCI Power Management Interrupt	0x0
26	PRST	PCI Reset Assert	0x0

Table 280: PCI Interrupt Cause (Continued)[1,2]

- PCI_0 Offset: 0x1d58
- PCI_1 Offset: 0x1dd8

Bits	Field Name	Function	Initial Value
31:27	Sel	Specifies the error event currently being reported in the Error Address, Error Data, and Error Command registers. 0x0 - SAPerr 0x1 - SWrPerr 0x2 - SRdPerr 0x3 - Reserved 0x4 - MAPerr 0x5 - MWrPerr 0x6 - MRdPerr 0x7 - Reserved 0x8 - MMabort 0x9 - MTabort 0xa - MMasterEn 0xb - MRetry 0xc - 0xf - Reserved 0x10 - SMabort 0x11 - STabort 0x12 - SAccProt 0x13 - SWrProt 0x14 - SRdBuf 0x15 - Arb 0x16 - 0x17 - Reserved 0x18 - BIST 0x19 - PMG 0x1a - PRST 0x1b - 0x1f - Reserved Read Only	

1. All bits are Clear Only. A cause bit set upon error event occurrence. A write of 0 clears the bit. A write of 1 has no affect.
2. PCI Interrupt bits are organized in four groups: bits[7:0] for address and data parity errors, bits[15:8] for PCI master transaction failure (possible external target problem), bits[23:16] for slave response failure (possible external master problem), and bit[26:24] for external PCI events that require CPU handle.

Table 281: PCI Error Mask

- PCI_0 Offset: 0x1d5c
- PCI_1 Offset: 0x1ddc

Bits	Field Name	Function	Initial Value
0	SAPerr	If set to ‘1’, SAPerr interrupt is enabled.	0x0
1	SWrPerr	If set to ‘1’, SWrPerr interrupt is enabled.	0x0
2	SRdPerr	If set to ‘1’, SRdPerr interrupt is enabled.	0x0
3	Reserved	Reserved.	0x0
4	MAPerr	If set to ‘1’, MAPerr interrupt is enabled.	0x0
5	MWrPerr	If set to ‘1’, MWrPerr interrupt is enabled.	0x0
6	MRdPerr	If set to ‘1’, MRdPerr interrupt is enabled.	0x0
7	Reserved	Reserved	0x0
8	MMabort	If set to ‘1’, MMabort interrupt is enabled.	0x0
9	MTabort	If set to ‘1’, MTabort interrupt is enabled.	0x0
10	MMasterEn	If set to ‘1’, MMasterEn interrupt is enabled.	0x0
11	MRetry	If set to ‘1’, MRetry interrupt is enabled.	0x0
15:12	Reserved	Reserved.	0x0
16	SMabort	If set to ‘1’, SMabort interrupt is enabled.	0x0
17	STabort	If set to ‘1’, STabort interrupt is enabled.	0x0
18	SAccProt	If set to ‘1’, SAccProt interrupt is enabled.	0x0
19	SWrProt	If set to ‘1’, SWrProt interrupt is enabled.	0x0
20	SRdBuf	If set to ‘1’, SRdBuf interrupt is enabled.	0x0
21	Arb	If set to ‘1’, Arb interrupt is enabled.	0x0
23:22	Reserved	Reserved.	0x0
24	BIST	If set to ‘1’, BIST interrupt is enabled.	0x0
25	PMG	If set to ‘1’, PMG interrupt is enabled.	0x0
26	PRST	If set to ‘1’, PRST interrupt is enabled.	0x0
31:27	Reserved	Reserved.	0x0

8.19.5 PCI Slave Debug Registers

NOTE: Reserved for Galileo Technology usage.

Table 282: X0 Address

- PCI Offset: 0x1d20

Bits	Field Name	Function	Initial Value
31:0	Addr	I2x0_ad[31:0] registered on (I2x0_req & x02I_ack)	0x0

Table 283: X0 Command and ID

- PCI Offset: 0x1d24

Bits	Field Name	Function	Initial Value
19:0	Cmd	I2x0_cbe[19:0] registered on (I2x0_req & x02I_ack)	0x0
31:20	ID	I2x0_id[11:0] registered on (I2x0_req & x02I_ack)	0x0

Table 284: Write Data (Low)

- PCI Offset: 0x1d30

Bits	Field Name	Function	Initial Value
31:0	Data	I2x0_ad[31:0] registered on I2x0_valid	0x0

Table 285: Write Data (High)

- PCI Offset: 0x1d34

Bits	Field Name	Function	Initial Value
31:0	Data	I2x0_ad[63:32] registered on I2x0_valid	0x0

Table 286: Write Byte Enables

- PCI Offset: 0x1d60

Bits	Field Name	Function	Initial Value
7:0	BE	I2x0_cbe registered on I2x0_valid	0x0
31:8	Reserved	Reserved.	0x0

Table 287: Read Data (Low), Offset: 0x1d38

- PCI Offset: 0x1d38

Bits	Field Name	Function	Initial Value
31:0	Data	x02l_ad[31:0] registered on x02l_rd_valid	0x0

Table 288: Read Data (High), Offset: 0x1d3c

- PCI Offset: 0x1d3c

Bits	Field Name	Function	Initial Value
31:0	Data	x02l_ad[63:32] registered on x02l_rd_valid	0x0

Table 289: Read ID, Offset: 0x1d64

- PCI Offset: 0x1d64

Bits	Field Name	Function	Initial Value
11:0	ID	x02l_id[11:0] registered on x02l_rd_valid	0x0
31:12	Reserved	Reserved.	0x0

8.19.6 Function 0 Configuration Registers

Table 290: PCI Device and Vendor ID

- PCI_0 Offset from CPU or PCI_0: 0x00
- PCI_0 Offset from PCI_1: 0x80
- PCI_1 Offset from CPU or PCI_0: 0x80
- PCI_1 Offset from PCI_1: 0x00

Bits	Field Name	Function	Initial Value
15:0	VenID	Galileo's Vendor ID. Read only from PCI.	0x11ab
31:16	DevID	GT–64240A Device ID. Read only from PCI.	0x6430

Table 291: PCI Status and Command

- PCI_0 Offset from CPU or PCI_0: 0x04
- PCI_0 Offset from PCI_1: 0x84
- PCI_1Offset from CPU or PCI_0: 0x84
- PCI_1 Offset from PCI_1: 0x04

Bits	Field Name	Function	Initial Value
0	IOEn	Controls the GT–64240A's ability to response to PCI I/O accesses. 0 - Disable 1 - Enable	0x0
1	MEMEn	Controls the GT–64240A's ability to response to PCI Memory accesses. 0 - Disable 1 - Enable	0x0
2	MasEn	Controls the GT–64240A's ability to act as a master on the PCI bus. 0 - Disable 1 - Enable	0x0
3	SpecialEn	Controls the GT–64240A's ability to respond to PCI special cycles. Read only 0 (GT–64240A PCI slave does not support special cycles).	0x0
4	MemWrInv	Controls the GT–64240A's ability to generate memory write and invalidate commands on the PCI bus. 0 - Disable 1 - Enable	0x0
5	VGA	VGA Palette Snoops Not supported. Read only 0.	0x0
6	PErrEn	Controls the GT–64240A's ability to respond to parity errors on the PCI by asserting the PErr* pin. 0 - Disable 1 - Enable	0x0
7	AddrStep	Address Stepping Enable The GT–64240A PCI master performs address stepping only on configuration accesses. Read only from the PCI.	0x0
8	SErrEn	Controls the GT–64240A's ability to assert the SErr* pin. 0 - Disable 1 - Enable	0x0

Table 291: PCI Status and Command (Continued)

- PCI_0 Offset from CPU or PCI_0: 0x04
- PCI_0 Offset from PCI_1: 0x84
- PCI_1Offset from CPU or PCI_0: 0x84
- PCI_1 Offset from PCI_1: 0x04

Bits	Field Name	Function	Initial Value
9	FastBTBEn	Controls the GT–64240A's ability to generate fast back-to-back transactions. 0 - Disable 1 - Enable	0x0
19:10	Reserved	Read only.	0x0
20	CapList	Capability List Support Indicates that the GT–64240A configuration header includes capability list. Read only from the PCI.	0x1
21	66MHzEn	66MHz Capable The GT–64240A PCI interface is capable of running at 66MHz regardless of this bit value. Read only from PCI.	0x1
22	Reserved	Read only.	0x0
23	TarFastBB	Indicates that the GT–64240A is capable of accepting fast back-to-back transactions on the PCI bus. Read only from the PCI.	0x1
24	DataPerr	Set by the GT–64240A when it detects a parity error (detects or asserts PERR*) as a master and the PErrEn bit is set. Clear only by writing '1'.	0x0
26:25	DevSelTim	Indicates the GT–64240A's DevSel timing (medium). Read only from the PCI.	0x1
27	SlaveTabort	Set when the GT–64240A's slave terminates a transaction with Target Abort. Clear only by writing 1.	0x0
28	MasterTabort	Set when the GT–64240A's master detects a Target Abort termination. Clear only by writing 1.	0x0
29	MAbort	Set when the GT–64240A's master generates a Master Abort (except of special cycle). Clear only by writing 1.	0x0
30	SysErr	Set when the GT–64240A asserts SERR*. Clear only by writing 1.	0x0

Table 291: PCI Status and Command (Continued)

- PCI_0 Offset from CPU or PCI_0: 0x04
- PCI_0 Offset from PCI_1: 0x84
- PCI_1Offset from CPU or PCI_0: 0x84
- PCI_1 Offset from PCI_1: 0x04

Bits	Field Name	Function	Initial Value
31	DetParErr	Set upon the GT–64240A detection of Parity error (both as master and slave). Clear only by writing 1.	0x0

Table 292: PCI Class Code and Revision ID

- PCI_0 Offset from CPU or PCI_0: 0x08
- PCI_0 Offset from PCI_1: 0x88
- PCI_1 Offset from CPU or PCI_0: 0x08
- PCI_1 Offset from PCI_1: 0x88

Bits	Field Name	Function	Initial Value
7:0	RevID	Indicates the GT–64240A Revision number. Read only from PCI.	0x10
15:8	Reserved	Read only.	0x0
23:16	SubClass	Indicates the GT–64240A Subclass. Read only from PCI.	0x80
31:24	BaseClass	Indicates the GT–64240A Base Class. Read only from PCI.	0x05

Table 293: PCI BIST, Header Type, Latency Timer, and Cache Line

- PCI_0 Offset from CPU or PCI_0: 0x0c
- PCI_0 Offset from PCI_1: 0x8c
- PCI_1 Offset from CPU or PCI_0: 0x8c
- PCI_1 Offset from PCI_1: 0x0c

Bits	Field Name	Function	Initial Value
7:0	CacheLine	Specifies the GT–64240A's cache line size.	0x00
15:8	LatTimer	Specifies in units of PCI bus clocks the latency timer value of the GT–64240A.	0x00
23:16	HeadType	Specifies Configuration Header Type Read only from PCI.	0x80

Table 293: PCI BIST, Header Type, Latency Timer, and Cache Line (Continued)

- PCI_0 Offset from CPU or PCI_0: 0x0c
- PCI_0 Offset from PCI_1: 0x8c
- PCI_1 Offset from CPU or PCI_0: 0x8c
- PCI_1 Offset from PCI_1: 0x0c

Bits	Field Name	Function	Initial Value
27:24	BISTComp	BIST Completion Code Written by the CPU upon BIST completion. Read only from PCI.	0x0
29:28	Reserved	Reserved.	0x0
30	BISTAct	BIST Activate bit Set to '1' by PCI to activate BIST. Cleared by CPU upon BIST completion.	0x0
31	BISTCap	BIST Capable Bit Read Only from PCI.	0x1

Table 294: PCI SCS[0]* Base Address

- PCI_0 Offset from CPU or PCI_0: 0x10
- PCI_0 Offset from PCI_1: 0x90
- PCI_1 Offset from CPU or PCI_0: 0x90
- PCI_1 Offset from PCI_1: 0x10

Bits	Field Name	Function	Initial Value
0	MemSpace	Memory Space Indicator Read only from PCI.	0x0
2:1	Type	BAR Type Read only from PCI.	0x0
3	Prefetch	Prefetch Enable Read only from PCI.	0x1
11:4	Reserved	Read only.	0x0
31:12	Base	Base address.	0x00000

Table 295: PCI SCS[1]* Base Address

- PCI_0 Offset from CPU or PCI_0: 0x14
- PCI_0 Offset from PCI_1: 0x94
- PCI_1 Offset from CPU or PCI_0: 0x94
- PCI_1 Offset from PCI_1: 0x10

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 294 on page 237.	0x00800008

Table 296: PCI SCS[2]* Base Address

- PCI_0 Offset from CPU or PCI_0: 0x18
- PCI_0 Offset from PCI_1: 0x98
- PCI_1 Offset from CPU or PCI_0: 0x98
- PCI_1 Offset from PCI_1: 0x18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 294 on page 237.	0x01000008

Table 297: PCI SCS[3]* Base Address

- PCI_0 Offset from CPU or PCI_0: 0x1c
- PCI_0 Offset from PCI_1: 0x9c
- PCI_1 Offset from CPU or PCI_0: 0x9c
- PCI_1 Offset from PCI_1: 0x1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 294 on page 237.	0x01800008

Table 298: PCI Internal Registers Memory Mapped Base Address

- PCI_0 Offset from CPU or PCI_0: 0x20
- PCI_0 Offset from PCI_1: 0xa0
- PCI_1 Offset from CPU or PCI_0: 0xa0
- PCI_1 Offset from PCI_1: 0x20

Bits	Field Name	Function	Initial Value
0	MemSpace	Memory Space Indicator Read only from PCI.	0x0

Table 298: PCI Internal Registers Memory Mapped Base Address (Continued)

- PCI_0 Offset from CPU or PCI_0: 0x20
- PCI_0 Offset from PCI_1: 0xa0
- PCI_1 Offset from CPU or PCI_0: 0xa0
- PCI_1 Offset from PCI_1: 0x20

Bits	Field Name	Function	Initial Value
2:1	Type	BAR Type Read only from PCI.	0x0
3	Prefetch	Prefetch Enable Read only from PCI.	0x0
15:4	Reserved	Read only.	0x0
31:16	Base	Base Address	0x1400

Table 299: PCI Internal Registers I/O Mapped Base Address

- PCI_0 Offset from CPU or PCI_0: 0x24
- PCI_0 Offset from PCI_1: 0xa4
- PCI_1 Offset from CPU or PCI_0: 0xa4
- PCI_1 Offset from PCI_1: 0x24

Bits	Field Name	Function	Initial Value
0	IOSpace	I/O Space Indicator Read only from PCI.	0x1
2:1	Type	BAR Type Read only from PCI.	0x0
3	Prefetch	Prefetch Enable Read only from PCI.	0x0
15:4	Reserved	Read only.	0x0
31:16	Base	Base Address	0x1400

Table 300: PCI Subsystem Device and Vendor ID

- PCI_0 Offset from CPU or PCI_0: 0x2c
- PCI_0 Offset from PCI_1: 0xac
- PCI_1 Offset from CPU or PCI_0: 0xac
- PCI_1 Offset from PCI_1: 0x2c

Bits	Field Name	Function	Initial Value
15:0	VenID	Subsystem Manufacturer ID Number	0x0

Table 300: PCI Subsystem Device and Vendor ID

- PCI_0 Offset from CPU or PCI_0: 0x2c
- PCI_0 Offset from PCI_1: 0xac
- PCI_1 Offset from CPU or PCI_0: 0xac
- PCI_1 Offset from PCI_1: 0x2c

Bits	Field Name	Function	Initial Value
31:16	DevID	Subsystem Device ID Number	0x0

Table 301: PCI Expansion ROM Base Address Register

- PCI_0 Offset from CPU or PCI_0: 0x30
- PCI_0 Offset from PCI_1: 0xb0
- PCI_1 Offset from CPU or PCI_0: 0xb0
- PCI_1 Offset from PCI_1: 0x30

Bits	Field Name	Function	Initial Value
0	ExpROMEn	Expansion ROM Enable 0 - Disable 1 - Enable	0x0
11:1	Reserved	Reserved.	0x0
31:12	ExpROMBase	Expansion ROM Base Address	0x1f000

Table 302: PCI Capability List Pointer Register

- PCI_0 Offset from CPU or PCI_0: 0x34
- PCI_0 Offset from PCI_1: 0xb4
- PCI_1 Offset from CPU or PCI_0: 0xb4
- PCI_1 Offset from PCI_1: 0x34

Bits	Field Name	Function	Initial Value
7:0	CapPtr	Capability List Pointer Read only.	0x40
31:8	Reserved	Reserved.	0x0

Table 303: PCI Interrupt Pin and Line

- PCI_0 Offset from CPU or PCI_0: 0x3c
- PCI_0 Offset from PCI_1: 0xbc
- PCI_1 Offset from CPU or PCI_0: 0xbc
- PCI_1 Offset from PCI_1: 0x3c

Bits	Field Name	Function	Initial Value
7:0	IntLine	Provides interrupt line routing information.	0x0
15:8	IntPin	Indicates which interrupt pin is used by the GT–64240A. Read only from PCI.	0x1
31:16	Reserved	Read only.	0x0

Table 304: PCI Power Management Capability

- PCI_0 Offset from CPU or PCI_0: 0x40
- PCI_0 Offset from PCI_1: 0xc0
- PCI_1 Offset from CPU or PCI_0: 0xc0
- PCI_1 Offset from PCI_1: 0x40

Bits	Field Name	Function	Initial Value
7:0	CapID	Capability ID Read only from PCI.	0x1
15:8	NextPtr	Next Item Pointer Read only from PCI.	0x48
18:16	Ver	PCI Power Management Spec Revision Read only from PCI.	0x1
19	PMEClk	PME Clock Indicates that the PCI clock is required for the GT–64240A to assert PME* Read only from PCI.	0x1
20	Reserved	Read only from PCI.	0x0
21	DSI	Device Specific Initialization Read only from PCI.	0x0
24:22	AuxCur	Auxiliary Current Requirements Read only from PCI.	0x0
25	D1Sup	D1 Power Management State Support Read only from PCI. 0 - Not supported 1 - Supported	0x1

Table 304: PCI Power Management Capability (Continued)

- PCI_0 Offset from CPU or PCI_0: 0x40
- PCI_0 Offset from PCI_1: 0xc0
- PCI_1 Offset from CPU or PCI_0: 0xc0
- PCI_1 Offset from PCI_1: 0x40

Bits	Field Name	Function	Initial Value
26	D2Sup	D2 Power Management State Support Read only from PCI. 0 - Not supported 1 - Supported	0x1
31:27	PMESup	PME* Signal Support Indicates in which power states the GT–64240A supports the PME* pin. Each bit corresponds to different state (bit[0] - D0, bit[1] - D1, bit[2] - D2, bit[3] - D3-hot, bit[4] - D3-cold). For example, 'b01001 stands for supporting PME* only on D0 and D3-hot states. Read only from PCI.	0x0f

Table 305: PCI Power Management Control and Status Register

- PCI_0 Offset from CPU or PCI_0: 0x44
- PCI_0 Offset from PCI_1: 0xc4
- PCI_1 Offset from CPU or PCI_0: 0xc4
- PCI_1 Offset from PCI_1: 0x44

Bits	Field Name	Function	Initial Value
1:0	PState	Power State 00 - D0 01 - D1 10 - D2 11 - D3-hot	0x0
7:2	Reserved	Read only from PCI.	0x0
8	PME_EN	PME* Pin Assertion Enable	0x0
12:9	DSel	Data Select	0x0
14:13	DScale	Data Scale Read only from PCI.	0x0
15	PME_Stat	PME* Pin Status CPU set only by writing '1'. PCI clear only by writing '1'. When set to '1', the GT–64240A asserts PME* pin.	0x0

Table 305: PCI Power Management Control and Status Register (Continued)

- PCI_0 Offset from CPU or PCI_0: 0x44
- PCI_0 Offset from PCI_1: 0xc4
- PCI_1 Offset from CPU or PCI_0: 0xc4
- PCI_1 Offset from PCI_1: 0x44

Bits	Field Name	Function	Initial Value
23:16	P2P	Power Management Status and Control for P2P Bridge Read only from PCI.	0x0
31:24	Data	State Data Read only from PCI.	0x0

Table 306: PCI VPD Address

- PCI_0 Offset from CPU or PCI_0: 0x48
- PCI_0 Offset from PCI_1: 0xc8
- PCI_1 Offset from CPU or PCI_0: 0xc8
- PCI_1 Offset from PCI_1: 0x48

Bits	Field Name	Function	Initial Value
7:0	CapID	Capability ID Read only from PCI	0x3
15:8	NextPtr	Next Item Pointer Read only from PCI	0x50
30:16	Addr	VPD Address Points to the location of the VPD structure in memory. **NOTE:** The GT–64240A also implements remapping of the high address bits through the PCI Address Decoding Control register.	0x0
31	Flag	Flag Flipped by System or GT–64240A during VPD Access On VPD writes, system sets the flag to ‘1’ indicating VPD write is required. The GT–64240A clears the flag to indicate that the VPD write is done (data from the VPD Data register was written to memory). On VPD reads, the system sets the flag to ‘0’, indicating VPD read is required. The GT–64240A sets the flag to ‘1’ when the read is done (data has been read from memory and put in VPD Data register).	0x0

Table 307: PCI VPD Data

- PCI_0 Offset from CPU or PCI_0: 0x4c
- PCI_0 Offset from PCI_1: 0xcc
- PCI_1 Offset from CPU or PCI_0: 0xcc
- PCI_1 Offset from PCI_1: 0x4c

Bits	Field Name	Function	Initial Value
31:0	Data	VPD Data	0x0

Table 308: PCI MSI Message Control

- PCI_0 Offset from CPU or PCI_0: 0x50
- PCI_0 Offset from PCI_1: 0xd0
- PCI_1 Offset from CPU or PCI_0: 0xd0
- PCI_1 Offset from PCI_1: 0x50

Bits	Field Name	Function	Initial Value
7:0	CapID	Capability ID Read only from PCI.	0x5
15:8	NextPtr	Next Item Pointer Read only from PCI.	0x60
16	MSIEn	MSI Enable 0 - Disable The GT–64240A generates a PCI interrupt. 1 - Enabled The GT–64240A generates MSI messages instead of interrupts.	0x0
19:17	MultiCap	Multiple Messages Capable The GT–64240A is capable of driving a single message. Read only from PCI.	0x0
22:20	MultiEn	Multiple Messages Enable The number of messages the system allocates to the GT–64240A (must be smaller or equal to MultiCap).	0x0
23	Addr64	64-bit Addressing Capable Indicates whether the GT–64240A is capable of generating 64-bit message address. Read only from PCI. 0 - Not capable 1 - Capable	0x1
31:24	Reserved	Read only 0.	0x0

Table 309: PCI MSI Message Address

- PCI_0 Offset from CPU or PCI_0: 0x54
- PCI_0 Offset from PCI_1: 0xd4
- PCI_1 Offset from CPU or PCI_0: 0xd4
- PCI_1 Offset from PCI_1: 0x54

Bits	Field Name	Function	Initial Value
31:0	Addr	Message Address	0x0

Table 310: PCI MSI Message Upper Address

- PCI_0 Offset from CPU or PCI_0: 0x58
- PCI_0 Offset from PCI_1: 0xd8
- PCI_1 Offset from CPU or PCI_0: 0xd8
- PCI_1 Offset from PCI_1: 0x58

Bits	Field Name	Function	Initial Value
31:0	Addr	Message Upper Address 32 upper address bits. If set to a value other than '0', the GT–64240A issues MSI message as DAC cycle.	0x0

Table 311: PCI MSI Data Control

- PCI_0 Offset from CPU or PCI_0: 0x5c
- PCI_0 Offset from PCI_1: 0xdc
- PCI_1 Offset from CPU or PCI_0: 0xdc
- PCI_1 Offset from PCI_1: 0x5c

Bits	Field Name	Function	Initial Value
15:0	Data	Message Data	0x0
31:16	Reserved	Read only 0.	0x0

Table 312: PCI CompactPCI HotSwap Capability

- PCI_0 Offset from CPU or PCI_0: 0x60
- PCI_0 Offset from PCI_1: 0xe0
- PCI_1 Offset from CPU or PCI_0: 0xe0
- PCI_1 Offset from PCI_1: 0x60

NOTE: CompactPCI Hot Swap is only supported on the PCI_0 interface.

Bits	Field Name	Function	Initial Value
7:0	CapID	Capability ID Read only from PCI.	0x6
15:8	NextPtr	Next Item Pointer Read only from PCI.	0x0
16	Reserved	Read only 0.	0x0
17	EIM	ENUM* Interrupt Mask 0 - Enable signal 1 - Mask signal	0x0
18	Reserved	Read only 0.	0x0
19	LOO	LED On/Off 0 - LED off 1 - LED on	0x0
21:20	Reserved	Read only 0.	0x0
22	Ext	Extraction Indicates that the board is about to be extracted (set to 1).	0x0
23	Ins	Insertion Indicates that the board has just been inserted (set to 1).	0x0
31:24	Reserved	Read only 0.	0x0

8.19.7 Function 1 Configuration Registers

Table 313: PCI CS[0]* Base Address

- PCI_0 Offset from CPU or PCI_0: 0x10
- PCI_1 Offset from PCI_1: 0x90
- PCI_0 Offset from CPU or PCI_0: 0x90
- PCI_1 Offset from PCI_1: 0x10

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 294 on page 237.	0x1c000000

Table 314: PCI CS[1]* Base Address

- PCI_0 Offset from CPU or PCI_0: 0x14
- PCI_1 Offset from PCI_1: 0x94
- PCI_0 Offset from CPU or PCI_0: 0x94
- PCI_1 Offset from PCI_1: 0x14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 294 on page 237.	0x1c800000

Table 315: PCI CS[2]* Base Address

- PCI_0 Offset from CPU or PCI_0: 0x18
- PCI_1 Offset from PCI_1: 0x98
- PCI_0 Offset from CPU or PCI_0: 0x98
- PCI_1 Offset from PCI_1: 0x18

Bits	Field Name	Function	Initial Value
31:12	Various	Same as SCS[0]* Base Address. See Table 294 on page 237.	0x1d000000

Table 316: PCI CS[3]* Base Address

- PCI_0 Offset from CPU or PCI_0: 0x1c
- PCI_1 Offset from PCI_1: 0x9c
- PCI_0 Offset from CPU or PCI_0: 0x9c
- PCI_1 Offset from PCI_1: 0x1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 294 on page 237.	0x1f000000

Table 317: PCI Boot CS* Base Address

- PCI_0 Offset from CPU or PCI_0: 0x20
- PCI_1 Offset from PCI_1: 0xa0
- PCI_0 Offset from CPU or PCI_0: 0xa0
- PCI_1 Offset from PCI_1: 0x20

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 294 on page 237.	0x1f800000

8.19.8 Function 2 Configuration Registers

Table 318: PCI P2P Mem0 Base Address

- PCI_0 Offset from CPU or PCI_0: 0x10
- PCI_1 Offset from PCI_1: 0x90
- PCI_0 Offset from CPU or PCI_0: 0x90
- PCI_1 Offset from PCI_1: 0x10

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 294 on page 237.	PCI_0: 0x22000008 PCI_1: 0x12000008

Table 319: PCI P2P Mem1 Base Address

- PCI_0 Offset from CPU or PCI_0: 0x14
- PCI_1 Offset from PCI_1: 0x94
- PCI_0 Offset from CPU or PCI_0: 0x94
- PCI_1 Offset from PCI_1: 0x14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 294 on page 237.	PCI_0: 0x24000008 PCI_1: 0xf2000008

Table 320: PCI P2P I/O Base Address

- PCI_0 Offset from CPU or PCI_0: 0x18
- PCI_1 Offset from PCI_1: 0x98
- PCI_0 Offset from CPU or PCI_0: 0x98
- PCI_1 Offset from PCI_1: 0x18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 294 on page 237.	PCI_0: 0x20000001 PCI_1: 0x10000001

8.19.9 Function 4 Configuration Registers

Table 321: PCI DAC SCS[0]* Base Address (Low)

- PCI_0 Offset from CPU or PCI_0: 0x10
- PCI_1 Offset from PCI_1: 0x90
- PCI_0 Offset from CPU or PCI_0: 0x90
- PCI_1 Offset from PCI_1: 0x10

Bits	Field Name	Function	Initial Value
0	MemSpace	Memory Space Indicator Read only from PCI.	0x0
2:1	Type	BAR Type Read only from PCI.	0x2
3	Prefetch	Prefetch Enable Read only from PCI.	0x1
11:4	Reserved	Read only.	0x0
31:12	BaseLow	Base Low Address	0x0

Table 322: PCI DAC SCS[0]* Base Address (High)

- PCI_0 Offset from CPU or PCI_0: 0x14
- PCI_1 Offset from PCI_1: 0x94
- PCI_0 Offset from CPU or PCI_0: 0x94
- PCI_1 Offset from PCI_1: 0x14

Bits	Field Name	Function	Initial Value
31:0	BaseHigh	Base High Address	0x0

Table 323: PCI DAC SCS[1]* Base Address (Low)

- PCI_0 Offset from CPU or PCI_0: 0x18
- PCI_1 Offset from PCI_1: 0x98
- PCI_0 Offset from CPU or PCI_0: 0x98
- PCI_1 Offset from PCI_1: 0x18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x0080000c

Table 324: PCI DAC SCS[1]* Base Address (High)

- PCI_0 Offset from CPU or PCI_0: 0x1c
- PCI_1 Offset from PCI_1: 0x9c
- PCI_0 Offset from CPU or PCI_0: 0x9c
- PCI_1 Offset from PCI_1: 0x1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address.	0x0

Table 325: PCI DAC P2P Mem0 Base Address (Low)

- PCI_0 Offset from CPU or PCI_0: 0x20
- PCI_1 Offset from PCI_1: 0xa0
- PCI_0 Offset from CPU or PCI_0: 0xa0
- PCI_1 Offset from PCI_1: 0x20

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	PCI_0: 0x2600000c PCI_1: 0xf400000c

Table 326: PCI DAC P2P Mem0 Base Address (High)

- PCI_0 Offset from CPU or PCI_0: 0x24
- PCI_1 Offset from PCI_1: 0xa4
- PCI_0 Offset from CPU or PCI_0: 0xa4
- PCI_1 Offset from PCI_1: 0x24

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x0

8.19.10 Function 5 Configuration Registers

Table 327: PCI DAC SCS[2]* Base Address (Low)

- PCI_0 Offset from CPU or PCI_0: 0x10
- PCI_1 Offset from PCI_1: 0x90
- PCI_0 Offset from CPU or PCI_0: 0x90
- PCI_1 Offset from PCI_1: 0x10

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x0100000c

Table 328: PCI DAC SCS[2]* Base Address (High)

- PCI_0 Offset from CPU or PCI_0: 0x14
- PCI_1 Offset from PCI_1: 0x94
- PCI_0 Offset from CPU or PCI_0: 0x94
- PCI_1 Offset from PCI_1: 0x14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x0

Table 329: PCI DAC SCS[3]* Base Address (Low)

- PCI_0 Offset from CPU or PCI_0: 0x18
- PCI_1 Offset from PCI_1: 0x98
- PCI_0 Offset from CPU or PCI_0: 0x98
- PCI_1 Offset from PCI_1: 0x18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x0180000c

Table 330: PCI DAC SCS[3]* Base Address (High)

- PCI_0 Offset from CPU or PCI_0: 0x1c
- PCI_1 Offset from PCI_1: 0x9c
- PCI_0 Offset from CPU or PCI_0: 0x9c
- PCI_1 Offset from PCI_1: 0x1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x0

Table 331: PCI DAC P2P Mem1 Base Address (Low)

- PCI_0 Offset from CPU or PCI_0: 0x20
- PCI_1 Offset from PCI_1: 0xa0
- PCI_0 Offset from CPU or PCI_0: 0xa0
- PCI_1 Offset from PCI_1: 0x20

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	PCI_0: 0x2800000c PCI_1: 0xf600000c

Table 332: PCI DAC P2P Mem1 Base Address (High)

- PCI_0 Offset from CPU or PCI_0: 0x24
- PCI_1 Offset from PCI_1: 0xa4
- PCI_0 Offset from CPU or PCI_0: 0xa4
- PCI_1 Offset from PCI_1: 0x24

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x0

8.19.11 Function 6 Configuration Registers

Table 333: PCI DAC CS[0]* Base Address (Low)

- PCI_0 Offset from CPU or PCI_0: 0x10
- PCI_1 Offset from PCI_1: 0x90
- PCI_0 Offset from CPU or PCI_0: 0x90
- PCI_1 Offset from PCI_1: 0x10

Bits	Field Name	Function	Initial Value
31:12	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x1c000004

Table 334: PCI DAC CS[0]* Base Address (High)

- PCI_0 Offset from CPU or PCI_0: 0x14
- PCI_1 Offset from PCI_1: 0x94
- PCI_0 Offset from CPU or PCI_0: 0x94
- PCI_1 Offset from PCI_1: 0x14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x0

Table 335: PCI DAC CS[1]* Base Address (Low)

- PCI_0 Offset from CPU or PCI_0: 0x18
- PCI_1 Offset from PCI_1: 0x98
- PCI_0 Offset from CPU or PCI_0: 0x98
- PCI_1 Offset from PCI_1: 0x18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x1c800004

Table 336: PCI DAC CS[1]* Base Address (High)

- PCI_0 Offset from CPU or PCI_0: 0x1c
- PCI_1 Offset from PCI_1: 0x9c
- PCI_0 Offset from CPU or PCI_0: 0x9c
- PCI_1 Offset from PCI_1: 0x1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x0

Table 337: PCI DAC CS[2]* Base Address (Low)

- PCI_0 Offset from CPU or PCI_0: 0x20
- PCI_1 Offset from PCI_1: 0xa0
- PCI_0 Offset from CPU or PCI_0: 0xa0
- PCI_1 Offset from PCI_1: 0x20

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x1d000004

Table 338: PCI DAC CS[2]* Base Address (High)

- PCI_0 Offset from CPU or PCI_0: 0x24
- PCI_1 Offset from PCI_1: 0xa4
- PCI_0 Offset from CPU or PCI_0: 0xa4
- PCI_1 Offset from PCI_1: 0x24

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x0

8.19.12 Function 7 Configuration Registers

Table 339: PCI DAC CS[3]* Base Address (Low)

- PCI_0 Offset from CPU or PCI_0: 0x10
- PCI_1 Offset from PCI_1: 0x90
- PCI_0 Offset from CPU or PCI_0: 0x10
- PCI_1 Offset from PCI_1: 0x90

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x1f000004

Table 340: PCI DAC CS[3]* Base Address (High)

- PCI_0 Offset from CPU or PCI_0: 0x14
- PCI_1 Offset from PCI_1: 0x94
- PCI_0 Offset from CPU or PCI_0: 0x94
- PCI_1 Offset from PCI_1: 0x14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x0

Table 341: PCI DAC BootCS* Base Address (Low)

- PCI_0 Offset from CPU or PCI_0: 0x18
- PCI_1 Offset from PCI_1: 0x98
- PCI_0 Offset from CPU or PCI_0: 0x98
- PCI_1 Offset from PCI_1: 0x18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x1f800004

Table 342: PCI DAC BootCS* Base Address (High)

- PCI_0 Offset from CPU or PCI_0: 0x1c
- PCI_1 Offset from PCI_1: 0x9c
- PCI_0 Offset from CPU or PCI_0: 0x9c
- PCI_1 Offset from PCI_1: 0x1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 218 on page 202.	0x0

9. MESSAGING UNIT

The GT–64240A messaging unit includes hardware hooks for message transfers between PCI devices and the CPU. This includes all of the registers required for implementing the I_2O messaging, as defined in the Intelligent I/O (I_2O) Standard specification. This Messaging Unit is compatible with that found GT-64120 and GT-64130 devices.

The I_2O hardware support found in the GT–64240A also provides designers of non-I_2O embedded systems with important benefits. For example, the circular queue support in the Messaging Unit provides a simple, yet powerful, mechanism for passing queued messages between intelligent agents on a PCI bus. Even the simple message and doorbell registers can improve the efficiency of communication between agents on the PCI.

The I_2O specification defines a standard mechanism for passing messages between a host processor (a Pentium, for example) and intelligent I/O processors (a networking card based on the GT–64240A and a MIPS processor, for example.) This same message passing mechanism may be used to pass messages between peers in a system.

The GT–64240A Messaging Unit is implemented in both PCI interfaces. It allows for messaging between the CPU and PCI and inter-PCI interfaces messaging.

The GT–64240A Messaging Unit registers are accessible from the PCI through the GT–64240A internal space, as any other internal register. Setting the PCI Address Control register's MsgACC bit to '0' enables access to these registers through the lower 4Kbyte of SCS[0] BAR space.

NOTE: If accessing the Messaging Unit registers through SCS[0] BAR space, the PCI Access Control registers must not contain the lowest 4Kbyte of SCS[0] BAR space, see Section 8.8 "PCI Target Operation" on page 164.

The polarity of the messaging unit doorbells, interrupt cause, and interrupt mask registers bits are determined via the Queue Control register's Polarity bit, see Table 358 on page 271. If set to '0', interrupts are masked by a mask bit set to '0', cause bits are cleared by writing '0', and doorbell bits toggle by writing '0'. If set to '1', interrupts are masked by a mask bit set to '1', cause bits are cleared by writing '1', and doorbell bits toggle by writing '1'.

9.1 Message Registers

The GT–64240A uses the message registers to send and receive short messages over the PCI bus, without transferring data into local memory. When written to, the message registers may cause an interrupt to be generated either to the CPU or to the PCI bus. There are two types of message registers:

- Outbound messages sent by the GT–64240A's local CPU and received by an external PCI agent.
- Inbound messages sent by an external PCI bus agent and received by the GT–64240A's local CPU.

The interrupt status for outbound messages is recorded in the Outbound Interrupt Cause Register.

Interrupt status for inbound messages is recorded in the Inbound Interrupt Cause Register.

9.1.1 Outbound Messages

There are two Outbound Message Registers (OMRs).

When an OMR is written from the CPU side, a maskable interrupt request is generated in the Outbound Interrupt Status Register (OISR). If this request is unmasked, an interrupt request is issued on the PCI bus. The interrupt is cleared when an external PCI agent writes a value of '1' to the Outbound Message Interrupt bit in the OISR. The interrupt may be masked through the mask bits in the Outbound Interrupt Mask Register.

NOTE: An OMR can be written by the CPU or by the other PCI interface. It allows passing messages between CPU and PCI and between the two PCI interfaces.

9.1.2 Inbound Messages

There are two Inbound Message Registers (IMRs).

When an IMR is written from the PCI side, a maskable interrupt request is generated in the Inbound Interrupt Status Register (IISR). If this request is unmasked, an interrupt is issued to the CPU. The interrupt is cleared when the CPU writes a value of '1' to the Inbound Message Interrupt bit in the IISR. The interrupt may be masked through the mask bits in the Inbound Interrupt Mask Register.

NOTE: An inbound message sent from PCI bus can be targeted to the CPU or to the other PCI interface. The destination depends on the software setting of the interrupt mask registers, see Section 21.1.2 "Interrupts Mask Registers" on page 509.

9.2 Doorbell Registers

The GT–64240A uses the doorbell registers to request interrupts on both the PCI and CPU buses. There are two types of doorbell registers:

- Outbound doorbells are set by the GT–64240A's local CPU to request an interrupt service on the PCI bus.
- Inbound doorbells are set by an external PCI agent to request interrupt service from the local CPU.

9.2.1 Outbound Doorbells

The local processor can generate an interrupt request to the PCI bus by setting bits in the Outbound Doorbell Register (ODR). The interrupt may be masked in the OIMR register. However, masking the interrupt does not prevent the corresponding bit from being set in the ODR.

External PCI agents clear the interrupt by setting bits in the ODR (writing a '1').

NOTE: The CPU or the other PCI interface can set the ODR bits. This allows for passing interrupt requests not only between CPU and PCI, but also between the two PCI interfaces.

9.2.2 Inbound Doorbells

The PCI bus can generate an interrupt request to the local processor by setting bits in the Inbound Doorbell Register (IDR). The interrupt may be masked in the IIMR register. However, masking the interrupt does not prevent the corresponding bit from being set in the IDR.

The CPU clears the interrupt by setting bits in the IDR (writing a '1').

NOTE: The interrupt request triggered from the PCI bus can be targeted to the CPU or to the other PCI interface, depending on software setting of the interrupt mask registers.

9.3 Circular Queues

NOTE: Circular queues are only supported with I_2O ports being accessed in the first 4K of SCS[0] BAR.

The circular queues form the heart of the I_2O message passing mechanism and are the most powerful part of the messaging unit built into the GT–64240A. There are two inbound and two outbound circular queues in the Messaging Unit (MU).

NOTE: Whenever a reference is made to messages coming to or from the CPU, it also applies to messages coming to or from the other PCI interface.

9.3.1 Inbound Message Queues

The two inbound message queues are:

- Inbound Post
  Messages from other PCI agents that the CPU must process.
- Inbound Free
  Messages from the CPU to other PCI agent in response to an incoming message.

The two inbound message queues allow external PCI agents to post inbound messages to the local CPU in one queue and receive free messages (no longer in use) returning from the local CPU. The process is as follows:

1. An external PCI agent posts an inbound message.
2. The CPU receives and processes the message.
3. When the processing is complete, the CPU places the message back into the inbound free queue so that it may be reused.

9.3.2 Outbound Message Queues

The two outbound message queues are:

- Outbound Post
  Messages from the CPU to other PCI agents to process.
- Outbound Free
  Messages from other PCI agents to the CPU in response to an outgoing message.

The two outbound queues allow the CPU to post outbound messages for external PCI agents in one queue and receive free messages (no longer in use) returning from other external PCI agents. The process is as follows:

1. The CPU posts an outbound message.
2. The external PCI agent receives and processes the message.
3. When the processing is complete, the external PCI agent places the message back into the outbound free queue so that it may be reused.

9.3.3 Circular Queues Data Storage

Data storage for the circular queues must be allocated in local memory. It can be placed in any of SCS[3:0] BARs address ranges, depending on the setting of CirQDev bits in Queue Control register. The base address for the queues is set in the Queue Base Address Register (QBAR). Each queue entry is a 32-bit data value. The circular queue sizes range from 4K entries (16Kbytes) to 64K entries (256Kbytes) yielding a total local memory allotment of 64Kbytes to 1Mbyte. All four queues must be the same size and be contiguous in the memory space. Queue size is set in the Queue Control Register.

The starting address of each queue is based on the QBAR address and the size of the queues as shown inTable 343.

Table 343: Circular Queue Starting Addresses

Queue	Starting Address
Inbound Free	QBAR
Inbound Post	QBAR + Queue Size
Outbound Post	QBAR + 2*Queue Size
Outbound Free	QBAR + 3*Queue Size

Each queue has a head pointer and a tail pointer which are kept in the GT–64240A internal registers. These pointers are offsets from the QBAR. Writes to a queue occur at the head of the queue.Reads occur from the tail. The head and tail pointers are incremented by either the CPU software or messaging unit hardware. The pointers wrap around to the first address of a queue when they reach the queue size.

NOTE: PCI read/write from a queue is always a single 32-bit word. An attempt to burst from an I_2O queue results in disconnect after the first data transfer. Additionally, the GT–64240A never responds with ACK64* to an attempt to access the queue with a 64-bit transaction.

9.3.4 Inbound/Outbound Queue Port Function

Circular queues are accessed by external PCI agents through the Inbound and Outbound Queue Port virtual registers.

NOTE: With circular queues, you are not reading/writing a physical register within the GT–64240A. Instead, you are reading and writing pointers into the circular queues (located in SDRAM or Device) controlled by the GT–64240A. Refer to Figure 35 as you read the following sections.

When an Inbound Queue Port (IQP) is written from the PCI, the written data is placed on the Inbound Post Queue; it is posting the message to the local CPU.

When the Inbound Post Queue is written to alert the CPU that a message needs processing, an interrupt is generated to the CPU.

When this register is read from the PCI side, it is returning a free message from the tail of Inbound Free Queue.

The Outbound Queue Port (OQP) returns data from the tail of the Outbound Post Queue when read from the PCI side; it is returning the next message requiring service by the external PCI agent. When this register is written from the PCI, the data for the write is placed on the Outbound Free Queue; thus returning a free message for reuse by the local CPU.

Figure 35: I_2O Circular Queue Operation

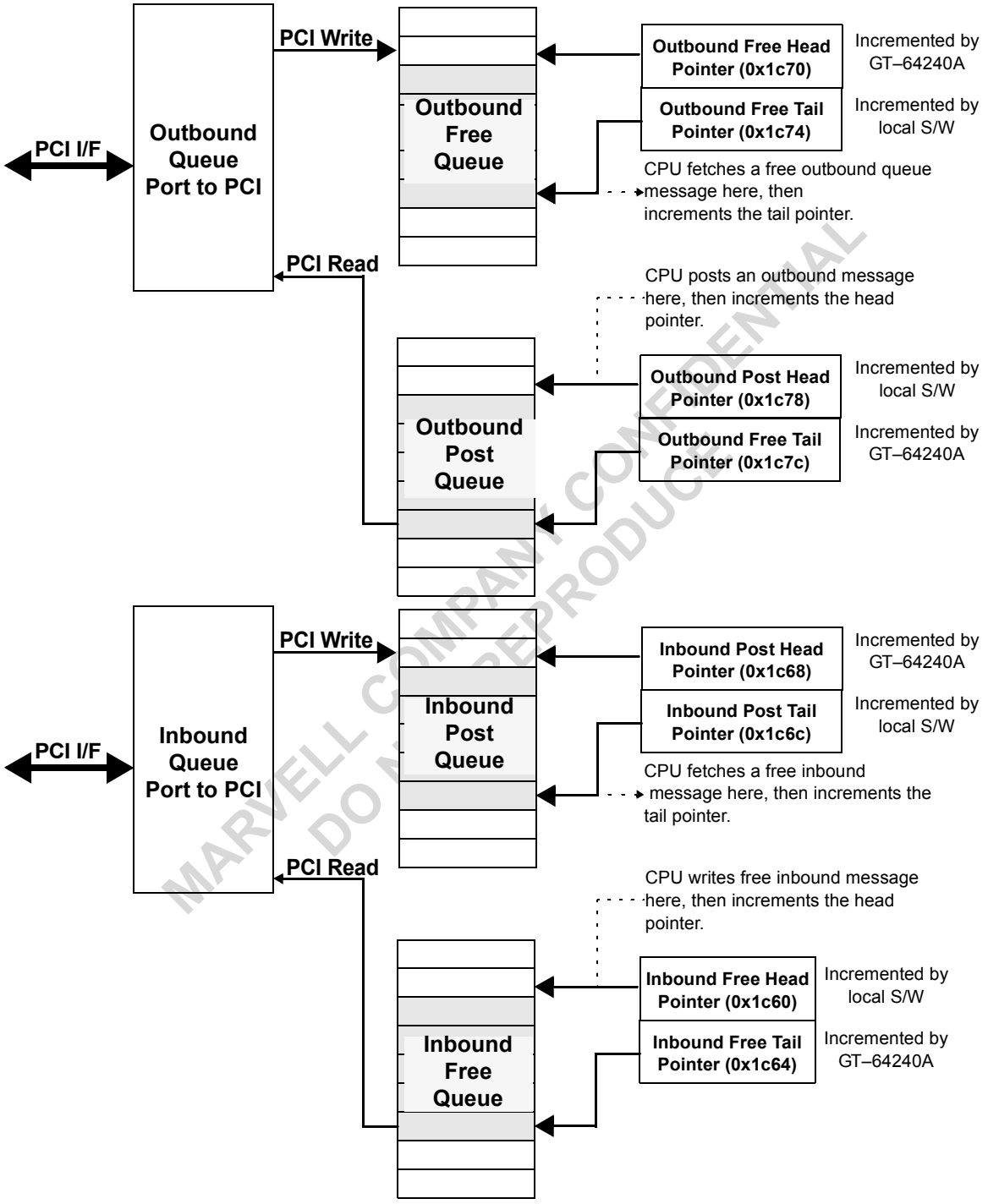


Table 344: I_2O Circular Queue Functional Summary

Queue Name	PCI Port	Generate PCI Interrupt?	Generate CPU Interrupt?	Head Pointer maintained by...	Tail Pointer maintained by...
Inbound Post	Inbound Queue Port	No	Yes, when queue is written	GT–64240A	CPU
Inbound Free		Yes, when queue is full.	No	CPU	GT–64240A
Out-bound Post	Out-bound Queue Port	Yes, when queue is not empty.	No	CPU	GT–64240A
Out-bound Free		No	Yes, when queue is full	GT–64240A	CPU

9.3.5 Inbound Post Queue

The Inbound Post Queue holds posted messages from external PCI agents to the CPU.

The CPU fetches the next message process from the queue tail; external agents post new messages to the queue head. The tail pointer is maintained by the CPU. The head pointer is maintained automatically by the GT–64240A upon posting of a new inbound message.

PCI writes to the Inbound Queue Port are passed to a local memory location at QBAR + Inbound Post Head Pointer. After this write completes, the GT–64240A increments the Inbound Post Head Pointer by 4 bytes (1 word); it now points to the next available slot for a new inbound message. An interrupt is also sent to the CPU to indicate the presence of a new message pointer.

From the time the PCI write ends till the data is actually written to SDRAM or Device, any new write to the Inbound port results in RETRY. If the queue is full, a new PCI write to the queue results in RETRY.

Inbound messages are fetched by the CPU by reading the contents of the address pointed to by the Inbound Post Tail Pointer. It is the CPUs responsibility to increment the tail pointer to point to the next unread message.

9.3.6 Inbound Free Queue

The Inbound Free Queue holds available inbound free messages for external PCI agents to use.

The CPU places free message at the queue head; external agents fetch free messages from the queue tail. The head pointer is maintained in software by the CPU. The tail pointer is maintained automatically by the GT–64240A upon a PCI agent fetching a new inbound free message.

PCI reads from the Inbound Queue Port return the data in the local memory location at QBAR + Inbound Free Tail Pointer. The following conditions apply:

- If the Inbound Free Queue is not empty (as indicated by Head Pointer not equal to Tail Pointer), the data pointed to by QBAR + Inbound Free Tail Pointer is returned.
- If the queue is empty (Head Pointer equals Tail Pointer), the value 0xFFFF.FFFF is returned. Indicating that there are no Inbound Message slots available. This is an error condition.

The processor places free message buffers in the Inbound Free Queue by writing the message to the location pointed to by the head pointer. It is the processor's responsibility to then increment the head pointer.

NOTE: It is the CPU's responsibility to make sure that the PCI agent keeps up the pace of the free messages and avoid pushing a new free message to the queue if it is full. There is no overflow indication when the Inbound Free Queue is full.

9.3.7 Outbound Post Queue

The Outbound Post Queue holds outbound posted messages from the CPU to external PCI agents.

The CPU places outbound messages at the queue head; external agents fetch the posted messages from the queue tail. The Outbound Post Tail Pointer is automatically incremented by the GT–64240A; the head pointer must be incremented by the local CPU.

PCI reads from the Outbound Queue Port return the data pointed to by QBAR + Outbound Post Tail Pointer (the next posted message in the Outbound Queue.) The following conditions apply:

- If the Outbound Post Queue is not empty (the head and tail pointers are not equal), the data is returned as usual and the GT–64240A increments the Outbound Post Tail Pointer.
- If the Outbound Post Queue is empty (the head and tail pointers are equal), the value 0xFFFF.FFFF is returned.

As long as the Outbound Post Head and Tail pointers are not equal, a PCI interrupt is requested. This is done to indicate the need to have the external PCI agent read the Outbound Post Queue. When the head and tail pointers are equal, no PCI interrupt is generated since no service is required on the part of the external PCI agent (or PCI system host in the case of a PC server.) In either case, the interrupt can be masked in the OIMR register.

The CPU places outbound messages in the Outbound Post Queue by writing to the local memory location pointed to by the Outbound Post Head Pointer. After writing this pointer, it is the CPU's responsibility to increment the head pointer.

9.3.8 Outbound Free Queue

The Outbound Free Queue holds available outbound message buffers for the local processor to use.

External PCI agents place free messages at the queue head; the CPU fetches free message pointers from the queue tail. The tail pointer in maintained in software by the CPU. The head pointer is maintained automatically by the GT–64240A upon a PCI agent posting a new ("returned") outbound free message.

PCI writes to the Outbound Queue Port result in the data being written to the local memory location at QBAR + Outbound Free Head Pointer. After the write completes, the GT–64240A increments the head pointer.

From the time the PCI write ends till the data is actually written to SDRAM or Device, any new write to Outbound port will result in RETRY. If the head pointer and tail pointer become equal (an indication that the queue is full), an interrupt is sent to the CPU. If queue is full, a new PCI write to the queue will result in RETRY.

The processor obtains free outbound message buffers from the Outbound Free Queue by reading data from the location pointed to by the tail pointer. It is the processor's responsibility to increment the tail pointer.

9.3.9 Queue Data Endianess

Circular Queues access is not controlled by PCI Access Control registers. The endianess convention of data placed in the circular queues is determined by SByteSwap and SWordSwap bits of PCI Command register. For more details, see Section 8.13 "Data Endianess" on page 170.

9.4 Messaging Unit Registers

NOTE: The offsets listed below relate to a CPU or PCI access to the Messaging Unit registers through the GT–64240A internal registers space.

If the register is accessed from PCI_0 through the SCS[0] BAR space, remove the offset's 0x1c prefix. For example, in the SCS[0] BAR space, the PCI_0 Outbound Interrupt Cause register is located at offset 0x30 and the PCI_1 Outbound Interrupt Cause register is located at offset 0xb0 . Also, if accessed from PCI_1 through SCS[0] BAR space, exchange the PCI_0 and PCI_1 registers offsets. This means that in the SCS[0] BAR space the PCI_0 Outbound Interrupt Cause register is located at offset 0xb0 and the PCI_1 Outbound Interrupt Cause register is located at offset 0x30.

Table 345: Messaging Unit Register Map

Register	Offset		Page
	PCI_0	PCI_1	
Inbound Message Register 0	0x1c10	0x1c90	page 266
Inbound Message Register 1	0x1c14	0x1c94	page 266
Outbound Message Register 0	0x1c18	0x1c98	page 267
Outbound Message Register 1	0x1c1c	0x1c9c	page 267
Inbound Doorbell Register	0x1c20	0x1ca0	page 267
Inbound Interrupt Cause Register	0x1c24	0x1ca4	page 267
Inbound Interrupt Mask Register	0x1c28	0x1ca8	page 268
Outbound Doorbell Register	0x1c2c	0x1cac	page 269
Outbound Interrupt Cause Register	0x1c30	0x1cb0	page 269
Outbound Interrupt Mask Register	0x1c34	0x1cb4	page 270
Inbound Queue Port Virtual Register	0x1c40	0x1cc0	page 271

Table 345: Messaging Unit Register Map (Continued)

Register	Offset		Page
	PCI_0	PCI_1	
Outbound Queue Port Virtual Register	0x1c44	0x1cc4	page 271
Queue Control Register	0x1c50	0x1cd0	page 271
Queue Base Address Register	0x1c54	0x1cd4	page 272
Inbound Free Head Pointer Register	0x1c60	0x1ce0	page 272
Inbound Free Tail Pointer Register	0x1c64	0x1ce4	page 273
Inbound Post Head Pointer Register	0x1c68	0x1ce8	page 273
Inbound Post Tail Pointer Register	0x1c6c	0x1cec	page 273
Outbound Free Head Pointer Register	0x1c70	0x1cf0	page 274
Outbound Free Tail Pointer Register	0x1c74	0x1cf4	page 274
Outbound Post Head Pointer Register	0x1cf8	0x1c78	page 275
Outbound Post Tail Pointer Register	0x1cfc	0x1c7c	page 275

Table 346: Inbound Message0

- PCI_0 Offset: 0x1c10
- PCI_1 Offset: 0x1c90

Bits	Field Name	Function	Initial Value
31:0	InMsg0	Inbound Message Register Read only from the CPU, or other PCI interface. When written, sets a bit in the Inbound Interrupt Cause Register and an interrupt is generated to the CPU, or other PCI interface.	0x0

Table 347: Inbound Message1

- PCI_0 Offset: 0x1c14
- PCI_1 Offset: 0x1c94

Bits	Field Name	Function	Initial Value
31:0	InMsg1	Same as Inbound Message0.	0x0

Table 348: Outbound Message0

- PCI_0 Offset: 0x1c18
- PCI_1 Offset: 0x1c98

Bits	Field Name	Function	Initial Value
31:0	OutMsg0	Outbound Message Register Read only from the PCI. When written, sets bit in the Outbound Interrupt Cause Register and an interrupt is generated to the PCI.	0x0

Table 349: Outbound Message1

- PCI_0 Offset: 0x1c1
- PCI_1 Offset: 0x1c9c

Bits	Field Name	Function	Initial Value
31:0	OutMsg1	Same as Outbound Message0.	0x0

Table 350: Inbound Doorbell

- PCI_0 Offset: 0x1c20
- PCI_1 Offset: 0x1ca0

Bits	Field Name	Function	Initial Value
31:0	InDoor	Inbound Doorbell Register The PCI setting a bit in this register to ‘1’ causes a CPU (or other PCI interface) interrupt. Writing ‘1’ to the bit by the CPU (or other PCI interface) clears the bit, and deasserts the interrupt).	0x0

Table 351: Inbound Interrupt Cause

- PCI_0 Offset: 0x1c24
- PCI_1 Offset: 0x1ca4

Bits	Field Name	Function	Initial Value
0	InMsg0	Inbound Message0 Interrupt Set when the Inbound Message0 register is written. The CPU writes a ‘1’ to clear it.	0x0
1	InDoorL	Inbound Doorbell Interrupt bits [15:0] Set when at least one bit [15:0] of the Inbound Doorbell register is set. Read Only.	0x0

Table 351: Inbound Interrupt Cause (Continued)

- PCI_0 Offset: 0x1c24
- PCI_1 Offset: 0x1ca4

Bits	Field Name	Function	Initial Value
3:2	Reserved	Reserved.	0x0
4	InPQ	Inbound Post Queue Interrupt Set when Inbound Post Queue gets written. The CPU writes it with a '1' to clear it.	0x0
5	OutFQOvr	Outbound Free Queue Overflow Interrupt Set when Outbound Free Queue is full. The CPU writes it with a '1' to clear it.	0x0
15:6	Reserved	Reserved.	0x0
16	InMsg1	Inbound Message1 Interrupt Set when Inbound Message1 register is written. The CPU writes it with a '1' to clear it.	0x0
17	InDoorH	Inbound Doorbell Interrupt bits [31:16] Set when at least one bit[31:16] of Inbound Doorbell register is set. Read Only.	0x0
31:18	Reserved	Reserved.	0x0

Table 352: Inbound Interrupt Mask

- PCI_0 Offset: 0x1c28
- PCI_1 Offset: 0x1ca8

Bits	Field Name	Function	Initial Value
0	InMsg0	If set to '1', the Inbound Message0 interrupt is enabled.	0x1
1	InDoorL	If set to '1', the Inbound Doorbell [15:0] interrupt is enabled.	0x1
3:2	Reserved	Reserved.	0x3
4	InPQ	If set to '1', the Inbound Post Queue interrupt is enabled.	0x1
5	OutFQOvr	If set to '1', the Outbound Free Queue Overflow interrupt is enabled.	0x1
15:6	Reserved	Reserved.	0x3ff
16	InMsg1	If set to '1', the Inbound Message1 interrupt is enabled.	0x1
17	InDoorH	If set to '1', the Inbound Doorbell [31:16] interrupt is enabled.	0x1

Table 352: Inbound Interrupt Mask (Continued)

- PCI_0 Offset: 0x1c28
- PCI_1 Offset: 0x1ca8

Bits	Field Name	Function	Initial Value
31:24	Reserved	Reserved.	0x0

Table 353: Outbound Doorbell

- PCI_0 Offset: 0x1c2c
- PCI_1 Offset: 0x1cac

Bits	Field Name	Function	Initial Value
31:0	OutDoor	Outbound Doorbell Register Setting a bit in this register to '1' by the CPU causes a PCI interrupt. Writing '1' to this bit by the PCI clears the bit, and deas-sert the interrupt.	0x0

Table 354: Outbound Interrupt Cause

- PCI_0 Offset: 0x1c30
- PCI_1 Offset: 0x1cb0

Bits	Field Name	Function	Initial Value
0	OutMsg0	Outbound Message0 Interrupt Set when the Outbound Message0 register is written. The PCI writes it with '1' to clear it. For the CPU, it is Read Only.	0x0
1	OutDoorL	Outbound Doorbell Interrupt bits[15:0] Set when at least one bit[15:0] of Outbound Doorbell register is set. Read Only.	0x0
2	Reserved	Reserved.	0x0
3	OutPQ	Outbound Post Queue Interrupt Set as long as Outbound Post Queue is not empty. This bit is read only.	0x0
15:4	Reserved	Reserved	0x0

Table 354: Outbound Interrupt Cause (Continued)

- PCI_0 Offset: 0x1c30
- PCI_1 Offset: 0x1cb0

Bits	Field Name	Function	Initial Value
16	OutMsg1	Outbound Message1 Interrupt Set when the Outbound Message1 Register is written. The PCI writes it with '1' to clear it. For the CPU, it is read only.	0x0
17	OutDoorH	Outbound Doorbell Interrupt bits[31:16] Set when at least one bit[31:16] of Outbound Doorbell register is set. Read Only.	0x0
31:18	Reserved	Reserved.	0x0

Table 355: Outbound Interrupt Mask Register

- PCI_0 Offset: 0x1c34
- PCI_1 Offset: 0x1cb4

Bits	Field Name	Function	Initial Value
0	OutMsg0	If set to '1', Outbound Message0 interrupt is enabled.	0x1
1	OutDoorL	If set to '1', Outbound Doorbell [15:0] interrupt is enabled.	0x1
2	Reserved	Reserved.	0x1
3	OutPQ	If set to '1', Outbound Post Queue interrupt is enabled.	0x1
15:4	Reserved	Reserved.	0xfff
16	OutMsg1	If set to '1', Outbound Message 1 interrupt is enabled.	0x1
17	OutDoorH	If set to '1', Outbound Doorbell 31:16] interrupt is enabled.	0x1
31:18	Reserved	Reserved.	0x0

Table 356: Inbound Queue Port Virtual Register

- PCI_0 Offset: 0x1c40
- PCI_1 Offset: 0x1cc0

Bits	Field Name	Function	Initial Value
31:0	InQPVReg	Inbound Queue Port Virtual Register A PCI write to this port results in a write to the Inbound Post Queue. A read from this port results in a read from the Inbound Free Queue. Reserved from the CPU side.	0x0

Table 357: Outbound Queue Port Virtual Register

- PCI_0 Offset: 0x1c44
- PCI_1 Offset: 0x1cc4

Bits	Field Name	Function	Initial Value
31:0	OutQPVReg	Outbound Queue Port Virtual Register A PCI write to this port results in a write to the Outbound Free Queue. A read from this port results in a read from the Outbound Post Queue. Reserved from CPU side.	0x0

Table 358: Queue Control

- PCI_0 Offset: 0x1c50
- PCI_1 Offset: 0x1cd0

Bits	Field Name	Function	Initial Value
0	CirQEn	Circular Queue Enable If '0', any PCI write to the queue is ignored. Upon a PCI read from the queue, 0xffffffff is returned. Read Only from PCI side.	0x0
5:1	CirQSize	Circular Queue Size 00001 - 16 Kbytes 00010 - 32 Kbytes 00100 - 64 Kbytes 01000 - 128 Kbytes 10000 - 256 Kbytes Read Only from the PCI side.	0x1

Table 358: Queue Control (Continued)

- PCI_0 Offset: 0x1c50
- PCI_1 Offset: 0x1cd0

Bits	Field Name	Function	Initial Value
7:6	CirQDev	Circular Queue Location 00 - SCS[0]* space 01 - SCS[1]* space 10 - SCS[2]* space 11 - SCS[3]* space Read Only from the PCI side.	0x0
8	Reserved	Reserved.	0x0
31:9	Reserved		0x0

Table 359: Queue Base Address Register

- PCI_0 Offset: 0x1c54
- PCI_1 Offset: 0x1cd4

Bits	Field Name	Function	Initial Value
19:0	Reserved	Reserved.	0x0
31:20	QBAR	Queue Base Address Register Read Only from the PCI side.	0x0

Table 360: Inbound Free Head Pointer Register

- PCI_0 Offset: 0x1c60
- PCI_1 Offset: 0x1ce0

Bits	Field Name	Function	Initial Value
1:0	Reserved	Reserved.	0x0
19:2	InFHPtr	Inbound Free Head Pointer Read only from the PCI side. **NOTE:** This register is maintained by the CPU software.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

Table 361: Inbound Free Tail Pointer Register

- PCI_0 Offset: 0x1c64
- PCI_1 Offset: 0x1ce4

Bits	Field Name	Function	Initial Value
1:0	Reserved	Reserved.	0x0
19:2	InFTPtr	Inbound Free Tail Pointer Read only from the PCI side. **NOTE:** This register is incremented by the GT–64240A after the PCI read from the Inbound port.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

Table 362: Inbound Post Head Pointer Register

- PCI_0 Offset: 0x1c68
- PCI_1 Offset: 0x1ce8

Bits	Field Name	Function	Initial Value
1:0	Reserved	Reserved.	0x0
19:2	InPHPtr	Inbound Post Head Pointer Read only from PCI side. **NOTE:** This register is incremented by the GT–64240A after the PCI write to the Inbound port.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

Table 363: Inbound Post Tail Pointer Register

- PCI_0 Offset: 0x1c6c
- PCI_1 Offset: 0x1cec

Bits	Field Name	Function	Initial Value
1:0	Reserved	Reserved.	0x0
19:2	InPTPtr	Inbound Post Tail Pointer Read only from the PCI side. **NOTE:** This register is maintained by the CPU software.	0x0

Table 363: Inbound Post Tail Pointer Register

- PCI_0 Offset: 0x1c6c
- PCI_1 Offset: 0x1cec

Bits	Field Name	Function	Initial Value
31:20	QBAR	Queue Base Address Register Read only.	0x0

Table 364: Outbound Free Head Pointer Register

- PCI_0 Offset: 0x1c70
- PCI_1 Offset: 0x1cf0

Bits	Field Name	Function	Initial Value
1:0	Reserved	Reserved.	0x0
19:2	OutFHPtr	Outbound Free Head Pointer Read only from the PCI side. **NOTE:** This register is incremented by the GT–64240A after the PCI write to the Outbound port.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

Table 365: Outbound Free Tail Pointer Register

- PCI_0 Offset: 0x1c74
- PCI_1 Offset: 0x1cf4

Bits	Field Name	Function	Initial Value
1:0	Reserved	Reserved.	0x0
19:2	OutFTPtr	Outbound Free Tail Pointer Read Only from PCI side. **NOTE:** This register is maintained by the CPU software.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

Table 366: Outbound Post Head Pointer Register

- PCI_0 Offset: 0x1c78
- PCI_1 Offset: 0x1cf8

Bits	Field Name	Function	Initial Value
1:0	Reserved	Reserved.	0x0
19:2	OutPHPtr	Outbound Post Head Pointer Read only from the PCI side. **NOTE:** This register is maintained by the CPU software.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

Table 367: Outbound Post Tail Pointer Register

- PCI_0 Offset: 0x1c7c
- PCI_1 Offset: 0x1cfc

Bits	Field Name	Function	Initial Value
1:0	Reserved	Reserved.	0x0
19:2	OutPTPtr	Outbound Post Tail Pointer Read only from the PCI side. **NOTE:** This register is incremented by the GT–64240A after the PCI read from the Outbound port.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

10. IDMA CONTROLLER

The GT–64240A has eight independent IDMA engines.

The IDMA engines optimize system performance by moving large amounts of data without significant CPU intervention. Instead of the CPU reading data from a source and writing it to destination, an IDMA engine can be programmed to automatically transfer data independent of the CPU. This allows the CPU to continue executing other instructions, simultaneous to the movement of data.

Each IDMA engine can move data between any source and any destination, such as the SDRAM, Device, PCI_0, or PCI_1. The IDMA controller can be programmed to move up to 16Mbyte of data per transaction. The burst length of each transfer of IDMA can be set from 1 to 128 bytes. Accesses can be non-aligned both in the source and the destination.

The IDMA channels support chained mode of operation. The chain descriptors may be placed anywhere. For example, IDMA can transfer data from SDRAM to PCI_0 using chain mode, while fetching new descriptors from a Device. The IDMA engine moves the data until a null descriptor pointer is reached.

The IDMA can be triggered by the CPU writing a register, an external request via a DMAReq* pin, or from a timer/counter. In cases where the transfer needs to be externally terminated, an End of Transfer pin can be asserted for the corresponding IDMA channel.

10.1 IDMA Operation

The IDMA unit contains two 2Kbyte buffers. Each buffer is coupled to four IDMA channels - channels 0-3 use one buffer and channels 4-7 use the other buffer. Each channels has a dedicated 512 bytes slice of the buffer.

When a channel is activated, data is read from the source into the channels buffer and then written to the destination. While writing the data to the destination, the channel reads the next burst into the buffer. This read/write behavior results in a minimal gap between consecutive IDMA transactions on the source and the destination interfaces. In cases of a PCI access, this read/write behavior enables generating a very long burst with zero wait states (using the PCI master interface combining feature).

This buffer structure enables concurrency of transactions between channels. For example, if channel 0 is moving data from PCI_0 to PCI_1 and channel 4 is moving data from SDRAM to Device, the two channels work independently. They don't share resources and run concurrently.

Since each buffer's four channels share the same resources, arbitration of resources is required. Each four channels has a configurable round-robin arbiter that allows different bandwidth allocation to each channel within the group, see Section 10.6 "Arbitration" on page 285.

10.2 IDMA Descriptors

Each IDMA Channel Descriptor consists of four 32-bit registers that can be written to by the CPU, PCI, or IDMA controller in the process of fetching a new descriptor from memory (in case of chain mode). Each channel can be configured to work in a compatibility mode, in which the descriptor structure is the same as in GT-64120 and GT-64130 devices, or work with new descriptor structure, as shown in Figure 36.

Figure 36: IDMA Descriptors

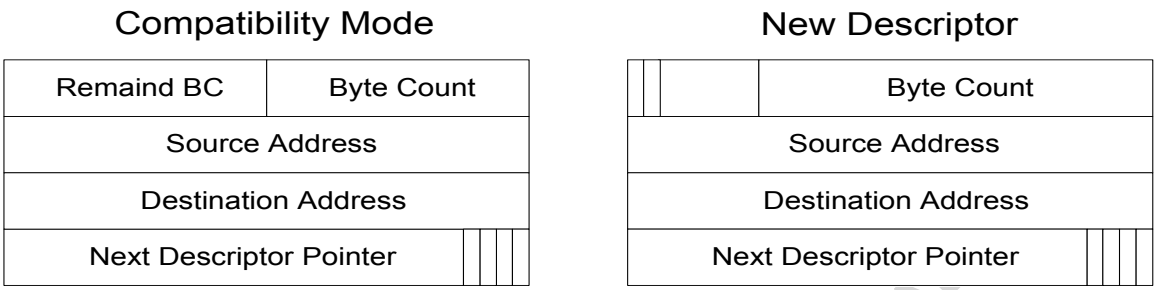


Table 368: DMA Descriptor Definitions

DMA Descriptor	Definition
Byte Count	Number of bytes of data to transfer. The maximum number of bytes which the IDMA controller can be configured to transfer is 64Kbyte-1 (16-bit register) in compatibility mode or 16Mbyte-1 (24-bit register) in the new descriptor structure. This register decrements at the end of every burst of transmitted data from the source to the destination. When the byte count register is 0, or the End of Transfer pin is asserted, the IDMA transaction is finished or terminated.
Source Address	Bits[31:0] of the IDMA source address. According to the setting of the Channel Control register, this register either increments or holds the same value. **NOTE:** For more information on the Channel Control register, see Section 10.9.2 "IDMA Channel Control Registers" on page 301.
Destination Address	Bits[31:0] of the IDMA destination address. According to the setting of the Channel Control register, this register either increments or holds the same value.
Pointer to the Next Descriptor	Bits[31:0] of the IDMA Next Descriptor address for chained operation. The descriptor must be 16 sequential bytes located at 16-bytes aligned address (bits[3:0] are 0). **NOTE:** Only used when the channel is configured to Chained Mode.

The upper bits of the byte count register are explained in Section 10.5.8 "Descriptor Ownership" on page 284.

NOTE: Source, destination and next descriptor addresses are 36-bit wide. The upper four bits of the address are not part of the dynamic 32 byte descriptor. These bits are fixed for the whole IDMA chain. An IDMA transfer is restricted to not cross 4Gbyte (32-bit address) boundary.

Figure 37 on page 280shows the basic IDMA operation.

10.3 IDMA Address Decoding

With each IDMA transaction, IDMA engine first compares the address (source, destination, or descriptor) against the CPU interface address decoding registers. This comparison is done to select the correct target interface (SDRAM, Device, or PCI). The address decoding process is the same as CPU address decoding, see Section 3.1 “CPU Address Decoding” on page 44.

If the address does not match any of the address windows, an interrupt is generated and the IDMA engine is stopped.

There might be cases where an IDMA access to the PCI is required to address space that is out of CPU-to-PCI address windows. In this case, the IDMA to PCI override feature can be used. The source, destination, and next descriptor address for each channel can be marked as PCI override, meaning the IDMA engine accesses the PCI interface directly without executing any address decoding.

The PCI interface supports 64-bit addressing. Each IDMA channel generates a 64-bit address to the PCI interface via source, destination, and next descriptor PCI High Address register. If the PCI High Address register value is ‘0’, the PCI master issues a SAC transaction. If it is not 0 (which means address is beyond 4Gbyte space), the PCI master generates a DAC transaction.

NOTES: There is no IDMA address remapping to the PCI. Due to the PCI override feature, it is not required.

IDMA always uses its own PCI High Address registers, even if not using PCI override.

10.4 IDMA Access Protection

Each IDMA transaction address is also checked against the CPU interface’s Access Protect registers. If the address matches one of those regions, and the transaction violates the region protection, the IDMA halts and an interrupt is asserted. For full details, see Section 4.15.4 “CPU Access Protect Registers” on page 93.

NOTE: IDMA access protection includes write protect and access protect. Unlike the CPU, there is no caching protection. Caching protection is meaningless in the case of IDMA.

10.5 IDMA Channel Control

Each IDMA Channel has its own unique control register where certain IDMA modes are programmed. Following are the bit descriptions for each field in the control registers. For detailed registers description, see Section 10.9.2 “IDMA Channel Control Registers” on page 301.

10.5.1 Address Increment/Hold

The IDMA engine supports both increment and hold modes.

If the SrcHold, bit [3], is set to ‘0’, the IDMA automatically increments the source address with each transfer.

If the SrcHold bit is set to ‘1’, the source address remains constant throughout the IDMA burst.

Similarly, If the DestHold, bit [5], is set to ‘0’, the IDMA automatically increments the destination address.

If the DestHold bit is set to ‘1’, the destination address remains constant throughout the IDMA burst.

Setting the SrcHold or DestHold bits is useful when the source/destination device is accessible through a constant address. For example, if the source/destination device is a FIFO, it is accessed with a single address, while data is being popped/pushed with each IDMA burst.

NOTE: When using Hold mode, the address is restricted to be aligned to the Burst Limit setting, see the Channel Control (Low) register's BurstLimit bits [8:6] on Table 437 on page 301.

10.5.2 Burst Limit

The whole IDMA byte count is chopped into small bursts.

The burst limit can vary from 8 to 128 bytes in modulo-2 steps (i.e. 8, 16..., 128). It determines the burst length of IDMA transaction against the source and destination. For example, setting the burst limit to 64 bytes means that the IDMA reads 64 bytes from the source and then writes the data to the destination. The IDMA continues this read/write loop until transfer of the whole byte count is complete.

The burst limit setting is affected by the source and destination characteristics, as well as system bandwidth allocation considerations.

NOTE: Regardless of the burst limit setting, the fetch of a new descriptor is always a 16 bytes burst. This implies that descriptors cannot be located in devices that don't support such bursts. Particularly, they can not be located in 8 or 16-bit devices on the GT–64240A device bus (see Section 7.3 "Data Pack/ Unpack and Burst Support" on page 143).

If an IDMA accesses a chache coherent DRAM regions, the burst limit must not exceed 32 bytes.

If the Channel Control (High) register's BLMode bit [31] (see Table 438 on page 304) is set to '1', the DMA engine usese a seperate burst limit for the souce and distination. The source burst limit is controlled by the DMA Control (Low) register's BurstLimit bits [8:6] (see Table 437 on page 301). The destination's burst limit is controlled by the same register's DstBurstLimit bits [2:0].

Seperately controlling the source and destination burst limit size is useful when one direction can use a large burst limit when the other direction has a restricted burst limit.

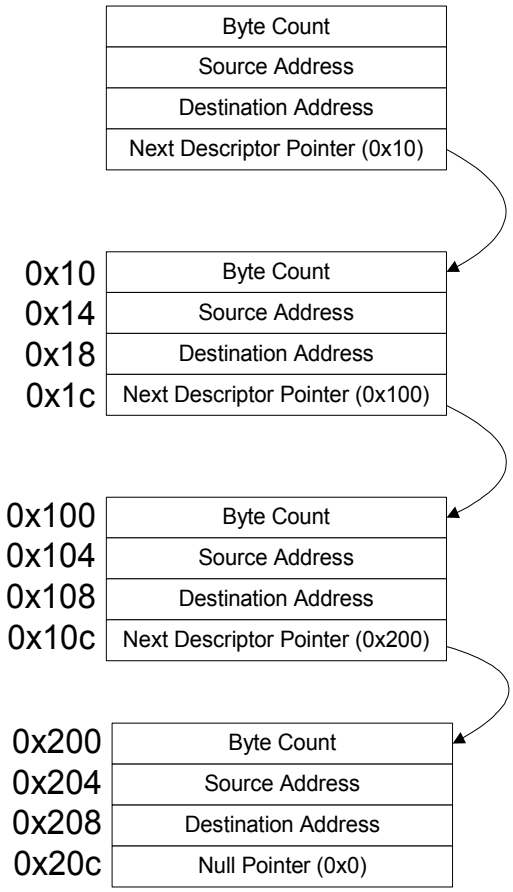
10.5.3 Chain Mode

When the ChainMode bit [9] is set to '0', chained mode is enabled.

In chain mode, at the completion of an IDMA transaction, the Pointer to Next Descriptor register provides the address of a new IDMA descriptor. If this register contains a value of '0' (NULL), this indicates that this is the last descriptor in the chain.

Figure 37 shows an example of an IDMA descriptors chain.

Figure 37: Chained Mode IDMA



Fetch next descriptor can be forced via the FetchND bit [13] in the Channel Control register.

Setting this bit to ‘1’ forces a fetch of the next descriptor based on the value in the Pointer to Next Descriptor register.

This bit can be set even if the current IDMA has not yet completed. In this case, the IDMA engine completes the current burst read and write and then fetches the next descriptor. This bit is reset back to ‘0’ after the fetch of the new descriptor is complete. Setting FetchND is not allowed if the descriptor equals Null.

NOTE: If using the FetchND bit while the current DMA is in progress, the DMA Control (Low) register’s Abr bit [20] must be set. See Table 437 on page 301.

The first descriptor of a chain can be set directly by programing the channels registers, or can be fetched from memory, using the FetchND bit. If fetched from memory, the next descriptor address must be first written to the Next Descriptor Pointer register of the channel. The channel then must be enabled by setting the Channel Control (Low) register’s ChanEn bit [12] to ‘1’ (see Section 10.5.4 “Channel Activation” on page 281) and setting FetchND to ‘1’.

When the IDMA transfer is done, an IDMA completion interrupt is set. When running in chain mode, the IntMode, bit [10] of the Channel Control register, controls whether to assert an interrupt on the completion of every

byte count transfer or only with last descriptor byte count completion. If set to ‘0’, the Comp bit is set every time the IDMA byte count reaches ‘0’. If set to ‘1’, the IDMAComp Interrupt bit is asserted when both the Pointer to Next Descriptor Register has a NULL value and byte count is 0.

If ChainMod is set to ‘1’, chained mode is disabled and the Pointer to Next Descriptor register is not loaded at the completion of the IDMA transaction.

NOTE: In non-chained mode the Byte Count, Source, and Destination registers must be initialized prior to enabling the channel.

If reading a new descriptor results in parity/ECC error indicated by the unit from which the descriptor is being read, the channel halts. This is done in order to prevent destructive reads/writes, due to bad source/destination pointers.

10.5.4 Channel Activation

Software channel activation is done via the Channel Control (Low) register’s ChanEn bit [12] (see Table 437 on page 301).

When set to ‘0’, the channel is disabled. When set to ‘1’, the IDMA is initiated based on the current setting loaded in the channel descriptor (i.e. byte count, source address, and destination address). An active channel can be temporarily stopped by clearing ChanEn bit and then continued later from the point it was stopped by setting ChanEn bit back to 1.

Clearing the ChanEn bit during IDMA operation does not guarantee an immediate channel pause. The IDMA engine must complete transferring the last burst it was working on. Software can monitor the channel status by reading ChanAct bit.

In order to restart a suspended channel in non-chained mode, the ChanEn bit must be set to ‘1’. In Chained mode, the software must find out if the first fetch took place. If the fetch did take place, only ChanEn bit is set to ‘1’. If the fetch did not take place, the FetchND bit must also be set to ‘1’.

The ChanAct bit [14] is read only. If set to ‘0’, the channel is not active. If set to ‘1’, the channel is active. In non-chain mode, this bit is deasserted when the byte count reaches zero. In chain-mode, this bit is deasserted when pointer to next descriptor is NULL and byte count reaches zero.

If ChanEn bit is set to ‘0’ during IDMA transfer, ChanAct bit toggles to ‘0’ as soon as the IDMA engine finishes the last burst it is working on.

In order to abort an IDMA transfer in the middle, software needs to set Abr bit [20] to ‘1’. Setting this bit has a similar affect to clearing ChanEn bit. However, it guarantees a smooth transfer of the IDMA engine to idle state. As soon as the IDMA is back in idle state, the Abr bit gets cleared, allowing the software to re-program the channel.

NOTES: If the byte count is smaller that the burst limit setting, the source and destination addresses must be aligned.

If the close descriptor feature is used, only set the Abr bit after first clearing the ChanEn bit and then the ChanAct bit.

Any write to the Channel Control register with ChanEn bit set to ‘1’ activates the channel. To program the channel control register,without activating the channel, the ChanEn bit must be set to ‘0’.

10.5.5 Source and Destination Addresses Alignment

The IDMA implementation maintains aligned accesses to both source and destination.

If source and destination addresses have different alignments, the IDMA performs multiple reads from the source to execute a write of full BurstLimit to the destination. For example, if the source address is 0x4, the destination address is 0x100, and BurstLimit is set to 8 bytes, the IDMA perform two reads from the source. First 4 bytes from address 0x4 then 8 bytes from address 0x8, and only then performs a write of 8 bytes to address 0x100.

This implementation guarantees that all reads from the source and all writes to the destination have all byte enables asserted (except for the IDMA block edges, in case they are not aligned). This is especially important when the source device does not tolerate read of extra data (destructive reads) or when the destination device does not support write byte enables.

NOTE: This implementation differs from the GT-64120 and GT-64130 devices. No SDA bit is required since the GT–64240A implementation keeps accesses to both source and destination aligned.

10.5.6 Demand Mode

The IDMA channel can be triggered by software via ChanEn bit (block mode) or by external assertion of DMAReq* pin (demand mode). Setting the DemandMode bit to '0', sets the channel to operate in demand mode.

Each channel is coupled to the DMAReq* and DMAAck* pins when working in demand mode. DMAReq* is the external trigger to activate the channel. DMAAck* is the channel response, notifying the external device that its request is being served.

Both DMAReq* and DMAAck* are multiplexed on MPP pins. If setting a channel to demand mode, the DMAReq* pin is mandatory. Setting a channel to demand mode without configuring an MPP pin to act as the channels DMAReq* causes the channel to hang. See Section 19.1 "MPP Multiplexing" on page 480 section for more information.

NOTE: Program the number of TClk cycles that DMAAck* is asserted through the DMAAck_Width bit [4], see Figure 437 on page 301.

DMAAck* cannot be targeted to both the source and destination devices. See the Channel x Control register's DMAAckDir bits [30:29] (Table 437 on page 301).

When running in demand mode, the IDMA moves a new BurstLimit of data upon demand, rather than continuos bursts from source to destination. This mode is required when the source device does not have the whole byte count in advance. It triggers a new burst limit transfer when it has a burst count available data to transfer. It can also be used in the compliment case, where the destination device cannot absorb the whole byte count, but only burst limit at a time.

The IDMA engine distinguishes between the DMAReq* generated by the source device, and DMAReq* generated by the target device, via DMAReqDir bit in the Channel Control register. If DMAReq* is generated by the source (DMAReqDir is set to '0'), the IDMA reads a new BurstLim of data from source with each new DMAReq assertion. However, it writes to the destination device whenever it can transfer a full BurstLim. In the alignment example in Section 10.5.5 Source and Destination Addresses Alignment, the first write to the destination occurs after two assertions of DMAReq* by the source. If DMAReq* is asserted by the destination (DMAReqDir is set to '1'), the DMA writes a new BurstLim of data to the destination device with each new DMAReq assertion. In this case, a read from the source occurs regardless of DMAReq* assertion.

NOTE: This implementation is different than the one in the GT-64120 and GT-64130. In these devices, each DMAReq* assertion results in a single read from source and write to the destination.

DMAReq* can be treated as level or edge triggered input, depending on the setting of DMAReqMode bit. When the device DMAReq* assertion is tightly coupled to the DMAAck* signal, an edge trigger DMAReq* might be needed, to prevent a redundant DMAReq* assertion due to late DMAReq* deassertion.

NOTE: The edge triggered DMAReq* is a new feature not supported by the GT-64120 and GT-64130. In these devices, the problematic DMAReq* deassertion timing is solved via the MDREQ bit. This bit is no longer supported.

The DMAAck* output pin indicates to the requesting device that the IDMA engine has finished transferring the current burst. DMAAck* can be configured to assert with the read from the source, with the write to destination, or with both read and write, via DMAAckDir bits. Setting DMAAck* to '1' results in DMAAck assertion with write access to the destination device.

Since the Device interface unit has a queue of transactions, actual IDMA transaction to the device bus might take place many cycles after the IDMA access to the Device interface unit completed. There are devices that expect to see the DMAAck* signal asserted along with the actual transaction on the device bus, rather than with the IDMA access to the Device interface unit completion. When setting DMAAckMode bit to '1', DMAAck is asserted with the actual transaction on the device bus. In this case, DMAAck* signal has the same timing characteristics as CSTiming* signal (see Section 7.2 "Device Timing Parameters" on page 141). When setting the DMAAck-Mode bit to '0', DMAAck is asserted for one or two TClk cycle, as soon as the IDMA engine issues the transaction to the target unit. The number of TClk cycles that DMAAck is asserted is dependent on the Channel Control (Low) register's DMAAck_Width bit [4] setting, see Table 437 on page 301.

NOTE: The DMAAckMode is only available for IDMA access to the device bus. Setting this bit to '1' while accessing other interface than the device bus results in no DMAAck* assertion at all.

When using demand mode, the trigger of the channel can be configured to come from the timer rather than from DMAReq* pin. Each of the eight IDMA channels is coupled to one of the eight GT–64240A timers (channel0 to timer0, channel1 to timer1, and so on). Setting TimerReq bit to '1' when channel is configured to demand mode, results in timer trigger rather than DMAReq* trigger. In this case, when the timer/counter reaches the terminal count, an internal IDMA request is set and a new IDMA transfer is initiated.

This mode is useful to generate an IDMA transfer for every 'n' cycle. Set the timer to 'n' cycles, activate it, and then activate the IDMA channel in demand mode with TimerReq bit set. The IDMA engine generates a new burst every 'n' cycles.

NOTE: When running in demand mode and using chain IDMA, when reaching byte count '0', the GT–64240A fetches a new descriptor regardless of the DMAReq*. The DMAReq* affects only IDMA access to data, not to descriptors. This means that chain descriptors must always be ready for fetch.

When running in demand mode, the GT–64240A does not issue a new burst read request from the source before completing the write transaction to the destination.

10.5.7 End Of Transfer

The GT–64240A supports IDMA termination in the middle not only by software, but also by external hardware via EOT pins. Each channel has its own EOT input pin (EOT[0] for channel0, EOT[1] for channel1...). EOT[7:0] pins are multiplexed on MPP pins. To use this feature, the MPP lines must be programmed to act as EOT pins (see Section 19.1 "MPP Multiplexing" on page 480). EOT pins are edge trigger pins.

Setting the EOTEn bit [18] to ‘1’ enables this feature. The affect of EOT assertion can be configured via the EOTMode bit [19].

If the EOTMode bit is set to ‘0’, EOT assertion, when working in chain mode, causes fetching of a new descriptor from memory (if pointer to next descriptor is not equal to NULL) and executing the next IDMA. This is equivalent to executing fetch next descriptor in software.

If the EOTMode bit is set to ‘1’, EOT assertion causes the channel to halt. This is equivalent to setting the Abr bit to ‘1’ via the software.

If the IDMA channel is in non-chain mode, the EOTMode bit is not relevant. EOT assertion causes the current IDMA transfer to be stopped without further action.

A DMA completion interrupt is asserted (if not masked) upon IDMA termination with EOT.

NOTE: The IDMA engine stops only after finishing the current burst. For example, if it is programed to a burst limit of 64 bytes and EOT is sampled active in the middle of the 64 bytes read, the IDMA engine completes the read, performs the 64 byte write, and then halts. When using EOT, the source and destination must be 64-bit aligned.

10.5.8 Descriptor Ownership

A typical application of chain mode IDMA involves the CPU preparing a chain of descriptors in memory and then preparing buffers to move from source to destination. Buffers may be dynamically prepared, this means once a buffer was transferred the CPU can prepare a new buffer in the same location to be sent. This application requires some handshake between the IDMA engine and the CPU.

When working with the new descriptors structure, Bit[31] of the Byte Count register acts as an ownership bit. If set to ‘1’, the descriptor is owned by the GT–64240A IDMA. If set to ‘0’, it is owned by the CPU. Once the CPU prepares a buffer to be transferred, it clears the ownership bit, indicating that the buffer is owned by the IDMA. Once the IDMA completes transferring the buffer, it closes the descriptor by writing back the upper byte of Byte Count register (bits[31:24]), with MSB set to ‘1’, indicating to the CPU the buffer was transferred. When the CPU recognizes that it owns the buffer, it is allowed to place a new buffer to be transferred. An attempt by the IDMA to fetch a descriptor that is owned by CPU (which means CPU did not prepare a new buffer yet), results in an interrupt assertion and an IDMA channel halt.

NOTE: This feature is not supported in compatibility mode.

The Descriptor is closed when the byte count reaches ‘0’ or when transfer is terminated in the middle via EOT or the fetch next descriptor command. In this case, the transfer may end with some data remaining in the buffer pointed by the current descriptor.

When working in compatibility mode, when closing the descriptor, the IDMA engine writes the left byte count to the upper 16-bit of the byte count field of the descriptor. This is useful if an IDMA is terminated in the middle and a CPU might want to re-transmit the left byte count. In case the IDMA ended properly (all byte count was transferred), a ‘0’ value is written back to the descriptor.

When working with the new descriptor structure, there is an alternative way to signal to the CPU that the descriptor was not completely transferred. In this case, the IDMA engine rather than writing back the remaining byte count, it writes back to only bits[31:24] of the descriptor’s ByteCount field, with bit[30] indicating whether the whole byte count was transferred (0) or terminated before transfer completion (1). Bits[29:24] are meaningless.

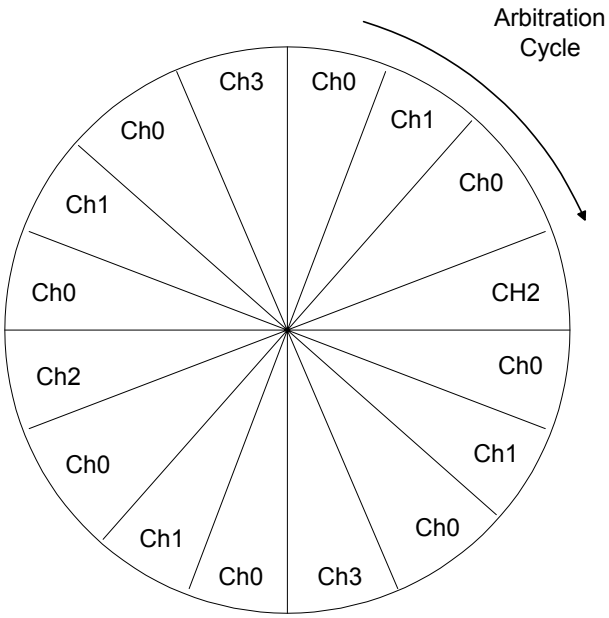
Each IDMA channel has a Current Descriptor Pointer register (CDPTR) associated with it. This register is used for closing the current descriptor before fetching the next descriptor. The register is a read/write register but the

CPU must not write to it. When the NPTR (Next pointer) is first programed, the CDPTR reloads itself with the same value written to NPTR. After processing a descriptor, the IDMA channel updates the current descriptor using CDPTR, saves NPTR into the CDPTR, and fetches a new descriptor.

10.6 Arbitration

The IDMA controller has two programmable round-robin arbiters per the two channels groups. Each channel can be configured to have different bandwidth allocation. Figure 38 shows an example of channels 0-3 arbiter.

Figure 38: Configurable Weights Arbiter



The user can define each of the 16 slices of this “pizza arbiter”. In Figure 38, channel0 gets 50% of the bandwidth, channel1 25%, channel2 and channel3 12.5% each. At each clock cycle, the arbiter samples all channels requests and gives the bus to the next agent according to the “pizza”.

10.7 Big and Little Endian Support

The GT–64240A supports both Little and Big Endian conventions.

The device endianess is determined by the CPU Configuration register’s Endianess bit, see Table 92 on page 88.

The internal registers of the device are always set in Little Endian mode. If the device is configured to Big Endian, descriptors fetched from memory must be converted to Little Endian before being placed in the device registers. The IDMA controller performs this data swapping.

The GT–64240A also supports access to Big and Little Endian devices on the PCI bus. When the IDMA engine is using the CPU address decoding registers, it also uses the CPU interface PCISwap control to determine data swapping on the PCI master interface, see Section 4.11 “CPU Endian Support” on page 71.

When the GT–64240A uses the PCI override feature, it uses the IDMA Control (High) register’s SrcPCISwap, DestPCISwap, and NextPCISwap bits to control the PCI master interface data swapping, see Table 438 on page 304.

10.8 DMA Interrupts

The IDMA interrupts are registered in the IDMA Interrupt Cause registers. There are two registers - one per each four channels. Upon an interrupt event, the corresponding cause bit is set to ‘1’. It is cleared upon a software write of ‘0’.

The IDMA Mask registers controls whether an interrupt event causes an interrupt assertion. The setting of the mask register only affects the interrupt assertion, it has no affect on the cause register bits setting.

The following interrupt events are supported per each channel:

- DMA completion
- DMA address out of range
- DMA access protect violation
- DMA write protect violation
- DMA descriptor ownership violation

In case of an error condition (address out of range, access protect violation, write protect violation, descriptor ownership violation), the IDMA transaction address is latched in the Address Error register. Once an address is latched, no new address (due to additional errors) can be latched, until the current address being read.

NOTE: In any of the error conditions, the DMA completion interrupt bit is set.

10.9 IDMA Registers

Table 369: IDMA Descriptor Register Map

Register	Offset	Page
Channel 0 DMA Byte Count	0x800	page 290
Channel 1 DMA Byte Count	0x804	page 291
Channel 2 DMA Byte Count	0x808	page 291
Channel 3 DMA Byte Count	0x80c	page 291
Channel 4 DMA Byte Count	0x900	page 291
Channel 5 DMA Byte Count	0x904	page 292

Table 369: IDMA Descriptor Register Map (Continued)

Register	Offset	Page
Channel 6 DMA Byte Count	0x908	page 292
Channel 7 DMA Byte Count	0x90c	page 292
Channel 0 DMA Source Address	0x810	page 292
Channel 1 DMA Source Address	0x814	page 292
Channel 2 DMA Source Address	0x818	page 292
Channel 3 DMA Source Address	0x81c	page 292
Channel 4 DMA Source Address	0x910	page 293
Channel 5 DMA Source Address	0x914	page 293
Channel 6 DMA Source Address	0x918	page 293
Channel 7 DMA Source Address	0x91c	page 293
Channel 0 DMA Destination Address	0x820	page 293
Channel 1 DMA Destination Address	0x824	page 293
Channel 2 DMA Destination Address	0x828	page 293
Channel 3 DMA Destination Address	0x82c	page 294
Channel 4 DMA Destination Address	0x920	page 294
Channel 5 DMA Destination Address	0x924	page 294
Channel 6 DMA Destination Address	0x928	page 294
Channel 7 DMA Destination Address	0x92c	page 294
Channel 0 Next Descriptor Pointer	0x830	page 294
Channel 1 Next Descriptor Pointer	0x834	page 294
Channel 2 Next Descriptor Pointer	0x838	page 295
Channel 3 Next Descriptor Pointer	0x83c	page 295
Channel 4 Next Descriptor Pointer	0x930	page 295
Channel 5 Next Descriptor Pointer	0x934	page 295
Channel 6 Next Descriptor Pointer	0x938	page 295
Channel 7 Next Descriptor Pointer	0x93c	page 295
Channel 0 Current Descriptor Pointer	0x870	page 295
Channel 1 Current Descriptor Pointer	0x874	page 296
Channel 2 Current Descriptor Pointer	0x878	page 296
Channel 3 Current Descriptor Pointer	0x87c	page 296

Table 369: IDMA Descriptor Register Map (Continued)

Register	Offset	Page
Channel 4 Current Descriptor Pointer	0x970	page 296
Channel 5 Current Descriptor Pointer	0x974	page 297
Channel 6 Current Descriptor Pointer	0x978	page 297
Channel 7 Current Descriptor Pointer	0x97c	page 297
Channel 0 Source High PCI Address	0x890	page 297
Channel 1 Source High PCI Address	0x894	page 297
Channel 2 Source High PCI Address	0x898	page 297
Channel 3 Source High PCI Address	0x89c	page 298
Channel 4 Source High PCI Address	0x990	page 298
Channel 5 Source High PCI Address	0x994	page 298
Channel 6 Source High PCI Address	0x998	page 298
Channel 7 Source High PCI Address	0x99c	page 298
Channel 0 Destination High PCI Address	0x8a0	page 298
Channel 1 Destination High PCI Address	0x8a4	page 298
Channel 2 Destination High PCI Address	0x8a8	page 299
Channel 3 Destination High PCI Address	0x8ac	page 299
Channel 4 Destination High PCI Address	0x9a0	page 299
Channel 5 Destination High PCI Address	0x9a4	page 299
Channel 6 Destination High PCI Address	0x9a8	page 299
Channel 7 Destination High PCI Address	0x9ac	page 299
Channel 0 Next Descriptor High PCI Address	0x8b0	page 299
Channel 1 Next Descriptor High PCI Address	0x8b4	page 300
Channel 2 Next Descriptor High PCI Address	0x8b8	page 300
Channel 3 Next Descriptor High PCI Address	0x8bc	page 300
Channel 4 Next Descriptor High PCI Address	0x9b0	page 300
Channel 5 Next Descriptor High PCI Address	0x9b4	page 300
Channel 6 Next Descriptor High PCI Address	0x9b8	page 300
Channel 7 Next Descriptor High PCI Address	0x9bc	page 300

Table 370: IDMA Control Register Map

Register	Offset	Page
Channel 0 Control (Low)	0x840	page 301
Channel 0 Control (High)	0x880	page 304
Channel 1 Control (Low)	0x844	page 305
Channel 1 Control (High)	0x884	page 305
Channel 2 Control (Low)	0x848	page 306
Channel 2 Control (High)	0x888	page 306
Channel 3 Control (Low)	0x84c	page 306
Channel 3 Control (High)	0x88c	page 306
Channel 4 Control (Low)	0x940	page 306
Channel 4 Control (High)	0x980	page 306
Channel 5 Control (Low)	0x944	page 306
Channel 5 Control (High)	0x984	page 307
Channel 6 Control (Low)	0x948	page 307
Channel 6 Control (High)	0x988	page 307
Channel 7 Control (Low)	0x94c	page 307
Channel 7 Control (High)	0x98c	page 307
Channels 0-3 Arbiter Control	0x860	page 307
Channels 4-7 Arbiter Control	0x960	page 308
Channels 0-3 Crossbar Timeout	0x8d0	page 308
Channels 4-7 Crossbar Timeout	0x9d0	page 308

Table 371: IDMA Interrupt Register Map

Register	Offset	Page
Channels 0-3 Interrupt Cause	0x8c0	page 309
Channels 0-3 Interrupt Mask	0x8c4	page 309
Channels 0-3 Error Address	0x8c8	page 310
Channels 0-3 Error Select	0x8cc	page 311
Channels 4-7 Interrupt Cause	0x9c0	page 311
Channels 4-7 Interrupt Mask	0x9c4	page 311

Table 371: IDMA Interrupt Register Map (Continued)

Register	Offset	Page
Channels 4-7 Error Address	0x9c8	page 312
Channels 4-7 Error Select	0x9cc	page 312

Table 372: IDMA Debug Register Map
NOTE: Reserved for Galileo Technology usage.

Register	Offset	Page
X0 Address	0x8e0	page 312
X0 Command and ID	0x8e4	page 312
X0 Write Data (Low)	0x8e8	page 312
X0 Write Data (High)	0x8ec	page 312
X0 Write Byte Enables	0x8f8	page 313
X0 Read Data (Low)	0x8f0	page 313
X0 Read Data (High)	0x8f4	page 313
X0 Read ID	0x8fc	page 313
X1 Address	0x9e0	page 313
X1 Command and ID	0x9e4	page 313
X1 Write Data (Low)	0x9e8	page 313
X1 Write Data (High)	0x9ec	page 314
X1 Write Byte Enables	0x9f8	page 314
X1 Read Data (Low)	0x9f0	page 314
X1 Read Data (High)	0x9f4	page 314
X1 Read ID	0x9fc	page 314

10.9.1 IDMA Descriptor Registers

Table 373: Channel 0 DMA Byte Count, Offset: 0x800[1]

Bits	Field Name	Function	Initial Value
23:0	ByteCnt	Number of bytes left for the IDMA to transfer. When running in compatibility mode, the byte count is 16-bit only (bits[15:0]).	0x0
29:24	Reserved	Reserved.	0x0

Table 373: Channel 0 DMA Byte Count, Offset: 0x800[1] (Continued)

Bits	Field Name	Function	Initial Value
30	BCLeft	Left Byte Count When running in non-compatibility mode and when closing a descriptor, indicates whether the whole byte count was completely transferred. 0 - The whole byte count transferred. 1 - Transfer terminated before the whole byte count was transferred.	0x0
31	Own	Ownership Bit When running in non-compatibility mode, this bit indicates whether the descriptor is owned by the IDMA engine (1) or the CPU (0).	0x0

1. When running in compatibility mode and when closing the descriptor, the IDMA writes to bits[31:16] the left byte count to be transferred.

Table 374: Channel 1 DMA Byte Count, Offset: 0x804

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Byte Count.	0x0

Table 375: Channel 2 DMA Byte Count, Offset: 0x808

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Byte Count.	0x0

Table 376: Channel 3 DMA Byte Count, Offset: 0x80c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Byte Count.	0x0

Table 377: Channel 4 DMA Byte Count, Offset: 0x900

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Byte Count.	0x0

Table 378: Channel 5 DMA Byte Count, Offset: 0x904

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Byte Count.	0x0

Table 379: Channel 6 DMA Byte Count, Offset: 0x908

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Byte Count.	0x0

Table 380: Channel 7 DMA Byte Count, Offset: 0x90c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Byte Count.	0x0

Table 381: Channel 0 DMA Source Address, Offset: 0x810

Bits	Field Name	Function	Initial Value
31:0	SrcAdd	Bits[31:0] of the IDMA source address.	0x0

Table 382: Channel 1 DMA Source Address, Offset: 0x814

Bits	Field Name	Function	Initial Value
31:0	SrcAdd	Bits[31:0] of the IDMA source address.	0x0

Table 383: Channel 2 DMA Source Address, Offset: 0x818

Bits	Field Name	Function	Initial Value
31:0	SrcAdd	Bits[31:0] of the IDMA source address.	0x0

Table 384: Channel 3 DMA Source Address, Offset: 0x81c

Bits	Field Name	Function	Initial Value
31:0	SrcAdd	Bits[31:0] of the IDMA source address.	0x0

Table 385: Channel 4 DMA Source Address, Offset: 0x910

Bits	Field Name	Function	Initial Value
31:0	SrcAdd	Bits[31:0] of the IDMA source address.	0x0

Table 386: Channel 5 DMA Source Address, Offset: 0x914

Bits	Field Name	Function	Initial Value
31:0	SrcAdd	Bits[31:0] of the IDMA source address.	0x0

Table 387: Channel 6 DMA Source Address, Offset: 0x918

Bits	Field Name	Function	Initial Value
31:0	SrcAdd	Bits[31:0] of the IDMA source address.	0x0

Table 388: Channel 7 DMA Source Address Offset: 0x91c

Bits	Field Name	Function	Initial Value
31:0	SrcAdd	Bits[31:0] of the IDMA source address.	0x0

Table 389: Channel 0 DMA Destination Address, Offset: 0x820

Bits	Field Name	Function	Initial Value
31:0	DestAdd	Bits[31:0] of the IDMA destination address.	0x0

Table 390: Channel 1 DMA Destination Address, Offset: 0x824

Bits	Field Name	Function	Initial Value
31:0	DestAdd	Bits[31:0] of the IDMA destination address.	0x0

Table 391: Channel 2 DMA Destination Address, Offset: 0x828

Bits	Field Name	Function	Initial Value
31:0	DestAdd	Bits[31:0] of the IDMA destination address.	0x0

Table 392: Channel 3 DMA Destination Address, Offset: 0x82c

Bits	Field Name	Function	Initial Value
31:0	DestAdd	Bits[31:0] of the IDMA destination address.	0x0

Table 393: Channel 4 DMA Destination Address, Offset: 0x920

Bits	Field Name	Function	Initial Value
31:0	DestAdd	Bits[31:0] of the IDMA destination address.	0x0

Table 394: Channel 5 DMA Destination Address, Offset: 0x924

Bits	Field Name	Function	Initial Value
31:0	DestAdd	Bits[31:0] of the IDMA destination address.	0x0

Table 395: Channel 6 DMA Destination Address, Offset: 0x928

Bits	Field Name	Function	Initial Value
31:0	DestAdd	Bits[31:0] of the IDMA destination address.	0x0

Table 396: Channel 7 DMA Destination Address, Offset: 0x92c

Bits	Field Name	Function	Initial Value
31:0	DestAdd	Bits[31:0] of the IDMA destination address.	0x0

Table 397: Channel 0 Next Descriptor Pointer, Offset: 0x830

Bits	Field Name	Function	Initial Value
31:0	NextDescPtr	Bits[31:0] of the DMA next descriptor address. The address must be 32-byte aligned (bits[3:0] must be 0x0).	0x0

Table 398: Channel 1 Next Descriptor Pointer Offset: 0x834

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the channel 0 next descriptor pointer.	0x0

Table 399: Channel 2 Next Descriptor Pointer, Offset: 0x838

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the channel 0 next descriptor pointer.	0x0

Table 400: Channel 3 Next Descriptor Pointer, Offset: 0x83c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the channel 0 next descriptor pointer.	0x0

Table 401: Channel 4 Next Descriptor Pointer, Offset: 0x930

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the channel 0 next descriptor pointer.	0x0

Table 402: Channel 5 Next Descriptor Pointer, Offset: 0x934

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the channel 0 next descriptor pointer.	0x0

Table 403: Channel 6 Next Descriptor Pointer, Offset: 0x938

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the channel 0 next descriptor pointer.	0x0

Table 404: Channel 7 Next Descriptor Pointer, Offset: 0x93c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the channel 0 next descriptor pointer.	0x0

Table 405: Channel 0 Current Descriptor Pointer, Offset: 0x870

Bits	Field Name	Function	Initial Value
31:0	CDPTR0	Bits[31:0] of the address from which the current descriptor was fetched.	0x0

Table 406: Channel 1 Current Descriptor Pointer, Offset: 0x874

Bits	Field Name	Function	Initial Value
31:0	CDPTR1	Bits[31:0] of the address from which the current descriptor was fetched.	0x0

Table 407: Channel 2 Current Descriptor Pointer, Offset: 0x878

Bits	Field Name	Function	Initial Value
31:0	CDPTR2	Bits[31:0] of the address from which the current descriptor was fetched.	0x0

Table 408: Channel 3 Current Descriptor Pointer, Offset: 0x87c

Bits	Field Name	Function	Initial Value
31:0	CDPTR3	Bits[31:0] of the address from which the current descriptor was fetched.	0x0

Table 409: Channel 4 Current Descriptor Pointer, Offset: 0x970

Bits	Field Name	Function	Initial Value
31:0	CDPTR4	Bits[31:0] of the address from which the current descriptor was fetched.	0x0

Table 410: Channel 5 Current Descriptor Pointer, Offset: 0x974

Bits	Field Name	Function	Initial Value
31:0	CDPTR5	Bits[31:0] of the address from which the current descriptor was fetched.	0x0

Table 411: Channel 6 Current Descriptor Pointer, Offset: 0x978

Bits	Field Name	Function	Initial Value
31:0	CDPTR6	Bits[31:0] of the address from which the current descriptor was fetched.	0x0

Table 412: Channel 7 Current Descriptor Pointer, Offset: 0x97c

Bits	Field Name	Function	Initial Value
31:0	CDPTR7	Bits[31:0] of the address from which the current descriptor was fetched.	0x0

Table 413: Channel 0 Source PCI High Address, Offset: 0x890

Bits	Field Name	Function	Initial Value
31:0	SrcHAddr	Bits[63:32] of the PCI source address.	0x0

Table 414: Channel 1 Source PCI High Address, Offset: 0x894

Bits	Field Name	Function	Initial Value
31:0	SrcHAddr	Bits[63:32] of the PCI source address.	0x0

Table 415: Channel 2 Source PCI High Address, Offset: 0x898

Bits	Field Name	Function	Initial Value
31:0	SrcHAddr	Bits[63:32] of the PCI source address.	0x0

Table 416: Channel 3 Source PCI High Address, Offset: 0x89c

Bits	Field Name	Function	Initial Value
31:0	SrcHAddr	Bits[63:32] of the PCI source address.	0x0

Table 417: Channel 4 Source PCI High Address, Offset: 0x990

Bits	Field Name	Function	Initial Value
31:0	SrcHAddr	Bits[63:32] of the PCI source address.	0x0

Table 418: Channel 5 Source PCI High Address, Offset: 0x994

Bits	Field Name	Function	Initial Value
31:0	SrcHAddr	Bits[63:32] of the PCI source address.	0x0

Table 419: Channel 6 Source PCI High Address, Offset: 0x998

Bits	Field Name	Function	Initial Value
31:0	SrcHAddr	Bits[63:32] of the PCI source address.	0x0

Table 420: Channel 7 Source PCI High Address, Offset: 0x99c

Bits	Field Name	Function	Initial Value
31:0	SrcHAddr	Bits[63:32] of the PCI source address.	0x0

Table 421: Channel 0 Destination PCI High Address, Offset: 0x8a0

Bits	Field Name	Function	Initial Value
31:0	DestHAddr	Bits[63:32] of the PCI destination address.	0x0

Table 422: Channel 1 Destination PCI High Address, Offset: 0x8a4

Bits	Field Name	Function	Initial Value
31:0	DestHAddr	Bits[63:32] of the PCI destination address.	0x0

Table 423: Channel 2 Destination PCI High Address, Offset: 0x8a8

Bits	Field Name	Function	Initial Value
31:0	DestHAddr	Bits[63:32] of the PCI destination address.	0x0

Table 424: Channel 3 Destination PCI High Address, Offset: 0x8ac

Bits	Field Name	Function	Initial Value
31:0	DestHAddr	Bits[63:32] of the PCI destination address.	0x0

Table 425: Channel 4 Destination PCI High Address, Offset: 0x9a0

Bits	Field Name	Function	Initial Value
31:0	DestHAddr	Bits[63:32] of the PCI destination address.	0x0

Table 426: Channel 5 Destination PCI High Address, Offset: 0x9a4

Bits	Field Name	Function	Initial Value
31:0	DestHAddr	Bits[63:32] of the PCI destination address.	0x0

Table 427: Channel 6 Destination PCI High Address, Offset: 0x9a8

Bits	Field Name	Function	Initial Value
31:0	DestHAddr	Bits[63:32] of the PCI destination address.	0x0

Table 428: Channel 7 Destination PCI High Address, Offset: 0x9ac

Bits	Field Name	Function	Initial Value
31:0	DestHAddr	Bits[63:32] of the PCI destination address.	0x0

Table 429: Channel 0 Next Descriptor PCI High Address, Offset: 0x8b0

Bits	Field Name	Function	Initial Value
31:0	NextHAddr	Bits[63:32] of the PCI destination address.	0x0

Table 430: Channel 1 Next Descriptor PCI High Address, Offset: 0x8b4

Bits	Field Name	Function	Initial Value
31:0	NextHAddr	Bits[63:32] of the PCI destination address.	0x0

Table 431: Channel 2 Next Descriptor PCI High Address, Offset: 0x8b8

Bits	Field Name	Function	Initial Value
31:0	NextHAddr	Bits[63:32] of the PCI destination address.	0x0

Table 432: Channel 3 Next Descriptor PCI High Address, Offset: 0x8bc

Bits	Field Name	Function	Initial Value
31:0	NextHAddr	Bits[63:32] of the PCI next descriptor address.	0x0

Table 433: Channel 4 Next Descriptor PCI High Address, Offset: 0x9b0

Bits	Field Name	Function	Initial Value
31:0	NextHAddr	Bits[63:32] of the PCI next descriptor address.	0x0

Table 434: Channel 5 Next Descriptor PCI High Address, Offset: 0x9b4

Bits	Field Name	Function	Initial Value
31:0	NextHAddr	Bits[63:32] of the PCI next descriptor address.	0x0

Table 435: Channel 6 Next Descriptor PCI High Address, Offset: 0x9b8

Bits	Field Name	Function	Initial Value
31:0	NextHAddr	Bits[63:32] of the PCI next descriptor address.	0x0

Table 436: Channel 7 Next Descriptor PCI High Address, Offset: 0x9bc

Bits	Field Name	Function	Initial Value
31:0	NextHAddr	Bits[63:32] of the PCI next descriptor address.	0x0

10.9.2 IDMA Channel Control Registers

Table 437: Channel 0 Control (Low), Offset: 0x840

Bits	Field name	Function	Initial Value
2:0	DstBurstLimit	101 - Reserved 110 - Reserved 010 - Reserved 000 - 8 Bytes 001 - 16 Bytes 011 - 32 Bytes 111 - 64 Bytes 100 - 128 Bytes	0x0
3	SrcHold	Source Hold 0 - Increment source address. 1 - Hold in the same value.	0x0
4	DMAAck_Width	0 - Asserted for one TClk cycle (default setting) 1 - Asserted for two TClk cycles	0x0
5	DestHold	Destination Hold 0 - Increment destination address. 1 - Hold in the same value.	0x0
8:6	BurstLimit	Burst Limit in Each IDMA Access 101 - Reserved 110 - Reserved 010 - Reserved 000 - 8 Bytes 001 - 16 Bytes 011 - 32 Bytes 111 - 64 Bytes 100 - 128 Bytes **NOTE:** If BLMode is set to '1', BurstLimit acts as the source burst limit.	0x0
9	ChainMode	Chained Mode 0 - Chained mode 1 - Non-Chained mode	0x0
10	IntMode	Interrupt Mode 0 - Interrupt asserted every time the IDMA byte count reaches '0'. 1 - Interrupt asserted when the Next Descriptor pointer is NULL and the IDMA byte count reaches '0'. **NOTE:** IntMode is only relevant in chain mode.	0x0

Table 437: Channel 0 Control (Low), Offset: 0x840 (Continued)

Bits	Field name	Function	Initial Value
11	DemandMode	Demand Mode Enable 0 - Demand mode 1 - Block mode	0x0
12	ChanEn	Channel Enable When the software sets this bit to '1', it activates the channel. Setting this bit to '0' causes the channel to suspend. Re-setting the bit to '1', allows the channel to continue the IDMA transfer.	0x0
13	FetchND	Fetch Next Descriptor If set to '1', forces a fetch of the next descriptor. Cleared after the fetch is completed. **NOTE:** FetchND is only relevant in chain mode	0x0
14	ChanAct	DMA Channel Active Read only. 0 - Channel is not active. 1 - Channel is active.	0x0
15	DMAReqDir	DMAReq Direction 0 - DMAReq* generated by the source. 1 - DMAReq* generated by the destination.	0x0
16	DMAReqMode	DMAReq* Mode 0 - DMAReq* is level input. 1 - DMAReq* is edge triggered input.	0x0
17	CDEn	Close Descriptor Enable If enabled, the IDMA writes the upper byte(s) of the byte count field back to memory. In compatibility mode, it writes the remainder byte count into bits[31:16] of the byte count field. In non-compatibility mode, it writes the ownership and status bits into bits[31:24] of byte count field. 0 - Disable 1 - Enable	0x0
18	EOTEn	End Of Transfer Enable If enabled, an IDMA transfer can be stopped in the middle of the transfer using EOT signal. 0 - Disable 1 - Enable	0x0
19	EOTMode	End of Transfer Affect 0 - Fetch next descriptor 1 - Channel halt	0x0

Table 437: Channel 0 Control (Low), Offset: 0x840 (Continued)

Bits	Field name	Function	Initial Value
20	Abr	Channel Abort When the software sets this bit to '1', the IDMA aborts in the middle. The bit is cleared by the IDMA hardware.	0x0
22:21	SAddrOvr	Override Source Address 00 - No override. 01 - Source address is in PCI_0 memory space 10 - Source address is in PCI_1 memory space 11 - Reserved	0x0
24:23	DAddrOvr	Override Destination Address 00 - No override. 01 - Destination address is in PCI_0 memory space 10 - Destination address is in PCI_1 memory space 11 - Reserved	0x0
26:25	NAddrOvr	Override Next Descriptor Address 00 - No override 01 - Descriptor address is in PCI_0 memory space 10 - Descriptor address is in PCI_1 memory space 11 - Reserved	0x0
27	DMAAckMode	DMA Acknowledge Mode 0 - Asserted for one TClk when the IDMA engine issues the transaction. 1 - Asserted only with the actual transaction driven on the device bus (same timing as CSTiming signal).	0x0
28	TimerReq	Timer IDMA Request Enable 0 - IDMA requests taken from the DMAReq* pin. 1 - IDMA requests taken from the timer/counter.	0x0
30:29	DMAAckDir	DMA Acknowledge Direction 00 - Reserved 01 - Asserted with accesses to destination. 10 - Asserted with accesses to source. 11 - Reserved.	0x0
31	DescMode	Descriptor Mode 0 - Compatibility mode 1 - New descriptor structure	0x0

Table 438: Channel 0 Control (High), Offset: 0x880
NOTE: Program the High Control register prior to channel activation.

Bits	Field Name	Function	Initial Value
3:0	SrcHAddr	Bits[35:32] of the source address.	0x0
5:4	SrcPCISwap	PCI Master Data Swap Control 00 - Byte Swap 01 - No swapping 10 - Both byte and word swap 11 - Word swap SrcPCISwap is applicable only when using SAddrOvr. Otherwise, the PCI master data swapping is controlled via the PCI Memory Low Decode register's PCISwap field, see Table 55 on page 80.	0x1
6	Reserved	Reserved	0x0
7	SrcPCIReq64	PCI Master REQ64* Policy 0 - Only Asserts REQ64* when a read from the source is longer than 64-bits. 1 - Always assert REQ64*. SrcPCIReq64 is only applicable when using SAddrOvr. Otherwise, the PCI master REQ64* policy is controlled via the PCI Memory Low Decode register's PCIReq64 bit, see Table 55 on page 80.	0x0
11:8	DestHAddr	Bits[35:32] of Destination Address	0x0
13:12	DestPCISwap	PCI Master Data Swap Control 00 - Byte Swap 01 - No swapping 10 - Both byte and word swap 11 - Word swap DestPCISwap is only applicable when using DAddrOvr. Otherwise, the PCI master data swapping is controlled via the PCI Memory Low Decode register's PCISwap field, see Table 55 on page 80.	0x1
14	Reserved	Reserved	0x0
15	DestPCIReq64	PCI Master REQ64* Policy 0 - Only asserts REQ64* when the write to a destination is longer than 64-bits. 1 - Always assert REQ64*. DestPCIReq64 is only applicable when using DAddrOvr. Otherwise, the PCI master REQ64* policy is controlled via the PCI Memory Low Decode register's PCIReq64 bit, see Table 55 on page 80.	0x0

Table 438: Channel 0 Control (High), Offset: 0x880 (Continued)
NOTE: Program the High Control register prior to channel activation.

Bits	Field Name	Function	Initial Value
19:16	NextHAddr	Bits[35:32] of the next descriptor address.	0x0
21:20	NextPCISwap	PCI Master Data Swap Control 00 - Byte Swap 01 - No swapping 10 - Both byte and word swap 11 - Word swap NextPCISwap is only applicable when using NAddrOvr. Otherwise, the PCI master data swapping is controlled via the PCI Memory Low Decode register's PCISwap field, see Table 55 on page 80.	0x1
22	Reserved	Reserved	0x0
23	NextPCIReq64	PCI Master REQ64* Policy 0 - Only asserts REQ64* when the read of the next descriptor is longer than 64-bits. 1 - Always assert REQ64* NextPCIReq64 is only applicable when using NAddrOvr. Otherwise, the PCI master REQ64* policy is controlled via PCI Memory Low Decode register's PCIReq64 bit, see Table 55 on page 80.	0x0
24	SrcHWSwap	PCI Master Half-Word Swap Enable	0x0
25	DestHWSwap	PCI Master Half-Word Swap Enable	0x0
26	NextHWSwap	PCI Master Half-Word Swap Enable	0x0
30:24	Reserved	Must be 0.	0x0
31	BLMode	Burst Limit Mode 0 - Same burst limit for source and destination. 1 - Different burst limit for source and destination	0x0

Table 439: Channel 1 Control (Low), Offset: 0x844

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (Low).	0x0

Table 440: Channel 1 Control (High), Offset: 0x884

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (High).	0x101010

Table 441: Channel 2 Control (Low), Offset: 0x848

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (Low).	0x0

Table 442: Channel 2 Control (High), Offset: 0x888

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (High).	0x101010

Table 443: Channel 3 Control (Low), Offset: 0x84c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (Low).	0x0

Table 444: Channel 3 Control (High), Offset: 0x88c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (High).	0x101010

Table 445: Channel 4 Control (Low), Offset: 0x940

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (Low).	0x0

Table 446: Channel 4 Control (High), Offset: 0x980

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (High).	0x101010

Table 447: Channel 5 Control (Low), Offset: 0x944

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (Low).	0x0

Table 448: Channel 5 Control (High), Offset: 0x984

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (High).	0x101010

Table 449: Channel 6 Control (Low), Offset: 0x948

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (Low).	0x0

Table 450: Channel 6 Control (High), Offset: 0x988

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (High).	0x101010

Table 451: Channel 7 Control (Low), Offset: 0x94c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (Low).	0x0

Table 452: Channel 7 Control (High), Offset: 0x98c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (High).	0x101010

Table 453: Channels 0-3 Arbiter Control, Offset: 0x860

Bits	Field name	Function	Initial Value
1:0	Arb0	Slice 0 of “pizza arbiter”. 00 - Channel0 01 - Channel1 10 - Channel2 11 - Channel3	0x0
3:2	Arb1	Slice 1 of “pizza arbiter”.	0x1
5:4	Arb2	Slice 2 of “pizza arbiter”.	0x2
7:6	Arb3	Slice 3 of “pizza arbiter”.	0x3
9:8	Arb4	Slice 4 of “pizza arbiter”.	0x0

Table 453: Channels 0-3 Arbiter Control, Offset: 0x860 (Continued)

Bits	Field name	Function	Initial Value
11:10	Arb5	Slice 5 of “pizza arbiter”.	0x1
13:12	Arb6	Slice 6 of “pizza arbiter”.	0x2
15:14	Arb7	Slice 7 of “pizza arbiter”.	0x3
17:16	Arb8	Slice 8 of “pizza arbiter”.	0x0
19:18	Arb9	Slice 9 of “pizza arbiter”.	0x1
21:20	Arb10	Slice 10 of “pizza arbiter”.	0x2
23:22	Arb11	Slice 11 of “pizza arbiter”.	0x3
25:24	Arb12	Slice 12 of “pizza arbiter”.	0x0
27:26	Arb13	Slice 13 of “pizza arbiter”.	0x1
29:28	Arb14	Slice 14 of “pizza arbiter”.	0x2
31:30	Arb15	Slice 15 of “pizza arbiter”.	0x3

Table 454: Channels 4-7 Arbiter Control, Offset: 0x960

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0-3 Arbiter Control.	0xe4e4e4e4

Table 455: Channels 0-3 Crossbar Timeout, Offset: 0x8d0
NOTE: Reserved for Galileo Technology usage.

Bits	Field Name	Function	Initial Value
7:0	Timeout	CrossBar Arbiter Timeout Preset Value	0xff
15:8	Reserved	Reserved.	0x0
16	TimeoutEn	CrossBar Arbiter Timer Enable 0 - Enable 1 - Disable	0x1
31:17	Reserved	Reserved.	0x0

Table 456: Channels 4-7 Crossbar Timeout, Offset: 0x9d0
NOTE: Reserved for Galileo Technology usage.

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0-3 Crossbar Timeout.	0x100ff

10.9.3 IDMA Interrupt Registers

Table 457: Channels 0-3 Interrupt Cause, Offset: 0x8c0

NOTE: All cause bits are clear only. They are set to '1' upon an interrupt event and cleared when the software writes a value of '0'. Writing '1' has no affect.

Bits	Field name	Function	Initial Value
0	Comp	Channel0 IDMA Completion.	0x0
1	AddrMiss	Channel0 Address Miss Failed address decoding.	0x0
2	AccProt	Channel0 Access Protect Violation	0x0
3	WrProt	Channel0 Write Protect	0x0
4	Own	Channel0 Descriptor Ownership Violation Attempt to access the descriptor owned by the CPU.	0x0
7:5	Reserved	Reserved.	0x0
12:8	Various	Same as channel0 cause bits.	0x0
15:13	Reserved	Reserved	0x0
22:16	Various	Same as channel0 cause bits.	0x0
23:21	Reserved	Reserved.	0x0
28:24	Various	Same as channel0 cause bits.	0x0
31:29	Reserved	Reserved.	0x0

Table 458: Channels 0-3 Interrupt Mask, Offset: 0x8c4

Bits	Field Name	Function	Initial Value
0	Comp	If set to '1', Comp interrupt is enabled.	0x0
1	AddrMiss	If set to '1', AddrMiss interrupt is enabled.	0x0
2	AccProt	If set to '1', AccProt interrupt is enabled.	0x0
3	WrProt	If set to '1', WrProt interrupt is enabled.	0x0
4	Own	If set to '1', Own interrupt is enabled.	0x0
5	EOT	If set to '1', EOT interrupt is enabled.	0x0
7:6	Reserved	Reserved.	0x0
13:8	Various	Same as channel0 mask bits.	0x0
15:14	Reserved	Reserved.	0x0
21:16	Various	Same as channel0 mask bits.	0x0
23:22	Reserved	Reserved.	0x0

Table 458: Channels 0-3 Interrupt Mask, Offset: 0x8c4 (Continued)

Bits	Field Name	Function	Initial Value
29:24	Various	Same as channel0 mask bits.	0x0
31:30	Reserved	Reserved.	0x0

Table 459: Channels 0-3 Error Address, Offset: 0x8c8

Bits	Field Name	Function	Initial Value
31:0	ErrAddr	Bits[31:0] of Error Address Latched upon any of the error events interrupts (address miss, access protection, write protection, ownership violation). Once the address is latched, no new address is latched until the register is read.	0x0

Table 460: Channels 0-3 Error Select, Offset: 0x8cc

Bits	Field Name	Function	Initial Value
4:0	Sel	Specifies the error event currently reported in the Error Address register. 0x0 - Comp Channel 0 0x1 - AddrMiss Channel 0 0x2 - AccProt Channel 0 0x3 - WrProt Channel 0 0x4 - Own Channel 0 0x5 - 0x7 - Reserved 0x8 - Comp Channel 1 0x9 - AddrMiss Channel 1 0xa - AccProt Channel 1 0xb - WrProt Channel 1 0xc - Own Channel 1 0xd - EOT Channel 1 0xe - 0xf - Reserved 0x10 - Comp Channel 2 0x11 - AddrMiss Channel 2 0x12 - AccProt Channel 2 0x13 - WrProt Channel 2 0x14 - Own Channel 2 0x16 - 0x17 - Reserved 0x18 - Comp Channel 3 0x19 - AddrMiss Channel 3 0x1a - AccProt Channel 3 0x1b - WrProt Channel 3 0x1c - Own Channel 3 0x1d - 0x1f - Reserved Read Only.	0x0
31:5	Reserved	Reserved.	0x0

Table 461: Channels 4-7 Interrupt Cause, Offset: 0x9c0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0-3 Interrupt Cause.	0x0

Table 462: Channels 4-7 Interrupt Mask, Offset: 0x9c4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0-3 Interrupt Mask.	0x0

Table 463: Channels 4-7 Error Address, Offset: 0x9c8

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0-3 error address.	0x0

Table 464: Channels 4-7 Error Select, Offset: 0x9cc

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0-3 error select.	0x0

10.9.4 IDMA Debug Registers

NOTE: Reserved for Galileo Technology usage.

Table 465: X0 Address, Offset: 0x8e0

Bits	Field Name	Function	Initial Value
31:0	Addr	a2x0_ad[31:0] registered on (a2x0_req & x02a_ack)	0x0

Table 466: X0 Command and ID, Offset: 0x8e4

Bits	Field Name	Function	Initial Value
19:0	Cmd	a2x0_cbe[19:0] registered on (a2x0_req & x02a_ack)	0x0
31:20	ID	a2x0_id[11:0] registered on (a2x0_req & x02a_ack)	0x0

Table 467: X0 Write Data (Low), Offset: 0x8e8

Bits	Field Name	Function	Initial Value
31:0	Data	a2x0_ad[31:0] registered on a2x0_valid	0x0

Table 468: X0 Write Data (High), Offset: 0x8ec

Bits	Field Name	Function	Initial Value
31:0	Data	a2x0_ad[63:32] registered on a2x0_valid	0x0

Table 469: X0 Write Byte Enables, Offset: 0x8f8

Bits	Field Name	Function	Initial Value
7:0	BE	a2x0_cbe registered on a2x0_valid	0x0
31:8	Reserved	Reserved.	0x0

Table 470: X0 Read Data (Low), Offset: 0x8f0

Bits	Field Name	Function	Initial Value
31:0	Data	x02a_ad[31:0] registered on x02a_rd_valid	0x0

Table 471: X0 Read Data (High), Offset: 0x8f4

Bits	Field Name	Function	Initial Value
31:0	Data	x02a_ad[63:32] registered on x02a_rd_valid	0x0

Table 472: X0 Read ID, Offset: 0x8fc

Bits	Field Name	Function	Initial Value
11:0	ID	x02a_id[11:0] registered on x02a_rd_valid	0x0
31:12	Reserved	Reserved.	0x0

Table 473: X1 Address, Offset: 0x9e0

Bits	Field Name	Function	Initial Value
31:0	Addr	a2x1_ad[31:0] registered on (a2x1_req & x12a_ack)	0x0

Table 474: X1 Command and ID, Offset: 0x9e4

Bits	Field Name	Function	Initial Value
19:0	Cmd	a2x1_cbe[19:0] registered on (a2x1_req & x12a_ack)	0x0
31:20	ID	a2x1_id[11:0] registered on (a2x1_req & x12a_ack)	0x0

Table 475: X1 Write Data (Low), Offset: 0x9e8

Bits	Field Name	Function	Initial Value
31:0	Data	a2x1_ad[31:0] registered on a2x1_valid	0x0

Table 476: X1Write Data (High), Offset: 0x9ec

Bits	Field Name	Function	Initial Value
31:0	Data	a2x1_ad[63:32] registered on a2x1_valid	0x0

Table 477: X1 Write Byte Enables, Offset: 0x9f8

Bits	Field Name	Function	Initial Value
7:0	BE	a2x1_cbe registered on a2x1_valid	0x0
31:8	Reserved	Reserved.	0x0

Table 478: X1 Read Data (Low), Offset: 0x9f0

Bits	Field Name	Function	Initial Value
31:0	Data	x12a_ad[31:0] registered on x12a_rd_valid	0x0

Table 479: X1 Read Data (High), Offset: 0x9f4

Bits	Field Name	Function	Initial Value
31:0	Data	x12a_ad[63:32] registered on x12a_rd_valid	0x0

Table 480: X1 Read ID, Offset: 0x9fc

Bits	Field Name	Function	Initial Value
11:0	ID	x12a_id[11:0] registered on x12a_rd_valid	0x0
31:12	Reserved	Reserved.	0x0

11. TIMER/COUNTERS

There are eight 32-bit wide timer/counters on the GT–64240A. Each timer/counter can be selected to operate as a timer or as a counter.

Each timer/counter decrements with every Tclk rising edge.

In Counter mode, the counter counts down to terminal count, stops, and issues an interrupt.

In Timer mode, the timer counts down, issues an interrupt on terminal count, reloads itself to the programmed value, and continues to count.

Reads from the counter or timer are done from the counter itself, while writes are to its register. This means that read results are in the counter's real time value.

Each timer/counter can be configured to have an external count enable input, through one of the MPP pins. In this configuration, the counter counts down as long as the count enable pin is active low.

Each timer/counter has a TCTcnt output pin. This pin is asserted when the counter reaches zero. It is also muxed on the MPP pins. The Timer/Counter 0-3 Control register's TCnt_Width bits (see Table 490 on page 317) determine if TcTcnt is asserted for one or two Tclk cycles.

If a wider timer is required, cascade two timers to generate a 64-bit timer. Cascade the timers by connecting the first timer's TCTcnt output to the second timer's TCEn input. With this configuration, each time the first counter reaches terminal count the second counter decrements by one.

NOTE: If using an external count enable input, it is necessary to configure the appropriate MPP pin prior to counter activation.

TCTcnt is asserted one TClk cycle after the counter reaches zero.

MPP pins can also be configured to act as timer/counter terminal count output pins. In this configuration, the corresponding MPP pin is asserted low whenever the timer/counter reaches terminal count.

11.1 Timers/Counters Registers

Table 481: IDMA Descriptor Register Map

Register	Offset	Page
Timer/Counter 0	0x850	page 316
Timer/Counter 1	0x854	page 316
Timer/Counter 2	0x858	page 316
Timer/Counter 3	0x85c	page 316
Timer/Counter 4	0x950	page 316
Timer/Counter 5	0x954	page 317
Timer/Counter 6	0x958	page 317
Timer/Counter 7	0x95c	page 317

Table 481: IDMA Descriptor Register Map (Continued)

Register	Offset	Page
Timer/Counter 0-3 Control	0x864	page 317
Timer/Counter 0-3 Interrupt Cause	0x868	page 319
Timer/Counter 0-3 Interrupt Mask	0x86c	page 319
Timer/Counter 4-7 Control	0x964	page 320
Timer/Counter 4-7 Interrupt Cause	0x968	page 320
Timer/Counter 4-7 Interrupt Mask	0x96c	page 320

Table 482: Timer/Counter 0, Offset: 0x850

Bits	Field Name	Function	Initial Value
31:0	TC0	Timer/Counter 0 Value	0x0

Table 483: Timer/Counter 1, Offset: 0x854

Bits	Field Name	Function	Initial Value
31:0	TC1	Timer/Counter 1 value.	0x0

Table 484: Timer/Counter 2, Offset: 0x858

Bits	Field Name	Function	Initial Value
31:0	TC2	Timer/Counter 2 value.	0x0

Table 485: Timer/Counter 3, Offset: 0x85c

Bits	Field Name	Function	Initial Value
31:0	TC3	Timer/Counter 3 value.	0x0

Table 486: Timer/Counter 4, Offset: 0x950

Bits	Field Name	Function	Initial Value
31:0	TC4	Timer/Counter 4 value.	0x0

Table 487: Timer/Counter 5, Offset: 0x954

Bits	Field Name	Function	Initial Value
31:0	TC5	Timer/Counter 5 value.	0x0

Table 488: Timer/Counter 6, Offset: 0x958

Bits	Field Name	Function	Initial Value
31:0	TC6	Timer/Counter 6 value.	0x0

Table 489: Timer/Counter 7, Offset: 0x95c

Bits	Field Name	Function	Initial Value
31:0	TC7	Timer/Counter 7 value.	0x0

Table 490: Timer/Counter 0-3 Control, Offset: 0x864

Bits	Field name	Function	Initial Value
0	TC0En	Timer/Counter Enable 0 - Disable 1 - Enable **NOTE:** When configured to counter, new count starts only with new write of '1' to the TcEn bit. In timer mode, the count continues as long as TcEn is set to '1'. Counting starts two cycles after TCEn assertion.	0x0
1	TC0Mode	Timer/Counter Mode 0 - Counter 1 - Timer	0x0
2	TC0Trig	Timer/Counter Trigger 0 - No external trigger Starts counting as soon as TC0En is set to '1'. 1 - External trigger. Starts counting as soon as TC0En is set to '1' AND the external TC0En input is asserted.	0x0
3	TCnt0_Width	0 - TCTcnt asserted for one Tclk cycle. 1 - TCTcnt asserted for two Tclk cycles.	0x0
7:4	Reserved	Reserved.	0x0

Table 490: Timer/Counter 0-3 Control, Offset: 0x864 (Continued)

Bits	Field name	Function	Initial Value
8	TC1En	Timer/Counter Enable 0 - Disable 1 - Enable **NOTE:** When configured to counter, new count starts only with new write of '1' to the TcEn bit. In timer mode, the count continues as long as TcEn is set to '1'. Counting starts two cycles after TCEn assertion.	0x0
9	TC1Mode	Timer/Counter Mode 0 - Counter 1 - Timer	0x0
10	TC1Trig	Timer/Counter Trigger 0 - No external trigger Starts counting as soon as TC1En is set to '1'. 1 - External trigger Starts counting as soon as TC1En is set to '1' AND the external TC1En input is asserted.	0x0
11	TCnt1_Width	0 - TCTcnt asserted for one TClk cycle. 1 - TCTcnt asserted for two TClk cycles.	0x0
15:12	Reserved	Reserved.	0x0
16	TC2En	Timer/Counter Enable 0 - Disable 1 - Enable **NOTE:** When configured to counter, new count starts only with new write of '1' to the TcEn bit. In timer mode, the count continues as long as TcEn is set to '1'. Counting starts two cycles after TCEn assertion.	0x0
17	TC2Mode	Timer/Counter Mode 0 - Counter 1 - Timer	0x0
18	TC2Trig	Timer/Counter Trigger 0 - No external trigger. Starts counting as soon as TC2En is set to '1'. 1 - External trigger. Starts counting as soon as TC2En is set to '1' AND the external TC2En input is asserted.	0x0

Table 490: Timer/Counter 0-3 Control, Offset: 0x864 (Continued)

Bits	Field name	Function	Initial Value
19	TCnt2_Width	0 - TCTcnt asserted for one Tclk cycle. 1 - TCTcnt asserted for two Tclk cycles.	0x0
23:20	Reserved	Reserved.	0x0
24	TC3En	Timer/Counter Enable 0 - Disable 1 - Enable **NOTE:** When configured to counter, new count starts only with new write of '1' to the TcEn bit. In timer mode, the count continues as long as TcEn is set to '1'. Counting starts two cycles after TCEn assertion.	0x0
25	TC3Mode	Timer/Counter Mode 0 - Counter 1 - Timer	0x0
26	TC3Trig	Timer/Counter Trigger 0 - No external trigger Starts counting as soon as TC3En is set to '1'. 1 - External trigger Starts counting as soon as TC3En is set to '1' AND external TC3En input is asserted.	0x0
27	TCnt3_Width	0 - TCTcnt asserted for one Tclk cycle. 1 - TCTcnt asserted for two Tclk cycles.	0x0
31:28	Reserved	Reserved	0x0

Table 491: Timer/Counter 0-3 Interrupt Cause, Offset: 0x868

NOTE: All cause bits are clear only. They are set to '1' upon timer terminal count. They are cleared by writing a value of '0'. Writing a value of '1' has no affect.

Bits	Field Name	Function	Initial Value
0	TC0	Timer/Counter 0 terminal count.	0x0
1	TC1	Timer/Counter 1 terminal count.	0x0
2	TC2	Timer/Counter 2 terminal count.	0x0
3	TC3	Timer/Counter 3 terminal count.	0x0
30:4	Reserved	Reserved.	0x0
31	Sum	Summary of all cause bits. Read Only	0x0

Table 492: Timer/Counter 0-3 Interrupt Mask, Offset: 0x86c

Bits	Field Name	Function	Initial Value
0	TC0	If set to ‘1’, TC0 interrupt is enabled.	0x0
1	TC1	If set to ‘1’, TC1 interrupt is enabled.	0x0
2	TC2	If set to ‘1’, TC2 interrupt is enabled.	0x0
3	TC3	If set to ‘1’, TC3 interrupt is enabled.	0x0
31:4	Reserved	Reserved.	0x0

Table 493: Timer/Counter 4-7 Control, Offset: 0x964

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Timer/Counter 0-3 Control.	0x0

Table 494: Timer/Counter 4-7 Interrupt Cause, Offset: 0x968

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Timer/Counter 0-3 Interrupt Cause.	0x0

Table 495: Timer/Counter 4-7 Interrupt Mask, Offset: 0x96c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Timer/Counter 0-3 Interrupt Mask.	0x0

12. COMMUNICATION UNIT

The GT–64240A's integrates the following into its communication unit:

- Three ethernet controllers
- Two MPSC controllers
- SDMAs
- Baude rate generators
- An I^2C interface

NOTE: It is only possible to use all three Ethernet controllers when configured to RMII, see Section 19. "Pins Multiplexing" on page 480.

The communication unit acts as a master or slave interface:

- Slave interface: CPU or PCI access to its internal registers.
- Master interface: Communication controller's SDMAs access to the memory or PCI bus.

12.1 Address Decoding

With each communication controllers transactions, the address (buffer, descriptor, or hash table address) is first compared against the CPU interface address decoding registers to select the target interface (SDRAM, Device, PCI bus). The address decoding process is similar to the CPU address decoding, see Section 3.1 "CPU Address Decoding" on page 44).

NOTE: CPU addresses is 36-bit wide. The upper four address bits are programed in each port Address Control register. These four upper bits are fixed for the whole SDMA operation. The communication controller buffers and descriptors addresses are restricted to not cross the 4Gbyte (32-bit address) boundary.

If the address does not match any of the address windows, an interrupt is generated. In case of a descriptor fetch, the SDMA engine also halts.

Cases may occur when the buffers or descriptors are located on a device on the PCI bus that is not mapped by the CPU interface PCI windows. In this case, use the PCI override feature. Buffer, descriptor, or hash table address of each controller can be marked as PCI override. This means the controller accesses the PCI interface directly without executing address decoding.

The PCI interface supports 64-bit addressing. Each communication controller can generate a 64-bit address to the PCI interface via the PCI High Address register. If the PCI High Address register value is '0', the PCI master issues a SAC transaction. If it is not '0' (which means the address is beyond 4Gbyte space), the PCI master generates a DAC transaction.

NOTES: There is no Communication Unit address remapping to the PCI. It is not required due to the PCI override feature.

The Communication Unit always uses its own PCI High Address registers, even if not using PCI override.

12.2 Access Protection

Each communication controller transaction address is also checked against the CPU Interface Access Protect registers. If the address matches one of those regions and the transaction violates the region protection, an interrupt is asserted. Also, in case of the violation occurring during a descriptor fetch, the SDMA halts. For full details, see Section 4.2 “CPU Access Protection” on page 57.

NOTE: The communication controller access protection covers write protect and access protect. Unlike the CPU, there is no caching protection (it is meaningless in the case of comm port access).

12.3 Big and Little Endian Support

The GT–64240A supports both Little and Big Endian conventions. The device endianess is determined via CPU Configuration register’s Endianess bit [12], see Table 92 on page 88.

Since the internal registers of the device are always set in Little Endian mode, if the device is configured to Big Endian, the descriptors fetched from memory must be converted to Little Endian before being placed in the device registers. Each communication controller performs this data swapping. Also, each of the controllers must be programmed to treat the transmit/receive data as Little or Big Endian. See Section 13. “10/100Mb Ethernet Unit” on page 339 and Section 15. “MPSC Serial DMAs (SDMA)” on page 457 sections for more details.

In addition, the GT–64240A supports access to Little and Big Endian devices on the PCI bus. When the comm port uses the CPU address decoding registers, it also uses the CPU interface PCISwap control to determine data swapping on the PCI master interface (see Section 4.11 “CPU Endian Support” on page 71). When the comm port uses the PCI override feature, it uses the following bits in the port’s Address Control (Low) register (Table 497 on page 326) to control the PCI master interface data swapping:

- RxBPCISwap
- RxDPCISwap
- TxBPCISwap
- TxDPCISwap
- HashPCISwap

12.4 Arbitration

The different communication controllers share the same data path to the other GT–64240A interfaces. Arbitration over this data path is performed inside the communication unit, using a weighted round-robin arbiter.

The communication unit serves one request at a time. Since the GT–64240A’s internal data path bandwidth (64-bit @ 133MHz) is much higher than the total bandwidth required by ALL communication ports, it is designed to guarantee that there will never be an overrun or underrun condition.

Each agent (E0, E1, E2, SDMAs, I^2C) is assigned a programmable priority (high, medium, or low), and the arbitration is done according to these priorities. A simple Round Robin arbitration is performed within each priority level. A weighted function is implemented for arbitrating between the high, medium and the low priority groups.

Figure 39 shows the arbitration flow.

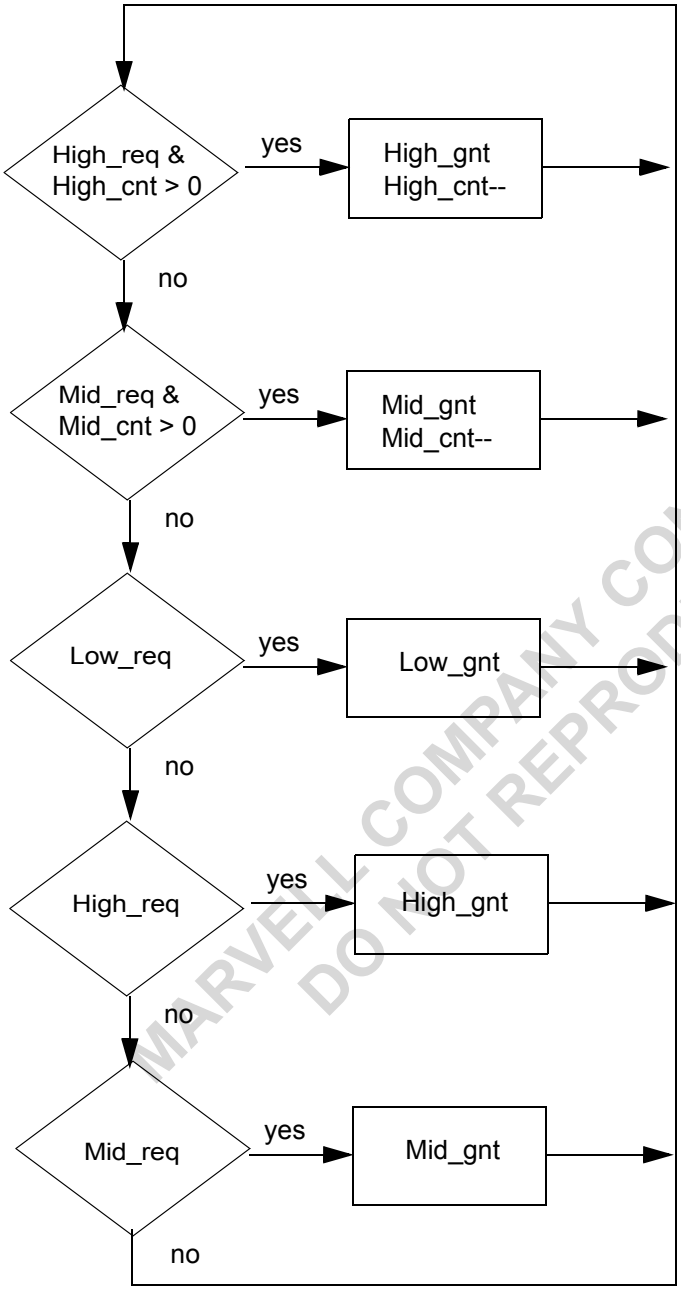
The Arbitration flow works as follows:

- The three request signals (High_req, Mid_req, Low_req) are generated by “ANDing” each of the request lines with its respective priority attribute, and ORing the results. For example:

  High_req = (req_[0] AND (req_prio[0]==high)) OR (req_[1] AND (req_prio[1]==high)) OR...

- Two counters are associated with the priority scheme - High_cnt and Mid_cnt. The counters assign different weights to each priority level. Each counter is a count down counter that decrements each time its corresponding priority request is granted. When the count expires, a slot opens for lower priority requests.
- When the mid and high counters reach zero, they return to their preset values. These preset values are set in the Comm Unit Arbiter Control register’s MCntVal (mid) and HCntVal (high) bits, see Table 530 on page 334.

Figure 39: Comm Unit Arbiter Flow



12.5 Communication Unit Registers

Table 496: Communication Unit Register Map

Register	Offset	Page
Ethernet 0 Address Control (Low)	0xf200	page 326
Ethernet 0 Address Control (High)	0xf204	page 327
Ethernet 0 Receive Buffer PCI High Address	0xf208	page 328
Ethernet 0 Transmit Buffer PCI High Address	0xf20c	page 328
Ethernet 0 Receive Descriptor PCI High Address	0xf210	page 328
Ethernet 0 Transmit Descriptor PCI High Address	0xf214	page 329
Ethernet 0 Hash Table PCI High Address	0xf218	page 329
Ethernet 1 Address Control (Low)	0xf220	page 329
Ethernet 1 Address Control (High)	0xf224	page 329
Ethernet 1 Receive Buffer PCI High Address	0xf228	page 329
Ethernet 1 Transmit Buffer PCI High Address	0xf22c	page 329
Ethernet 1 Receive Descriptor PCI High Address	0xf230	page 329
Ethernet 1 Transmit Descriptor PCI High Address	0xf234	page 330
Ethernet 1 Hash Table PCI High Address	0xf238	page 330
Ethernet 2 Address Control (Low)	0xf240	page 330
Ethernet 2 Address Control (High)	0xf244	page 330
Ethernet 2 Receive Buffer PCI High Address	0xf248	page 330
Ethernet 2 Transmit Buffer PCI High Address	0xf24c	page 330
Ethernet 2 Receive Descriptor PCI High Address	0xf250	page 330
Ethernet 2 Transmit Descriptor PCI High Address	0xf254	page 331
Ethernet 2 Hash Table PCI High Address	0xf258	page 331
MPSC 0 Address Control (Low)	0xf280	page 331
MPSC 0 Address Control (High)	0xf284	page 332
MPSC 0 Receive Buffer PCI High Address	0xf288	page 333
MPSC 0 Transmit Buffer PCI High Address	0xf28c	page 333
MPSC 0 Receive Descriptor PCI High Address	0xf290	page 333
MPSC 0 Transmit Descriptor PCI High Address	0xf294	page 333
MPSC 1 Address Control (Low)	0xf2c0	page 333
MPSC 1 Address Control (High)	0xf2c4	page 333

Table 496: Communication Unit Register Map (Continued)

Register	Offset	Page
MPSC 1 Receive Buffer PCI High Address	0xf2c8	page 333
MPSC 1 Transmit Buffer PCI High Address	0xf2cc	page 334
MPSC 1 Receive Descriptor PCI High Address	0xf2d0	page 334
MPSC 1 Transmit Descriptor PCI High Address	0xf2d4	page 334
Comm Unit Arbiter Control	0xf300	page 334
Comm Unit Configuration	0xb40c	page 335
Comm Unit Crossbar Timeout	0xf304	page 336
Comm Unit Interrupt Cause	0xf310	page 336
Comm Unit Interrupt Mask	0xf314	page 337
Comm Unit Error Address	0xf318	page 338

Table 497: Ethernet 0 Address Control (Low), Offset: 0xf200

Bits	Field Name	Function	Initial Value
3:0			0x0
5:4	RxBPCISwap	PCI Master Data Swap Control 00 - Byte Swap 01 - No swapping 10 - Both byte and word swap 11 - Word swap Only applicable when using RxBAddrOvr. Otherwise, the PCI master data swapping is controlled via the CPU to PCI Address Decode register's PCISwap field.	0x1
6			0x0
7	RxBPCIReq64	PCI Master REQ64* Policy 0 - Only asserts REQ64* when the write to receive buffer is longer than 64-bits. 1 - Always assert REQ64*. Only applicable when using RxBAddrOvr. Otherwise, the PCI master REQ64* policy is controlled via the CPU to PCI Address Decode register's Req64 bit.	0x0
11:8			0x0
13:12	TxBPCISwap	Transmit Buffer PCI Master Data Swap Control	0x1
14			0x0

Table 497: Ethernet 0 Address Control (Low), Offset: 0xf200 (Continued)

Bits	Field Name	Function	Initial Value
15	TxBPCIReq64	PCI Master REQ64* Policy 0 - Only asserts REQ64* when the read from transmit buffer is longer than 64-bits. 1 - Always assert REQ64*. Only applicable when using TxBAddrOvr. Otherwise, the PCI master REQ64* policy is controlled via the CPU to PCI Address Decode register's Req64 bit.	0x0
19:16			0x0
21:20	RxDPCISwap	Receive Descriptor PCI Master Data Swap Control	0x1
22			0x0
23	RxDPCIReq64	PCI Master REQ64* Policy 0 - Only asserts REQ64* when the read of the receive descriptor is longer than 64-bits. 1 - Always assert REQ64*. Only applicable when using RxDAddrOvr. Otherwise, the PCI master REQ64* policy is controlled via the CPU to PCI address decode register's Req64 bit.	0x0
27:24			0x0
29:28	TxDPCISwap	Transmit Descriptor PCI Master Data Swap Control	0x1
30			0x0
31	TxDPCIReq64	PCI Master REQ64* Policy 0 - Only asserts REQ64* when the read of the transmit descriptor is longer than 64-bits. 1 - Always assert REQ64*. Only applicable when using TxDAddrOvr. Otherwise, the PCI master REQ64* policy is controlled via the CPU to PCI address decode register's Req64 bit.	0x0

Table 498: Ethernet 0 Address Control (High), Offset: 0xf204

Bits	Field Name	Function	Initial Value
3:0			0x0
5:4	HashPCISwap	Hash Table PCI Master Data Swap Control	0x1
6			0x0

Table 498: Ethernet 0 Address Control (High), Offset: 0xf204 (Continued)

Bits	Field Name	Function	Initial Value
7	HashPCIReq64	PCI Master REQ64* Policy 0 - Only asserts REQ64* when the read from the hash table is longer than 64-bits. 1 - Always assert REQ64*. Only applicable when using HashAddrOvr. Otherwise, the PCI master REQ64* policy is controlled via the CPU to PCI address decode register's Req64 bit.	0x0
9:8	RxBAddrOvr	Receive Buffer PCI Override 00 - No override. 01 - Buffer address is in PCI_0 memory space. 10 - Buffer address is in PCI_1 memory space. 11 - Reserved.	0x0
11:10	TxBAddrOvr	Transmit Buffer PCI Override.	0x0
13:12	RxDAddrOvr	Receive Descriptor PCI Override	0x0
15:14	TxDAddrOvr	Transmit Descriptor PCI Override	0x0
17:16	HashAddrOvr	Hash Table PCI Override	0x0
31:	Reserved		0x0

Table 499: Ethernet 0 Receive Buffer PCI High Address, Offset: 0xf208

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 500: Ethernet 0 Transmit Buffer PCI High Address, Offset: 0xf20c

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 501: Ethernet 0 Receive Descriptor PCI High Address, Offset: 0xf210

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 502: Ethernet 0 Transmit Descriptor PCI High Address, Offset: 0xf214

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 503: Ethernet 0 Hash Table PCI High Address, Offset: 0xf218

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 504: Ethernet 1 Address Control (Low), Offset: 0xf220

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the Ethernet 0 Address Control register.	0x10101010

Table 505: Ethernet 1 Address Control (High), Offset: 0xf224

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the Ethernet 1 Address Control register.	0x10

Table 506: Ethernet 1 Receive Buffer PCI High Address, Offset: 0xf228

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 507: Ethernet 1 Transmit Buffer PCI High Address, Offset: 0xf22c

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 508: Ethernet 1 Receive Descriptor PCI High Address, Offset: 0xf230

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 509: Ethernet 1 Transmit Descriptor PCI High Address, Offset: 0xf234

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 510: Ethernet 1 Hash Table PCI High Address, Offset: 0xf238

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 511: Ethernet 2 Address Control (Low), Offset: 0xf240

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the Ethernet 0 Address Control register.	0x10101010

Table 512: Ethernet 2 Address Control (High), Offset: 0xf244

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the Ethernet 1 Address Control register.	0x10

Table 513: Ethernet 2 Receive Buffer PCI High Address, Offset: 0xf248

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 514: Ethernet 2 Transmit Buffer PCI High Address, Offset: 0xf24c

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 515: Ethernet 2 Receive Descriptor PCI High Address, Offset: 0xf250

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 516: Ethernet 2 Transmit Descriptor PCI High Address, Offset: 0xf254

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 517: Ethernet 2 Hash Table PCI High Address, Offset: 0xf258

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 518: MPSC 0 Address Control (Low), Offset: 0xf280

Bits	Field Name	Function	Initial Value
3:0			0x0
5:4	RxBPCISwap	PCI Master Data Swap Control 00 - Byte Swap 01 - No swapping 10 - Both byte and word swap 11 - Word swap Only applicable when using RxBAddrOvr. Otherwise, PCI master data swapping is controlled via the CPU to PCI Address Decode register's PCISwap field.	0x1
6			0x0
7	RxBPCIReq64	PCI Master REQ64* Policy 0 - Only asserts REQ64* when the write to the receive buffer is longer than 64-bit. 1 - Always assert REQ64*. Only applicable when using RxBAddrOvr. Otherwise, the PCI master REQ64* policy is controlled via the CPU to PCI Address Decode register's Req64 bit.	0x0
11:8			0x0
13:12	TxBPCISwap	Transmit Buffer PCI Master Data Swap Control	0x1
14			0x0
15	TxBPCIReq64	PCI Master REQ64* Policy 0 - Only asserts REQ64* when the read from the transmit buffer is longer than 64-bit. 1 - Always assert REQ64*. Only applicable when using TxBAddrOvr. Otherwise, the PCI master REQ64* policy is controlled via the CPU to PCI address decode register's Req64 bit.	0x0
19:16			0x0

Table 518: MPSC 0 Address Control (Low), Offset: 0xf280 (Continued)

Bits	Field Name	Function	Initial Value
21:20	RxDPCISwap	Receive Descriptor PCI Master Data Swap Control	0x1
22			0x0
23	RxDPCIReq64	PCI Master REQ64* Policy 0 - Only asserts REQ64* when the read of the receive descriptor is longer than 64-bits. 1 - Always assert REQ64*. Only applicable when using RxDAddrOvr. Otherwise, the PCI master REQ64* policy is controlled via the CPU to PCI address decode register's Req64 bit.	0x0
27:24			0x0
29:28	TxDPCISwap	Transmit Descriptor PCI Master Data Swap Control	0x1
30			0x0
31	TxDPCIReq64	PCI Master REQ64* Policy 0 - Only asserts REQ64* when the read of the receive descriptor is longer than 64-bits. 1 - Always assert REQ64*. Only applicable when using TxDAddrOvr. Otherwise, the PCI master REQ64* policy is controlled via the CPU to PCI address decode register's Req64 bit.	0x0

Table 519: MPSC 0 Address Control (High), Offset: 0xf284

Bits	Field Name	Function	Initial Value
7:0	Reserved	Reserved.	0x0
9:8	RxBAddrOvr	Receive Buffer PCI Override 00 - No override. 01 - Buffer address is in PCI_0 memory space. 10 - Buffer address is in PCI_1 memory space. 11 - Reserved.	0x0
11:10	TxBAddrOvr	Transmit Buffer PCI Override	0x0
13:12	RxDAddrOvr	Receive Descriptor PCI Override	0x0
15:14	TxDAddrOvr	Transmit Descriptor PCI Override	0x0
17:16	Reserved	Reserved.	0x0
31:18	Reserved	Must be 0.	0x0

Table 520: MPSC 0 Receive Buffer PCI High Address, Offset: 0xf288

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of PCI address	0x0

Table 521: MPSC 0 Transmit Buffer PCI High Address, Offset: 0xf28c

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of PCI address	0x0

Table 522: MPSC 0 Receive Descriptor PCI High Address, Offset: 0xf290

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of PCI address	0x0

Table 523: MPSC 0 Transmit Descriptor PCI High Address, Offset: 0xf294

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 524: MPSC 1 Address Control (Low), Offset: 0xf2c0

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the MPSC 0 Address Control register.	0x10101010

Table 525: MPSC 1 Address Control (High), Offset: 0xf2c4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as the MPSC 0 Address Control register.	0x1000000

Table 526: MPSC 1 Receive Buffer PCI High Address, Offset: 0xf2c8

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 527: MPSC 1 Transmit Buffer PCI High Address, Offset: 0xf2cc

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 528: MPSC 1 Receive Descriptor PCI High Address, Offset: 0xf2d0

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 529: MPSC 1 Transmit Descriptor PCI High Address, Offset: 0xf2d4

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 530: Comm Unit Arbiter Control, Offset: 0xf300

Bits	Field Name	Function	Initial Value
1:0	Eth0	Ethernet Controller 0 Priority 00 - Low 01 - Medium 10 - High 11 - Reserved	0x0
3:2	Eth1	Ethernet Controller 1 Priority Same as E0Prio.	0x0
5:4	Eth2	Ethernet Controller 2 Priority Same as E0Prio.	0x0
7:6	I2C	I2C interface Priority Same as E0Prio.	0x0
9:8	MPSC	MPSCs SDMAs Priority Same as E0Prio.	0x0
10	GPPInt	Selects either GPP edge or level interrupts. 0 - Edge trigger interrupt (default setting) 1 - Level sensitive interrupt	0x0
15:11	Reserved	Reserved.	0x0

Table 530: Comm Unit Arbiter Control, Offset: 0xf300 (Continued)

Bits	Field Name	Function	Initial Value
18:16	HCntVal	High Priority Counter Preset Value High_cnt counter decrements each time a high priority request is granted. When the counter reaches zero, it reloads with this preset value.	0x7
19	HCntEn	High Priority Counter Enable 0 - Medium and Low priority requests are only granted when no high priority request is pending 1 - Weighted round robin arbitration is performed between high priority, medium, and low priority groups.	0x1
22:20	MCntVal	Medium Priority Counter Preset Value Mid_cnt counter decrements each time a medium priority request is granted. When the counter reaches zero, it reloads with the preset value.	0x7
23	MCntEn	Medium Priority Counter Enable 0 - Low priority requests are only granted when no medium priority request is pending. 1 - Weighted round robin arbitration is performed between medium priority and low priority groups.	0x1
31:24	Reserved	Reserved.	0x11

Table 531: Comm Unit Configuration Register (CUACR), Offset: 0xb40c

Bits	Field Name	Function	Initial Value
23:0	Reserved	Reserved.	0x0
26:24	EthRst	Ethernet ports soft reset. **NOTE:** Reserved for Galileo Technology usage.	0x0
30:27	Reserved	Reserved.	0x0
31	DescEnd	Descriptors endianess. 0 - Big Endian 1 - Little Endian **NOTE:** Reserved for Galileo Technology usage.	0x1

Table 532: Comm Unit Crossbar Timeout, Offset: 0xf304
NOTE: Reserved for Galileo Technology usage.

Bits	Field Name	Function	Initial Value
7:0	Timeout	Crossbar Arbiter Timeout Preset Value	0xff
15:8	Reserved	Reserved.	0x0
16	TimeoutEn	CrossBar Arbiter Timer Enable 0 - Enable 1 - Disable	0x1
31:17	Reserved	Reserved.	0x0

Table 533: Comm Unit Interrupt Cause, Offset: 0xf310[1]

Bits	Field Name	Function	Initial Value
0	E0AddrMiss	Ethernet 0 Address Miss Failed address decoding.	0x0
1	E0AccProt	Ethernet 0 Access Protect Violation	0x0
2	E0WrProt	Ethernet 0 Write Protect Violation	0x0
3	Reserved	Reserved.	0x0
4	E1AddrMiss	Ethernet 1 Address Miss Failed address decoding.	0x0
5	E1AccProt	Ethernet 1 Access Protect Violation	0x0
6	E1WrProt	Ethernet 1 Write Protect Violation	0x0
7	Reserved	Reserved.	0x0
8	E2AddrMiss	Ethernet 2 Address Miss Failed address decoding.	0x0
9	E2AccProt	Ethernet 2 Access Protect Violation	0x0
10	E2WrProt	Ethernet 2 Write Protect Violation	0x0
15:11	Reserved	Reserved.	0x0
16	S0AddrMiss	MPSC 0 Address Miss Failed address decoding.	0x0
17	S0AccProt	MPSC 0 Access Protect Violation	0x0
18	S0WrProt	MPSC 0 Write Protect Violation	0x0
23:19	Reserved	Reserved.	0x0
24	S1AddrMiss	MPSC 1 Address Miss Failed address decoding.	0x0

Table 533: Comm Unit Interrupt Cause, Offset: 0xf310[1] (Continued)

Bits	Field Name	Function	Initial Value
25	S1AccProt	MPSC 1 Access Protect Violation	0x0
26	S1WrProt	MPSC 1 Write Protect Violation	0x0
31:27	Reserved	Reserved.	0x0

1. All cause bits are "Clear Only." They are set to '1' upon the interrupt event and cleared when the software writes a value of '0'. Writing '1' has no affect.

Table 534: Comm Unit Interrupt Mask, Offset: 0xf314

Bits	Field Name	Function	Initial Value
0	E0AddrMiss	If set to '1', E0AddrMiss interrupt is enabled.	0x0
1	E0AccProt	If set to '1', E0AccProt interrupt is enabled.	0x0
2	E0WrProt	If set to '1', E0WrProt interrupt is enabled.	0x0
3	Reserved	Reserved.	0x0
4	E1AddrMiss	If set to '1', E1AddrMiss interrupt is enabled.	0x0
5	E1AccProt	If set to '1', E1AccProt interrupt is enabled.	0x0
6	E1WrProt	If set to '1', E1WrProt interrupt is enabled.	0x0
7	Reserved	Reserved.	0x0
8	E2AddrMiss	If set to '1', E2AddrMiss interrupt is enabled.	0x0
9	E2AccProt	If set to '1', E2AccProt interrupt is enabled.	0x0
10	E2WrProt	If set to '1', E2WrProt interrupt is enabled.	0x0
15:11	Reserved	Reserved.	0x0
16	S0AddrMiss	If set to '1', S0AddrMiss interrupt is enabled.	0x0
17	S0AccProt	If set to '1', S0AccProt interrupt is enabled.	0x0
18	S0WrProt	If set to '1', S0WrProt interrupt is enabled.	0x0
19	Reserved	Reserved.	0x0
23:20	Reserved	Reserved.	0x0
24	S1AddrMiss	If set to '1', S1AddrMiss interrupt is enabled.	0x0
25	S1AccProt	If set to '1', S1AccProt interrupt is enabled.	0x0
26	S1WrProt	If set to '1', S1WrProt interrupt is enabled.	0x0

Table 534: Comm Unit Interrupt Mask, Offset: 0xf314 (Continued)

Bits	Field Name	Function	Initial Value
31:27	Sel	Specifies the error event currently being reported in the Error Address register. 0x0 - E0AddrMiss 0x1 - E0AccProt 0x2 - E0WrProt 0x3 - Reserved 0x4 - E1AddrMiss 0x5 - E1AccProt 0x6 - E1WrProt 0x7 - Reserved 0x8 - E2AddrMiss 0x9 - E2AccProt 0xa - E2WrProt 0xb - 0xf Reserved 0x10 - S0AddrMiss 0x11 -S0AccProt 0x12 - S0WrProt 0x13 - Reserved 0x14 - 0x17 Reserved 0x18 - S1AddrMiss 0x19 -S1AccProt 0x1a -S1WrProt Read only.	0x0

Table 535: Comm Unit Error Address, Offset: 0xf318

Bits	Field Name	Function	Initial Value
31:0	ErrAddr	Bits[31:0] of Error Address Latched upon any of the interrupt events. No new address can be latched (due to additional error condition) until the register is being read.	0x0

13. 10/100Mb Ethernet Unit

GT–64240A contains three Ethernet controllers. They can be configured to MII or RMII interface.

NOTE: In case of MII, only two Ethernet ports are available.

13.1 Functional Overview

The 10/100Mb Ethernet unit handles all functionality associated with moving packet data between local memory or PCI and the Ethernet ports. The unit in the GT–64240A is designed to support three independent 10/100Mb Ethernet ports.

Each 10/100 Mbit port is fully compliant with the IEEE 802.3 and 802.3u standards and integrates the MAC function and a dual speed MII interface. The port's speed (10 or 100Mb/s) as well as the duplex mode (half or full duplex) is auto-negotiated through the PHY and does not require user intervention. The port also features 802.3x flow-control mode for full-duplex and backpressure mode for half duplex.

Integrated address filtering logic provides support for up to 8K MAC addresses. The address table resides in memory and a proprietary hash function is used for address table management. The address table functionality supports Multicast as well as Unicast address entries.

An important feature related to the address recognition is IGMP packet trapping mode. In this mode layer 3 hardware analysis is performed in order to check if a packet being received is an IGMP packet. Each packet identified as IGMP is queued in the high priority queue of the port from which it was received. The IGMP analysis is performed on the fly, so it does not impact bandwidth capability.

The Ethernet unit integrates powerful DMA engines, which automatically manage data movement between buffer memory and the ports, and guarantee wire-speed operation on all ports (even when all ports are in 100Mb full-duplex mode). There are two DMA engines per port - one dedicated for receive and the other for transmit.

The DMA logic handles multiple priority queues per port, providing support for priority sensitive data in both directions. There are four receive priority queues and two transmit priority queues per port. Priority information for received packets is either extracted from the packet tag (if the packet is VLAN tagged) or from the destination-address entry in the address table (if the packet is not tagged).

13.2 Port Features

The 10/100Mb Ethernet port provides the following features:

- IEEE 802.3 compliant MAC layer function.
- IEEE 802.3u compliant MII interface.
- 10/100Mb operation - half and full duplex.
- Flow control features:
  - IEEE 802.3x flow-control for full-duplex operation mode.
  - Backpressure for half duplex operation mode.
- Internal and external loop back modes.
- Transmit functions:
  - Short frame (less than 64 bytes) zero padding.
  - Long frames transmission (limited only by external memory size).

- Programmable values for Inter Packet Gap and Blinder timers.
- CRC generation (programmable per packet).
- Automatic frame retransmission upon collision (with programmable retransmit limit).
- Backoff algorithm execution.
- Error report.

• Receive functions:

- 1/2k or 8k address filtering capability.
- Address filtering modes:
  - Perfect filtering.
  - Reverse filtering.
  - Promiscuous mode.
  - Broadcast reject mode.
- IGMP packet trapping (layer 3 analysis in hardware).
- Automatic discard of errored frames, short (less than 64 bytes) or collided.
- Reception of long frames (programmable up to 64Kbytes).
- CRC checking.
- Pass bad frames mode.
- Error report.

13.3 Operational Description

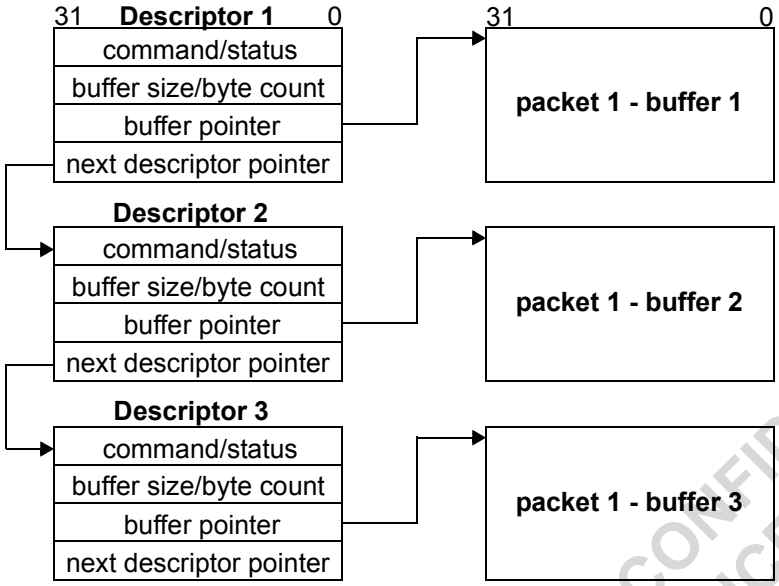
13.3.1 General Overview

The Ethernet unit provides multiple Ethernet ports functionality, with each port capable of running at either 10 or 100Mb/s (half or full-duplex) independently of the other port. Each port interfaces a MII PHY on its serial side and manages packet data transfer between memory and MII. The data is stored in memory buffers, with any single packet spanning multiple buffers if necessary. Upon completion of packet transmission or reception, a status report, which includes error indications, is written by the Ethernet unit to the first or last descriptor associated with this packet.

The buffers are allocated by the CPU and are managed through chained descriptor lists. Each descriptor points to a single memory buffer and contains all the relevant information relating to that buffer (i.e. buffer size, buffer pointer, etc.) and a pointer to the next descriptor. Data is read from buffer or written to the buffer according to information contained in the descriptor. Whenever a new buffer is needed (end of buffer or end of packet), a new descriptor is automatically fetched and the data movement operation is continued using the new buffer.

Figure 40 shows an example of memory arrangement for a single packet using three buffers.

Figure 40: Ethernet Descriptors and Buffers



The following sections provide detailed information about the operation and user interface of the Ethernet unit and its logic subsections.

13.3.2 Transmit Operation

In order to initialize a transmit operation, the CPU must do the following:

1. Prepare a chained list of descriptors and packet buffers.

NOTE: The TxDMA supports two priority transmit queues - high and low. If the user wants to take advantage of this capability, a separate list of descriptors and buffers must be prepared for each of the priority queues.

2. Write the pointer to the first descriptor to the DMA's current descriptor registers (TxCDP) associated with the priority queue to be started. If both priority queues are needed, initialize TxCDP for each queue.
3. Initialize and enable the Ethernet port by writing to the port's configuration and command registers.
4. Initialize and enable the DMA by writing to the DMA's configuration and command registers.

After completing these steps, the DMA starts and performs arbitration between the transmit queues according to the value programmed in Port Configuration Extend register's PRIOtx bits [5:3] (see Table 556 on page 375). The DMA then fetches the first descriptor from the specific queue it decided to serve, and starts transferring data from memory buffer to the Tx-FIFO. When either 384 bytes of packet data are in the FIFO or when the entire packet is in the FIFO (for packets shorter than 384 bytes), the port initiates transmission of the packet across the MII. While data is read from the FIFO, new data is written into the FIFO by the DMA.

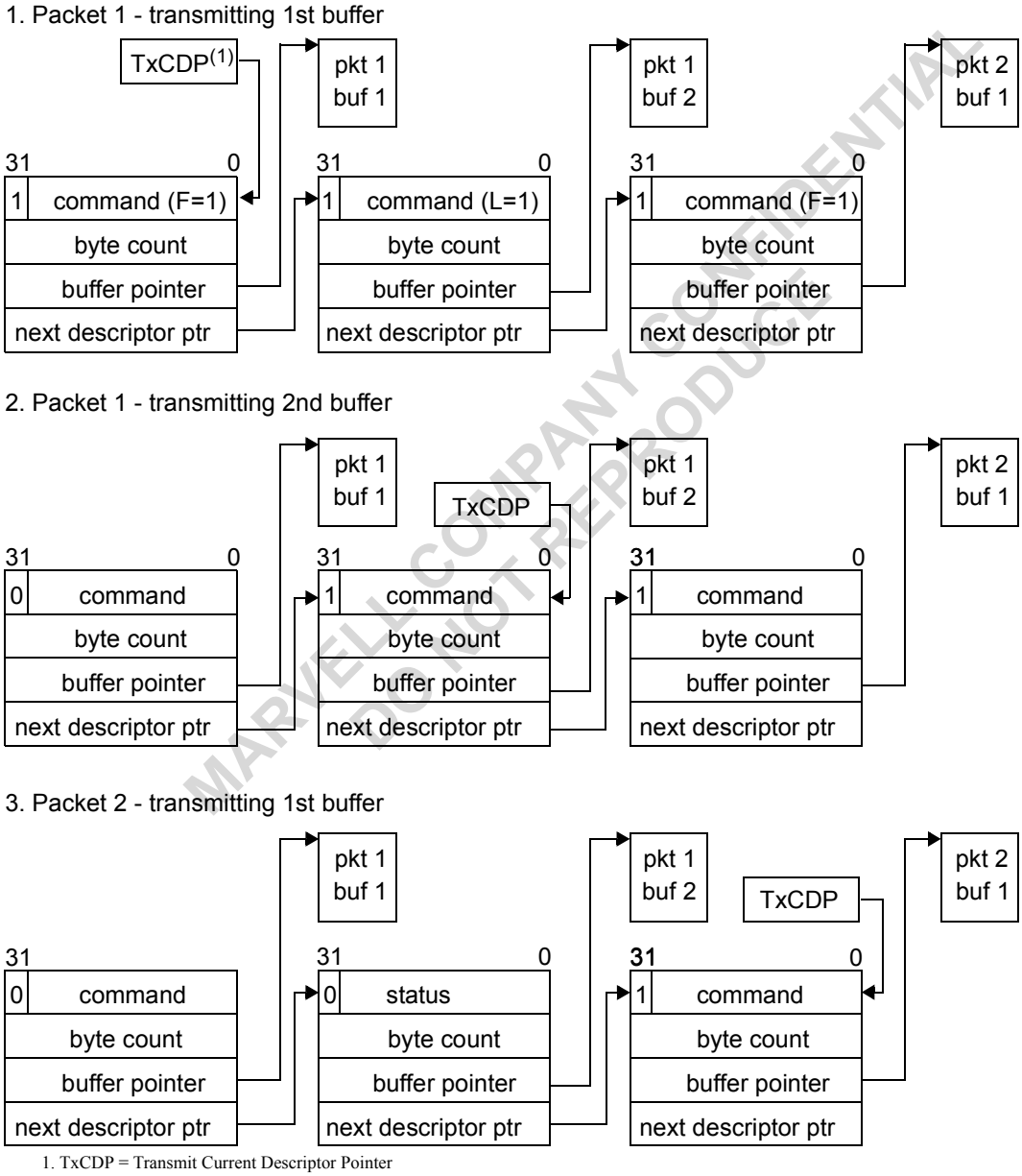
For packets that span more than one buffer in memory, the DMA will fetch new descriptors and buffers as necessary.

When transmission is completed, status is written to the first longword of the last descriptor. The Next Descriptor's address, which belongs to the next packet in the queue, is written to the current descriptor pointer register.

This process (starting with DMA arbitration) is repeated as long as there are packets pending in the transmit queues.

Figure 41 shows how the Tx descriptors are managed when a two buffers packet is transmitted.

Figure 41: Ethernet Packet Transmission Example



1. TxCDP = Transmit Current Descriptor Pointer

Ownership of any descriptor other than the last is returned to CPU upon completion of data transfer from the buffer pointed by that descriptor. The Last descriptor, however, is returned to CPU ownership only after the actual transmission of the packet is completed. While changing the ownership bit of the last descriptor, the DMA also writes status information, which indicates any errors that might have happened during transmission of this packet.

13.3.2.1 Retransmission (Collision)

Full collision support is integrated into the Ethernet port for half duplex operation mode.

In half duplex operation mode, a collision event is indicated each time receive and transmit are active simultaneously. When that happens, active transmission is stopped, jam pattern is transmitted and collision count for the packet increments. The packet is retransmitted after a waiting period, which conforms to the binary Backoff algorithm specified in the IEEE 802.3 standard. Retransmit process continues for multiple collision events as long as a specified limit is not reached. This retransmit limit, which sets the maximum number or transmit retries for a single packet, is defined by the IEEE 802.3 as 16. However, the user can program a different value (see Table 563 on page 382 for more details). The event of a single packet colliding 16 times is known as **EXCESSIVE COLLISION**.

As long as a packet is being retransmitted, its last descriptor is kept under port ownership. When a successful transmission takes place (i.e. no collision), a status word containing collision information is written to the last descriptor and ownership is returned to CPU.

If a retransmit limit is reached with no successful transmission, a status word with error indication is written to the packet's last descriptor, and the transmit process continuous with the next packet.

It is important to note that collision is considered legal only if it happens before transmitting the 65th byte of a packet. Any collision event that happens outside the first 64 byte window is known as **LATE COLLISION**, and is considered a fatal network error. Late collision is reported to the CPU through packet status, and no retransmission is done.

NOTE: A collision occurring during the transmission of the transmit packet's last two bytes are not detected.

13.3.2.2 Zero Padding (for short packets)

Zero Padding is a term used to denote the operation of adding zero bytes to a packet. This feature is used for CPU off-loading.

The Ethernet port offers a per packet padding request bit in the Tx descriptor. This causes the port logic to enlarge packets shorter than 64 bytes by appending zero bytes. When this feature is used, only packets equal or larger than 64 bytes are transmitted as is. Packets smaller than 64 bytes are zero padded and transmitted as 64 byte packets.

13.3.2.3 CRC Generation

Ethernet CRC denotes four bytes of Frame-Check-Sequence appended to each packet.

CRC logic is integrated into the port and can be used to automatically generate and append CRC to a transmitted packet. One bit in the Tx descriptor is used for specifying if CRC generation is required for a specific packet.

13.3.2.4 Tx DMA Descriptors

Figure 42 depicts the format of Tx DMA descriptors. The following set of restrictions apply to TX descriptors:

- Descriptor length is 4LW and it must be 4LW aligned (i.e. Descriptor_Address[3:0]=0000).
- Descriptors may reside anywhere is CPU address space except for NULL address (0x00000000), which is used to indicate end of descriptor chain.
- Last descriptor in the linked chain must have a NULL value in its NextDescriptorPointer field.
- TX buffers associated with TX descriptors are limited to 64K bytes and can reside anywhere in memory. However, buffers with a payload smaller than 8 bytes must be aligned to 64-bit boundary. Figure 43 illustrates possible alignments for 5 byte payload.

Figure 42: Ethernet TX Descriptor

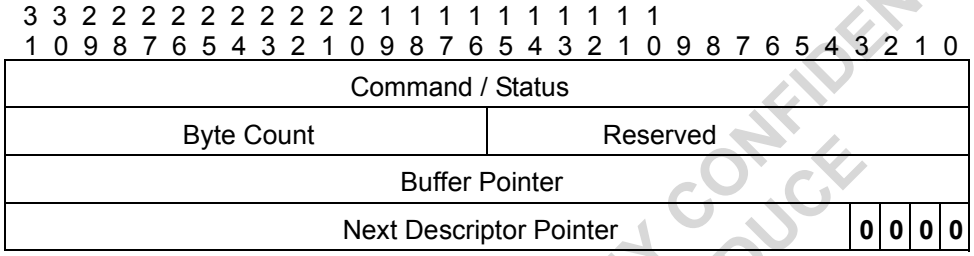


Figure 43: Ethernet TX Buffer Alignment Restrictions (5 byte payload)

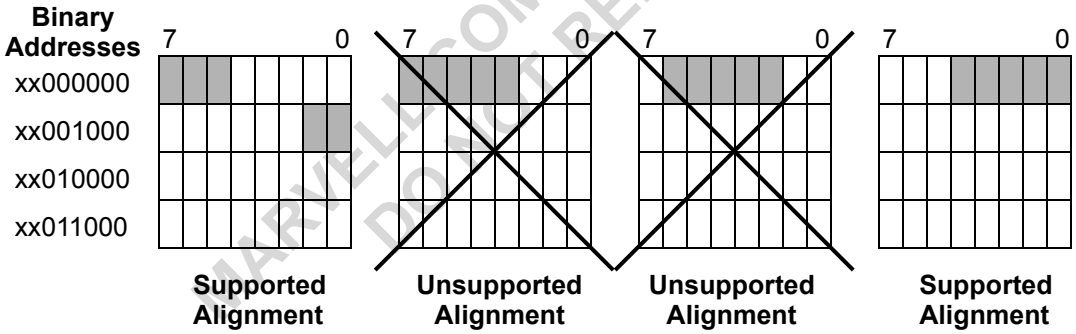


Table 536 through Table 539 provide detailed information about the TX descriptor.

Table 536: Ethernet TX Descriptor - Command/Status word

Bits	Field Name	Function
4:0	Reserved	Reserved.
5	LC	Late Collision error Collision occurred outside the collision window (i.e. more than 512 bits were transmitted before collision assertion). **NOTE:** Valid only if L (bit 16) is set.
6	UR	Under-Run error Indicates that part of the packet's data was not available while transmission was in progress, probably due to memory access delays). **NOTE:** Valid only if L (bit 16) is set
7	Reserved	Reserved.
8	RL	Retransmit Limit (Excessive Collision) error Indicates that retransmit count reached the limit specified in the DMA configuration register, see Table 563 on page 382). **NOTE:** Valid only if L (bit 16) is set
9	COL	Collision When set, indicates that at least one collision event occurred during transmission of the packet. **NOTE:** Valid only if L (bit 16) is set.
13:10	RC[3:0]	Retransmit Count Indicates actual number of retransmits for this packet. **NOTE:** Valid only if L (bit 16) is set
14	Reserved	Reserved.
15	ES	Error Summary ES = LC or UR or RL Set by the device to indicate an error event that occurred during the packet. **NOTE:** Valid only if L (bit 16) is set.
16	L	Last Indicates last buffer of a packet.
17	F	First Indicates first buffer of a packet.
18	P	Padding When this bit is set, zero bytes are appended to the packet if the packet is smaller than 60 bytes. Use this feature to prevent transmission of fragments. **NOTE:** All frame descriptors should be set to same value.
21:19	Reserved	Reserved.
22	GC	Generate CRC When set, CRC is generated and appended to this packet. **NOTE:** Valid only if L (bit 16) is set.

Table 536: Ethernet TX Descriptor - Command/Status word (Continued)

Bits	Field Name	Function
23	EI	Enable Interrupt The device generates a maskable TxBuffer interrupt upon closing the descriptor. **NOTE:** In order to limit the number of interrupts and prevent an interrupt per buffer situation, the user should set this bit only in descriptors associated with LAST buffers. If this is done, TxBuffer interrupt will be set only when transmission of a frame is completed.
29:24	Reserved	Reserved.
30	AM	Auto Mode When set, the DMA does not clear the Ownership bit at the end of buffer processing.
31	O	Ownership bit When set to '1', the buffer is "owned" by the device. When set to '0', the buffer is owned by the CPU. Buffers owned by the CPU are not processed by the DMA.

Table 537: Ethernet TX Descriptor - Byte Count

Bits	Field Name	Function
15:0		Reserved.
31:16	Byte Count	Number of bytes to be transmitted from associated buffer. This is the payload size in bytes.

Table 538: Ethernet TX Descriptor - Buffer Pointer

Bits	Field Name	Function
31:0	Buffer Pointer	32-bit pointer to the beginning of the buffer associated with this descriptor. **NOTE:** The alignment restrictions for buffers that have Byte-Count smaller than 8 bytes (see Figure 43 on page 344).

Table 539: Ethernet Tx Descriptor - Next Descriptor Pointer

Bits	Field Name	Function
31:0	Next Descriptor Pointer	32-bit pointer that points to the beginning of next descriptor. Bits [3:0] must be set to 0. DMA operation is stopped when a NULL (all zero) value in the Next Descriptor Pointer field is encountered.

13.3.2.5 Tx DMA Pointer Registers

The TX DMA employs a single 32-bit pointer register per queue: TxCDP.

- TxCDP - TX DMA Current Descriptor Pointer.

  TxCDP is a 32-bit register used to point to the current descriptor of a transmit packet. The CPU must initialize this register before enabling DMA operation. The value used for initialization should be the address of the first descriptor to use.

13.3.2.6 Tx DMA Notes

Transmit DMA process is packet oriented. The transmit DMA does not close the last descriptor of a packet, until the packet has been fully transmitted. When closing the last descriptor, the DMA writes packet transmission status to the Command/Status word and resets the ownership bit. A TxBuffer maskable interrupt is generated if the EI bit in the last descriptor is set.

Transmit DMA stops processing a Tx queue whenever a descriptor with a NULL value in the Next Descriptor Pointer field is reached or when a CPU owned descriptor is fetched. When that happens, a Tx_End maskable interrupt is generated. In order to restart the queue, the CPU should issue a Start_Tx command by writing '1' to the Start_Tx bit in the DMA command register. [1]

The transmit DMA does not expect a NULL Next Descriptor Pointer or a CPU owned descriptor in the middle of a packet. When that happens, the DMA aborts transmission and stops queue processing. A Tx_Resource_Error maskable interrupt is generated. In order to restart the queue, the CPU should issue a Start_Tx command.

A transmit underrun occurs when the DMA can not access the memory fast enough and packet data is not transferred to the FIFO before the FIFO gets empty. In this case, the DMA aborts transmission and closes the last descriptor with a UR bit set in the status word. Also, a Tx_Underrun maskable interrupt is generated. Transmit process continues with the next packet.

In order to stop DMA operation before the DMA reaches the end of descriptor chain, the CPU should issue a STOP command by writing '1' to the Stop_Tx bit in the DMA command register. The DMA stops queue processing as soon as the current packet transmission is completed and its last descriptor returned to CPU ownership. In addition, a Tx_End maskable interrupt is generated. In order to restart this queue, the CPU should issue a Start_Tx command.

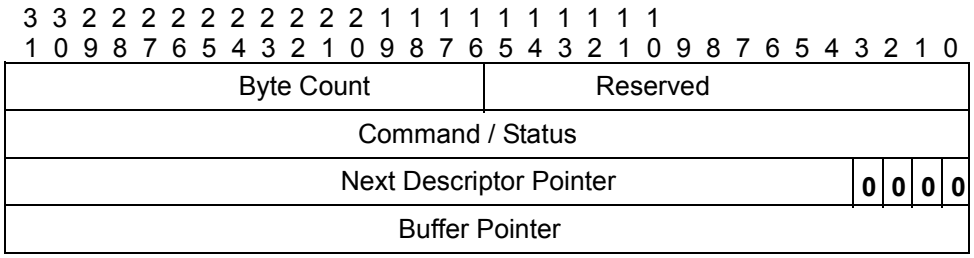
NOTE: Most of the terms used to denote either DMA commands (Start_Tx and Stop_Tx) or interrupts (TxBuffer, Tx_End, Tx_Resource_Error) actually reflect multiple terms (one per queue). For example, the GT–64240A provides two Start_Tx commands. There is a separate Start_Tx_High command, associated with the high priority queue, and a Start_Tx_low command that is related to the low priority queue. The same applies to the other commands and interrupts listed above.

When the Serial DMA fetches a descriptor from DRAM, it expects the descriptor in Little Endian format. However, since the DRAM byte order matches the CPU, consideration must be made when working with a CPU set for Big Endian.

If the software prepares the descriptors using 32-bit load/store operation, the descriptors must be word swapped when placed in DRAM. This means that the Tx descriptor must appear as described in Figure 44.

1. When the DMA stops due to NULL descriptor pointer, the CPU has to write TxCDP before issuing a Start_Tx command. Otherwise, TxCDP remains NULL and the DMA can not restart queue processing.

Figure 44: Word Swapped Ethernet TX Descriptor



13.3.3 Receive Operation

In order to initialize a receive operation, the CPU must do the following:

1. Prepare a chained list of descriptors and packet buffers.

NOTE: The RxDMA supports four priority queues. If the user wants to take advantage of this capability, a separate list of descriptors and buffers should be prepared for each of the priority queues.

2. Write the pointer to the first descriptor to the DMA's first and current descriptor registers (RxFDP, RxCDP) associated with the priority queue to be started. If multiple priority queues are needed, the user has to initialize RxFDP and RxCDP for each queue.
3. Initialize and enable the Ethernet port by writing to the port's configuration and command registers.
4. Initialize and enable the DMA channel by writing to the DMA's configuration and command registers.

After completing these steps, the port starts waiting for a receive frame to arrive at the MII interface. When this occurs, receive data is packed and transferred to the RxFiFO. At the same time, address filtering test is done in order to decide if the packet is destined to this port. If the packet passes address filtering check, a decision is made regarding the destination queue to which this packet should be transferred. When this is done, actual data transfer to memory takes place.

NOTE: Packets which fail address filtering are dropped and not transferred to memory.

For packets that span more than one buffer in memory, the DMA will fetch new descriptors as necessary. However, the first descriptor pointer will not be changed until packet reception is done.

When reception is completed, status is written to the first longword of the first descriptor, and the Next Descriptor's address is written to both first and current descriptor pointer registers. This process is repeated for each received packet.

NOTES: The RxCDP and RxFDP point to the same descriptor whenever the DMA is ready for receiving a new packet. RxFDP is not modified during packet reception and points to the first descriptor. Only after the packet had been fully received and status information was written to the first LW of the first descriptor, will the ownership bit be reset (i.e. descriptor returned to CPU ownership).

Ownership of any descriptor other than the first is returned to CPU upon completion of data transfer to the buffer pointed by that descriptor. This means that the first descriptor of a packet is the last descriptor to return to CPU ownership (per packet).

13.3.3.1 Rx DMA Descriptors

Figure 45 shows the format of Rx DMA descriptors.

The following set of restrictions apply to Rx descriptors:

- Descriptor length is 4LW and it must be 4LW aligned (i.e. Descriptor_Address[3:0]=0000).
- Descriptors reside anywhere in the CPU address space except NULL address, which is used to indicate end of descriptor chain.
- Rx buffers associated with Rx descriptors are limited to 64K bytes and must be 64-bit aligned. Minimum size for Rx buffers is 8 bytes.

Figure 45: Ethernet Rx DMA Descriptor

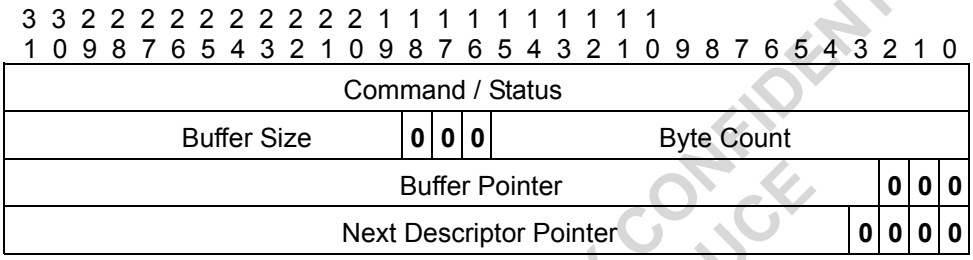


Table 540: Ethernet Rx Descriptor - Command/Status Word

Bits	Field Name	Function
0	CE	CRC Error Received CRC does not match calculated CRC for the received packet. **NOTE:** Valid only if F (bit 17) is set.
3:1	Reserved	Reserved.
4	COL	Collision Collision was sensed during packet reception. **NOTE:** In normal operation mode collided packets are automatically discarded by the port (being shorter than 64 bytes). Collided packets are accepted only when PBF is set in the Port Configuration register (see Table 555 on page 373). Valid only if F (bit 17) is set.
5	LC	Reserved.
6	OR	Overrun Error Indicates that the Rx DMA was unable to transfer data from RxFiFO to memory fast enough, causing data overrun in the FIFO. **NOTE:** Valid only if F (bit 17) is set.

Table 540: Ethernet Rx Descriptor - Command/Status Word (Continued)

Bits	Field Name	Function
7	MFL	Max Frame Length Error Indicates that a frame longer than MAX_FRAME_LEN was received. The maximum frame length is programmable (see Table 556 on page 375). **NOTE:** Valid only if F (bit 17) is set.
8	SF	Short Frame Error Indicates that a frame shorter than 64 bytes was received. In normal operation mode short packets are automatically discarded by the port. Short packets are accepted only when the Port Configuration register's PBF bit is set (see Table 555 on page 373). **NOTE:** Valid only if F (bit 17) is set.
10:9	Reserved	Reserved.
11	FT	Frame Type • 1 - 802.3 • 0 - Ethernet Set to '1' when the Type/Length field in the received packet has a value not bigger than 1500 (decimal). **NOTE:** Valid only if F (bit 17) is set.
12	M	Missed Frame • 0 - Match • 1 - Miss Set to indicate that this packet's destination address is not found in the address table. This bit may be set if HDM or PM are set in the Port Configuration register (see Table 555 on page 373). Also, set to receive broadcast packets regardless of the HDM or PM settings in the Port Configuration register. **NOTE:** This bit is valid only if F (bit 17) is set.
13	HE	Hash Table Expired Set to indicate that hash process was not completed in time. This means there is no definite answer as to whether this packet's address is in the hash table or not. Also, set when there is no room in the table for this address. **NOTE:** Valid only if F (bit 17) is set.
14	IGMP	Set to indicate that this packet has been identified as an IGMP packet. **NOTE:** Valid only if F (bit 17) is set.
15	ES	Error Summary ES = CE or COL or LC or OR or MFL or SF **NOTE:** Valid only if F (bit 17) is set.
16	L	Last Indicates last buffer of a packet.
17	F	First Indicates first buffer of a packet.

Table 540: Ethernet Rx Descriptor - Command/Status Word (Continued)

Bits	Field Name	Function
22:18		Reserved.
23	EI	Enable Interrupt The device generates a maskable interrupt upon closing the descriptor. **NOTE:** In order to limit the number of interrupts and prevent an interrupt per buffer situation, the user should set the EI bits in all the Rx descriptors and set RIFB bit in the DMA Configuration register (see Table 563 on page 382). The RxBuffer interrupt is set only on frame (rather than buffer) boundaries.
29:24		Reserved.
30	AM	Auto Mode When set, the DMA does not clear the Ownership bit at the end of buffer processing.
31	O	Ownership bit. When set to ‘1’, the buffer is “owned” by the device. When set to ‘0’, the buffer is owned by CPU.

Table 541: Ethernet Rx Descriptor - Buffer Size/Byte Count

Bits	Field Name	Function
15:0	Byte Count	When the descriptor is closed this field is written by the device with a value indicating number of bytes actually written by the DMA into the buffer.
31:16	Buffer Size	Buffer Size in Bytes When number of bytes written to this buffer is equal to Buffer Size value, the DMA closes the descriptor and moves to the next descriptor. **NOTE:** Bits [18:16] must be set to 0.

Table 542: Ethernet Rx Descriptor - Buffer Pointer

Bits	Field Name	Function
31:0	Buffer Pointer	32-bit Pointer to the beginning of the buffer associated with the descriptor. Rx buffers have to be 64-bit aligned, so bits [2:0] must be set to 0.

Table 543: Ethernet Rx Descriptor - Next Descriptor Pointer

Bits	Field Name	Function
31:0	Next Descriptor Pointer	32-bit Next Descriptor Pointer to the Beginning of Next Descriptor Bits [3:0] must be set to 0. DMA operation is stopped when a NULL value in the Next Descriptor Pointer field is encountered.

13.3.3.2 Rx DMA Pointer Registers

The Rx DMA employs two 32-bit pointer registers per queue: RxFDP and RxCDP.

- RxFDP - Rx DMA First Descriptor Pointer.

  RxFDP is a 32-bit register used to point to the first descriptor of a receive packet. The CPU must initialize this register before enabling DMA operation. The value used for initialization should be the address of the first descriptor to use.
- RxCDP - Rx DMA Current Descriptor Pointer.

  RxCDP is a 32-bit register used to point to the current descriptor of a receive packet. The CPU must initialize this register before enabling DMA operation. The value used for initialization should be the same as the value used for initializing RxFDP (i.e. address of first descriptor to use).

13.3.3.3 Rx Priority Queueing Type of Service Queueing

The GT–64240A supports four receive priority queues. The GT–64240A assigns priority to each packet according to the following algorithm:

1. The following packet types are always transferred to the HIGH priority queue:
   - IGMP packets (on IPv4,IPv6 over Ethernet/IEEE 802.3 SNAP).
     The IGMP packets are trapped only if the Port Configuration Extend register's IGMP bit [0] is set to '1', see Table 556 on page 375.
   - BPDU packets.
     BPDU packets are trapped only if Port Configuration Extend register's IGMP bit [1] bit is set to '1'.
2. The following algorithm is applied to determine the priority of the other packets received:

   If the Port Configuration Extend register's PRIOrx_Overide bit [8] is set to '1', priority is determined by the register's PRIOrx bits [7:6] .

   Otherwise the following is done:
   - If the packet is TAGed (per IEEE 802.1Q definition), priority is set according to the two most significant bits of the priority field in the tag.
   - If the destination address is found in the address table (and the packet is not tagged), priority is set according to the priority field in the address table.

In all other cases, priority is determined by Port Configuration Extend register's PRIOrx bits [7:6]. This includes BROADCAST packets, for which address lookup is not performed.

The Type of Service (TOS) queuing algorithm is based on the decoding of the DSCP field from the IP header. The DSCP field is located in the 6-MSB of the second byte in the IP header (See Figure 46). This field is the

index to the 64 IPT Table entries, residing in the GT–64240A Ethernet register space. This table's 2-bit priority output is referred to in the TOS algorithm as tos_priority.

NOTE: The tos_priority is only valid if the Ethernet Port Configuration Extend register's DSCPen bit [21] is enabled (see Table 556 on page 375).

If a VLAN tag exists in the packet, the VLAN priority tag is decoded from the 3-MSB bits of the second word in the VLAN tag. This field is the index to the eight entries in the VPT Table, residing in the GT–64240A Ethernet register space. This table's 2-bit priority output is referred to in the algorithm as vlan_priority.

The GT–64240A can decode BPDU and IGMP protocol packets, referred to in the TOS algorithm as frame_bpdu and frame_igmp. Protocol detection is controlled by the Ethernet Port Configuration Extend register's BPDU and IGMP bits [1:0].

If protocol detection is turned off, BPDU and IGMP will not send protocol packets to the highest value queue.

The Ethernet Port Configuration Extend register's PRIOrx_Override bit [8] takes precedence over tos_priority or vlan_priority. If this bit is set, all packets (except frame_bpdu and frame_igmp) are sent to the default priority queue (the PRIOrx bit in the Ethernet Port Configuration Extend register).

The packet type is checked after checking the source address, VLAN tag (if it exists) and LLC-SNAP (if it exists). The packet type is compared to offset 0x8100, referred to in the algorithm as vlan_type**,** or to offset 0x800, referred to in the algorithm as ip_type. If valid tos_priority is found in vlan_type or ip_type on the packet, it is referred to in the TOS algorithm as frame_tagged.

Broadcast packets, referred to in the algorithm as frame_broadcast and are not marked as frame_tagged, are also sent to the default priority queue.

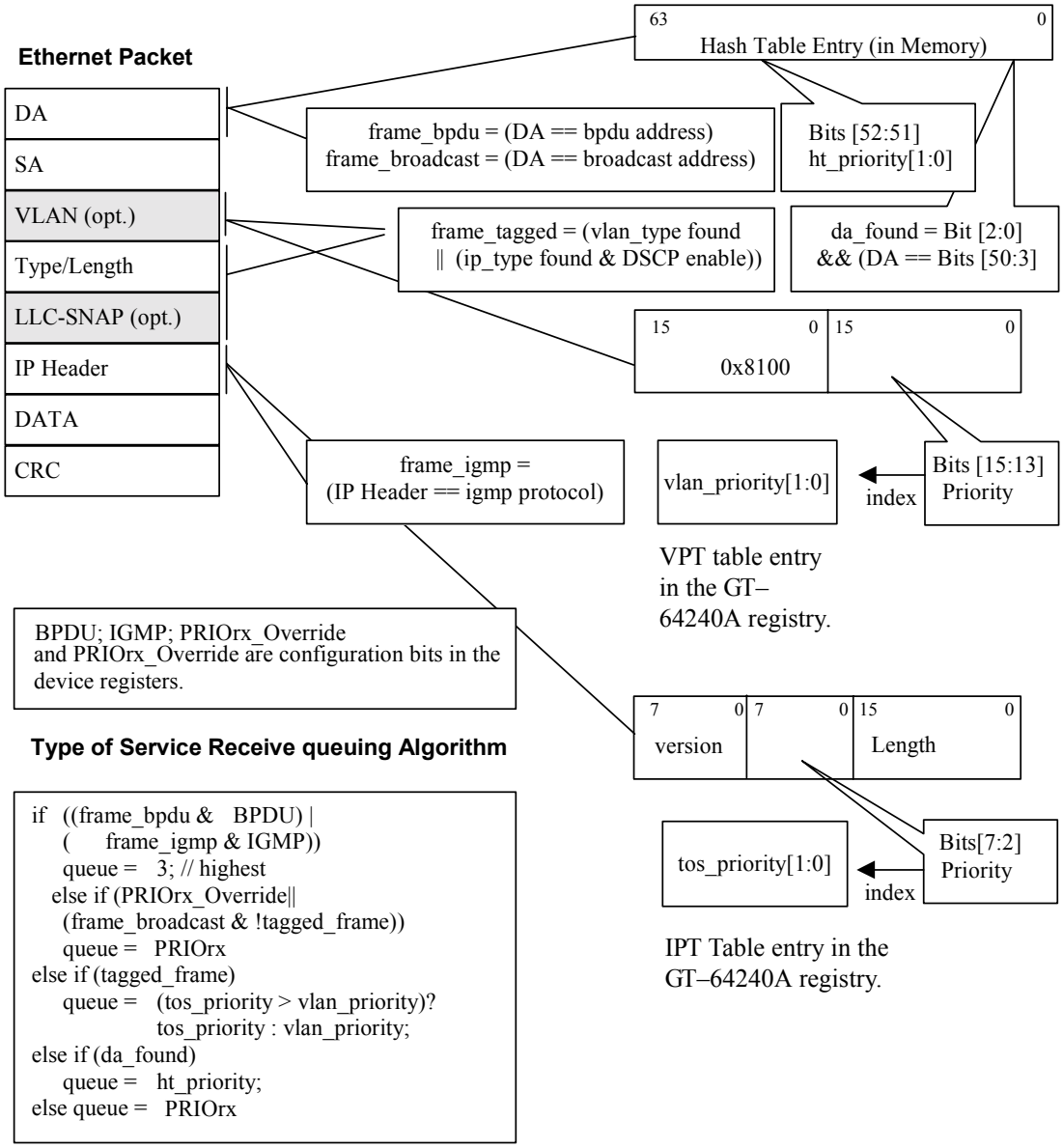
If the packet is marked as frame_tagged, the GT–64240A sends the packet to the tos_priority queue or vlan_priority queue. If both tos_priority and vlan_priority are extracted from the packet, the GT–64240A sends the packet to the higher value queue.

If tos_priority and vlan_priority are missing from the packet, the GT–64240A uses the priority value found in the matched Hash Table entry. The Hash Table entry match, referred to in the algorithm as da_found, occurs when the destination address matches the entry's address and the entry is valid.

The 2-bit priority value, referred to in the algorithm as ht_priority, is located on bits [52:51] of the Hash Table entry. The address to be compared is located on bits [50:3] of the Hash Table entry. The validity of the entry is stated in bits 2:0.

When the Hash Table entry does not return a priority value, the packet is sent to the default_priority queue.

Figure 46: Type of Service (TOS) Queueing Algorithm



13.3.3.4 Rx DMA notes

The Receive DMA process is packet oriented. The DMA does not close the first descriptor of a packet, until the last descriptor of the packet is closed. When closing the first descriptor, the DMA writes status to the Command/ Status word and resets the ownership bit. A RxBuffer maskable interrupt is generated if the EI bit in the first descriptor is set.

The receive DMA never expects a NULL next descriptor pointer or a CPU owned descriptor during normal operation. It is assumed that whenever the receive DMA needs a buffer, a buffer is ready for it. If this is not the case,

the RxDMA engine stops serving the current priority queue and a Rx_resource_error maskable interrupt is generated. To resume operation of the stopped queue, the following must be performed:

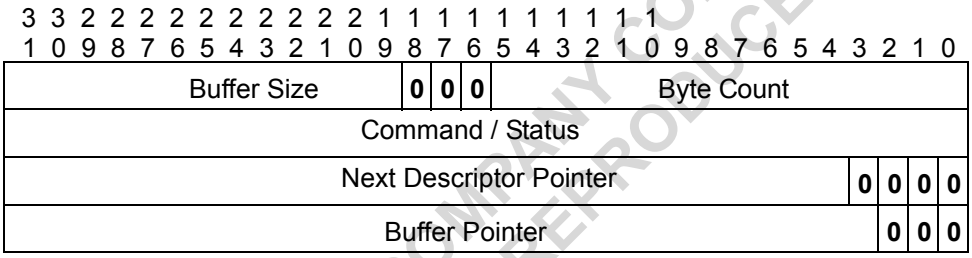
1. Read the RxCDP associated with the stopped queue.
2. If RxCDP is not NULL, it means that the error is due to a CPU owned descriptor. In this case, flip the ownership bit of the descriptor pointed by RxCDP.
3. If RxCDP is NULL, it means that the error is due to a NULL descriptor pointer. In this case, re-initialize the queue by writing a valid pointer to both RxCDP and RxFDP.

Stopping Rx DMA operation is possible using the Rx_ABORT command (see Table 564 on page 383).

When the Serial DMA fetches a descriptor from DRAM, it expects the descriptor in Little Endian format. However, since the DRAM byte order matches the CPU, consideration must be made when working with a CPU set for Big Endian.

If the software prepares the descriptors using 32-bit load/store operation, the descriptors must be word swapped when placed in DRAM. This means that the Rx descriptor must appear as described in Figure 47.

Figure 47: Word Swapped Ethernet Rx Descriptor



13.3.4 Ethernet Address Recognition

This section describes the Hash algorithm and Hash table data structure. The CPU must build this table for the GT–64240A before enabling the Ethernet port.

13.3.4.1 Hash Table Structure

The GT–64240A Hash table is a data structure prepared by the CPU and resides in the system DRAM. Its location is identified by a 32-bit pointer stored in the GT–64240A EHTP internal register (see Table 560 on page 381). The Hash table must be octet-byte aligned. The lowest three bits of the EHTP register are hard wired to ‘0’.

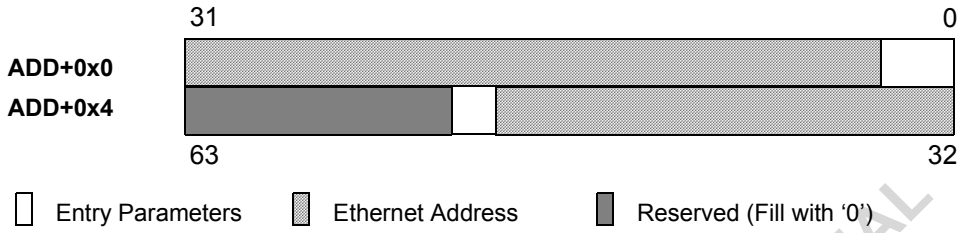
There are two possible sizes for the Hash table. Table size is selected by the HS bit in the Ethernet Configuration Register (seeTable 555 on page 373).

- 8K address table. 256KByte of DRAM required
- 1/2K address table. 16KByte of DRAM required

NOTE: The user must initialize the Hash table before enabling the Ethernet Controller.

Each Address entry is a two word data field (64 bits) as shown below:

Figure 48: Ethernet Hash Table Entry



The following table describes the Hash table entry fields.

Table 544: Hash Table Entry Fields

Bit	Command	Usage
0	Valid	Indicates Valid Entry
1	Skip	Skip empty entry in a chain
2	Receive/Discard (RD)	0 - Discard packet upon match 1 - Receive packet upon match
6:3	Ethernet Address[3:0]	Mapped to Ethernet MAC address[43:40].
11:7	Ethernet Address[7:4]	Mapped to Ethernet MAC address[47:44].
14:11	Ethernet Address[11:8]	Mapped to Ethernet MAC address[35:32].
18:15	Ethernet Address[15:12]	Mapped to Ethernet MAC address[39:36].
22:19	Ethernet Address[19:16]	Mapped to Ethernet MAC address[27:24].
26:23	Ethernet Address[23:20]	Mapped to Ethernet MAC address[31:28].
30:27	Ethernet Address[27:24]	Mapped to Ethernet MAC address[19:16].
34:31	Ethernet Address[31:28]	Mapped to Ethernet MAC address[23:20].
38:35	Ethernet Address[35:32]	Mapped to Ethernet MAC address[11:8].
42:39	Ethernet Address[39:36]	Mapped to Ethernet MAC address[15:12].
46:43	Ethernet Address[43:40]	Mapped to Ethernet MAC address[3:0].
50:47	Ethernet Address[47:44]	Mapped to Ethernet MAC address[7:4].
52:51	Priority	The priority queue of a packet sent to this Ethernet address, in case there are no other priority signals (i.e. ToS or VLAN).
63:513	Reserved	Fill With '0'

13.3.5 Hash Modes

There are two Hash functions in the GT–64240A; Hash Mode 1 and Hash Mode 0.

13.3.5.1 Hash Mode 0

In Hash mode 0, the Hash entry address is calculated in the following manner:

hashResult[14:0] = hashFunc0(ethernetADD[47:0])

- hashResult is the 15 bits Hash entry address.
- ethernetADD is a 48 bit number, which is derived from the Ethernet MAC address, by nibble swapping in every byte (i.e MAC address of 0x123456789abc translates to ethernetADD of 0x21436587a9cb).
- inverse every nibble; i.e. ethernetADD of 0x21436587a9cb translates to 0x482c6a1e593d

hashFunc0 calculates the hashResult in the following manner:

- hashResult[14:9] = ethernetADD[7:2]
- hashResult[8:0]= ethernetADD[14:8,1,0] XOR ethernetADD[23:15] XOR ethernetADD[32:24]

13.3.5.2 Hash Mode 1

In Hash mode 1, the Hash entry address is calculated in the following manner:

hashResult[14:0] = hashFunc1(ethernetADD[47:0])

- hashResult is the 15 bits Hash entry address.
- ethernetADD is a 48 bit number, which is derived from the Ethernet MAC address, by nibble swapping in every byte (i.e MAC address of 0x123456789abc translates to ethernetADD of 0x21436587a9cb).
- inverse every nibble; i.e. ethernetADD of 0x21436587a9cb translates to 0x482c6a1e593d

hashFunc1 calculates the hashResult in the following manner:

- hashResult[14:9] = ethernetADD[0:5]
- hashResult[8:0]= ethernetADD[6:14] XOR ethernetADD[15:23] XOR ethernetADD[24:32]

13.3.5.3 Hash Entry

For each Ethernet address, the Hash table entry address is the lower 13 bits of the hashResult for the 8KByte address table, or the lower 9 bits for the 0.5KByte address table. The entry is an offset from the address base and is octet-byte aligned. The address entry is therefore:

- 8K Address Table: tblEntryAdd = EHTP + {hashResult[14:0],000}
- 1/2K Address Table:tblEntryAdd = EHTP + {hashResult[10:0],000}

13.3.5.4 Hash Table Numbers

13.3.5.5 Table Filling

When preparing the Hash table data structure, the CPU must first (typically at boot time) initialize the Hash table memory to ‘0’.

The table filling algorithm is described below. The hopNumber should be selected and initialized before entering this routine. The Hash table hopNumber (Number of Hops) is 12. After 12 tries to identify an address, the GT–64240A passes the address to the CPU and sets the HE (Hash Expired) bit in the descriptor status field. There-

fore, the hopNumber is the number of times the CPU will attempt to write a newly learned Ethernet address into the Hash table.

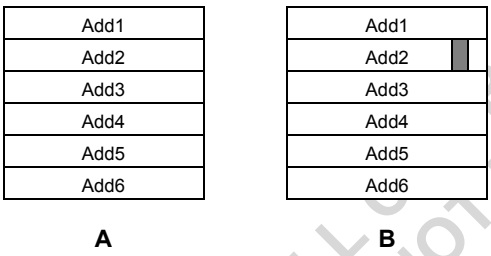
- Calculate tblEntryAdd according to mode of operation (Hash Mode 1 or Hash Mode 0).
- Check that tblEntry is empty (Valid Bit is "0").
- If the tblEntry is empty, Write the hashEntry (Valid, Skip and RD bits and Ethernet Address).
- If the tblEntry is occupied (i.e. Valid bit is 1 and Skip bit is 0), move to tblEntry+1.
- If less than hopNumber tries, Repeat to Step c.

If after hopNumber failed tries, the CPU has been unable to located a free table entry. The CPU can then:

- Defragment the table.
- Create a new Hash table using the alternate Hash Mode, which may redistribute the addresses more evenly in the table.

In cases where more than one address is mapped to the same table entry, an address chain is created. In this case, when the CPU needs to erase an address that is part of an address chain, it cannot clear its Valid bit since this would cut the chain. Instead, the CPU should set the Skip bit to ‘1’. This is shown in Figure 49.

Figure 49: Address Chain



In case A where Add1-6 has the same Hash function, and thus start with the same tblEntry, the CPU allocates them in the table by increasing tblEntry by one entry each time. Add1 is the first address to be written into the table and Add6 is the last.

When the CPU is required to remove Add2 from the table, it cannot clear its valid bit since that would break the chain from Add1 to Add3. Instead, it sets Add2’s Skip bit to ‘1’ (deonted as ▮). It is also recommended that the CPU defragments the table from time to time.

13.3.5.6 Address Recognition Process

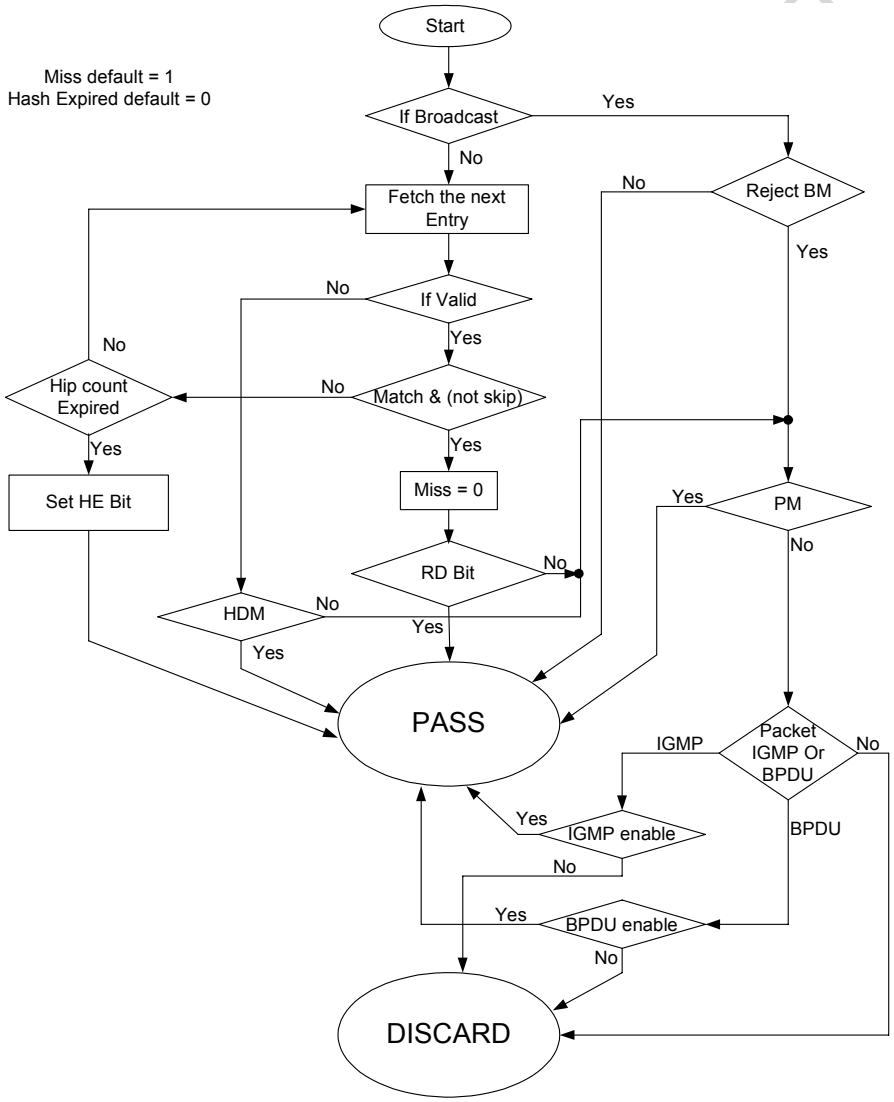
The following terms are used when referring to the address recognition process.

- **Broadcast** - The Frame MAC DA is a broadcast address.
- **Reject BM** - Determined by the configuration in the Port Configuration register’s BM bit [1], see Table 555 on page 373.
- **PM** - When enabled, all packets are passed to the CPU. The GT–64240A still executes the Hash process reporting to the CPU, regardless whether the address is in the Hash table or not. Determined by the configuration in the Port Configuration register’s PM bit [0], see Table 555 on page 373.
- **IGMP** - The Frame is Internet Group Management Protocol packets. To enable capture, see Port Configuration Extend register’s IGMP bit [0] on Table 556 on page 375
- **BPDU** -The Frame is Bridge Protocol Data unit. To enable capture, see Port Configuration Extend register’s BPDU bit [1] on Table 556 on page 375

- **Valid/Skip** - Valid hash table entry has its valid bit set or the temporary removed hash table entry has its skip bit set. See Table 544 on page 356.
- **Match** - The frame MAC DA equals the Hash table entry DA, see Table 544 on page 356.
- **RD Bit** - Receive/Discard the frame bit from the hash table entry, see Table 544 on page 356.
- **HOP Expired** - The number of Hash table entries fetched excede 12.
- **HDM** - See the Port Configuration register's HDM bit [14] (Table 555 on page 373).
- **Miss** - Address is not found in the table. Indication copied to transmit descriptor M bit [12], Table 540 on page 349.
- **HE** - Hash expired ication copied to transmit descriptor HE bit [13], Table 540 on page 349.

The GT–64240A address recognition process is illustrated by Figure 50.

Figure 50: Address Filtering Process



The GT–64240A uses the HE (Hash Expired) and M (Match) bits in the descriptor for reporting the packet filtering status. Table 545 describes the various reports and summarizes their meaning.

Table 545: Packet Filtering Status

HE	M	Condition
0	0	Hash Table No Hit The address was not found in the Hash table, but Promiscuous Mode is enabled
0	1	Hash Table Hit Either by an address found in the Hash table and RD bit set OR by an address that was not found in the Hash table, in case that HDM bit is set
1	0	Hash Table Expired The hopNumber expired before the address was found in the Hash table
1	1	UnUsed

13.4 Ethernet Port

13.4.1 Network Interface

The Ethernet port interfaces directly to a MII (Media Independent Interface) PHY compliant with the IEEE standard (please refer to IEEE 802.3u Fast Ethernet standard for detailed interface and timing information). The MII port has the following characteristics:

- Capable of supporting both 10 Mbps and 100 Mbps data rates in half or full duplex modes.
- Data and delimiters are synchronous to clock references.
- Provides independent 4-bit wide transmit and receive paths.
- Uses TTL signal levels.
- Provides a simple management interface (common to all ports).
- Capable of driving a limited length of shielded cable.

The port incorporates all the required digital circuitry to interface with a 100BaseTx, 100BaseT4, and 100BaseFX MII PHYs.

13.4.1.1 10/100 MII/RMII Compatible Interface

The port's MAC (Media Access Control) logic supports connection to a 10Mbps or 100Mbps network.

The MII interface consists of a separate nibble-wide stream for both transmit and receive data. Data transfers are clocked by the 25 MHz transmit and receive clocks in 100 Mbps operation, or by 2.5 MHz transmit and receive clocks in 10 Mbps operation. The clock inputs are driven by the PHY, which controls the clock rate according to the network connection speed.

The RMII interface consists of a separate 2bit-wide stream for both transmit and receive data. Data transfers are clocked by the 50 MHz clock in both 100 Mbps and 10 Mbps operation. The clock input is driven by an external source.

13.4.1.2 Media Access Control (MAC)

The MAC logic performs all of the functions of the 802.3 protocol such as frame formatting, frame stripping, collision handling, deferral to link traffic, etc. It also ensures that any outgoing packet complies with the 802.3 specification in terms of preamble structure - 56 preamble bits are transmitted before Start of Frame Delimiter (SFD).

The MAC operates in half duplex or full duplex modes. In half duplex mode, the MAC's transmit logic checks that there is no competitor for the network media before transmission.

In addition to waiting for idle before transmitting, the port handles collisions in a predetermined way. If two nodes attempt to transmit at the same time, the signals collide and the data on the line is garbled. The port listens while it is transmitting, and can detect a collision. If a collision is detected, 'JAM' pattern is transmitted and retransmission is delayed for a random time period determined by the Backoff algorithm. In full-duplex mode, the port transmits unconditionally.

13.4.1.3 Auto-Negotiation for Duplex Mode

The port's duplex operation mode (either half or full duplex) can be auto-negotiated or set by the CPU.

To enable auto-negotiation for duplex, the CPU must set the Port Configuration Extend register's DPLXen bit [9] to '0'. When auto-negotiation for duplex is enabled, the port decodes the duplex mode from the values of the PHY's Auto-Negotiation Advertisement register and Auto-Negotiation Link Partner Ability register at the end of the Auto-Negotiation process. Once the duplex mode is resolved, the Port Status register's Duplex bit [1] is set accordingly.

To resolve the duplex mode, the following operations are continuously performed:

1. Read the PHY's Auto-Negotiation Complete status as reported by the PHY bit 1.5 (Register 1, bit 5). If this bit is '0' switch to Half-Duplex mode and continue to read PHY register bit 1.5. Continue to step 2 when PHY bit 1.5 is '1', indicating that Auto-Negotiation is complete.

NOTE: Steps 2 through 6 are performed once for every transition of PHY bit 1.5 from '0' to '1'. Once PHY bit 1.5 remains '1' and PHY registers 4 and 5 have already been read, the port will continue to read PHY register 1, and monitor PHY bit 1.5. However, if after Rst* deassertion, the PHY bit 1.5 is already read as '1', steps 2 to 6 are performed at least once in order to update the port's duplex mode.

PHY bit 1.2 (Link Status) is read and latched during this same register read operation, regardless of the Auto-Negotiation status.

2. Read the Auto-Negotiation Advertisement register, PHY register 4. Continue to step 3.
3. Read the Auto-Negotiation Link Partner Ability register, PHY register 5. Continue to step 4.
4. Resolve the highest common ability of the two link partners in the following manner (according to the 802.3u Priority Resolution clause 28B.3):

   if (bit 4.8 AND bit 5.8) == '1' then ability is 100BASE-Tx Full Duplex

   else if (bit 4.9 AND bit 5.9) == '1' then ability is 100BASE-T4 Half Duplex

   else if (bit 4.7 AND bit 5.7) == '1' then ability is 100BASE-Tx Half Duplex

else if (bit 4.6 AND bit 5.6) == '1' then ability is 10BASE-T Full Duplex

else ability is 10BASE-T Half Duplex;

Continue to step 5.

5. Resolve the duplex mode of the two link partners in the following manner:

   if ((ability == “100BASE-Tx Full Duplex”) or (ability == “10BASE-T Full Duplex”)) then
   duplex mode = FULL DUPLEX
   else
   duplex mode = HALF DUPLEX;

   Continue to step 6.
6. Update the Port_Status register by writing the correct duplex mode bit. Continue with step 1.

13.4.1.4 Auto-Negotiation for Flow Control

Flow control mode (either enabled or disabled) can be auto-negotiated or set by the CPU. In order to enable auto-negotiation for flow-control, the CPU should set the Port Configuration Extend register’s FCTLen bit [10] (see Table 556 on page 375).

If the the Port Configuration Extend register’s FCTLen bit [10] to is set to ‘1’ , then auto-negotiation is initiated in the following cases:

- After RESET.
- After link fail (phy register 1 bit 2).

NOTE: The user may force the port to implement Flow-control by disabling auto-negotiation for flow-control and programming the Port Configuration Extend register’s FCTLen bit ‘1’.

Auto-negotiation for flow-control is done in two stages:

1. Setting Phy advertise word to support Flow Control.
   This is done by writing Phy register 4 in order to set advertise bit 10 (phy-reg4 bit 10 - Enable FC). The flow of such a cycle is:
   - Read Phy register 1. If link_status=1 and was 0 in the last cycle - continue.
   - Read Phy register 4.
   - Write Phy register 4 with bit 10 set.
2. Reading Phy Flow-Control status and determine result.
   This is done by constantly reading PHY’s register 4 and register 5 in order to determine if Flow-control is supported or not. Only if both link partners support FC (registers 4.10 and 5.10 are both SET), Port Status register’s Fctl bit [2] is set to ‘1’ (see Table 558 on page 378), and the port will send PAUSE packets when instructed to do so by the CPU. Otherwise, the Fctl bit is set to ‘0’, indicating that the support for 802.3x flow-control is disabled.

13.4.1.5 Backoff Algorithm Options

The port implements the truncated exponential Backoff algorithm defined by the 802.3 standard. Aggressiveness of the Backoff algorithm used is controlled by the Serial Parameters register’s Limit4 bit [22] (see Table 559 on page 379).

The Limit4 bit controls the number of consecutive packet collisions that will occur before the collision counter is reset.

When Limit4 feature is disabled, the port resets its collision counter after 16 consecutive retransmit trials and restarts the Backoff algorithm. Retransmission is done using the data already stored in the FIFO.

When Limit4 feature is enabled, the port will reset its collision counter and restart the Backoff algorithm after 4 consecutive transmit trials. This makes the port more aggressive in getting hold of the media following a collision. This may result better overall throughput in standardized tests.

13.4.1.6 Data Blinder

The data blinder field (DataBlind in the Serial_Parameters register) sets the period of time during which the port does not sense the wire before transmission (inhibit time). The default value is 32 bit times.

13.4.1.7 Inter Packet Gap (IPG)

IPG is the minimum idle time between transmission of any two successive packets from the same port. The default (from the standard) is 9.6uS for 10Mbps Ethernet and 960nsec for 100-Mbps Fast Ethernet.

NOTE: To make the IPG made smaller or larger than standard definition, program the Serial_Parameters register's IPG_DATA bit, see Table 559 on page 379.

13.4.1.8 10/100 Mbps MII Transmission

When the port has a frame ready for transmission, it samples link activity indicators. If the CRS signal is inactive (no activity on the link), and the Inter-packet gap (IPG) timer had expired, frame transmission begins. The data is transmitted via pins TxD[3:0] of the transmitting port, clocked on the rising edge of TxClk. The signal TxEn is asserted at this same time. In the case of collision, the PHY asserts the COL signal causing the port to stop transmitting the frame and append a jam pattern to the transmitted bit stream. At the end of a collided transmission, the port will back off and attempt to retransmit once the Backoff counter expires. Per the IEEE 802.3 specification, the clock to output delay must be a minimum of 0ns and a maximum of 25ns as shown in Figure 52.

13.4.1.9 10/100 Mbps RMII Transmission

The port starts transmission when it has a frame ready, and Inter-packet gap (IPG) timer has expired.

If in half_duplex mode, it also samples CRS_DV indicator for no activity. The data is transmitted via pins TxD[1:0] of the transmitting port, clocked on the rising edge of REF_CLK and the signal Tx_EN is asserted.

In half_duplex mode, in the case of collision (Tx_EN asserted with CRS_DV), the port stops transmitting the frame and appends a jam pattern to the transmitted bit stream. At the end of a collided transmission, the port backs off and attempts to retransmit once the Backoff counter expires. As the REF_CLK frequency is 10 times the data rate in 10 Mbps, the value on TxD[1:0] shall be valid so that it may be sampled every 10th cycle. For the RMII, transmission of each octet shall be done a di-bit at a time as per the order described in the Figure 51.

13.4.1.10 10/100 Mbps RMII Reception

Frame reception starts with the assertion of CRS_DV by the PHY. The port begins sampling incoming data on pins RxD[1:0] on the rising edge of REF_CLK. Reception ends when CRS_DV is deasserted by the PHY. The last di-bit sampled by the port is the data present on RxD[1:0] on the last REF_CLK rising edge in which CRS_DV is still asserted. CRS_DV is continuously asserted during reception. If an error is detected while CRS_DV is asserted, the decoded data is replaced in the receiving stream with "01" until the end of carrier activity. By replacing the data in the remainder of the frame, the CRC check is guaranteed to reject the packet as an error. When no reception takes place, CRS_DV should remain de-asserted. As the REF_CLK frequency is 10

times the data rate in 10 Mbps, the value of each octet shall be valid so that it may be sampled every 10th cycle. For the RMII, reception of each octet shall be done a di-bit at a time as per the order described in Figure 51.

The RMII transmission and reception of each octet is described in Figure 51.

Figure 51: RMII Di-Bit Stream

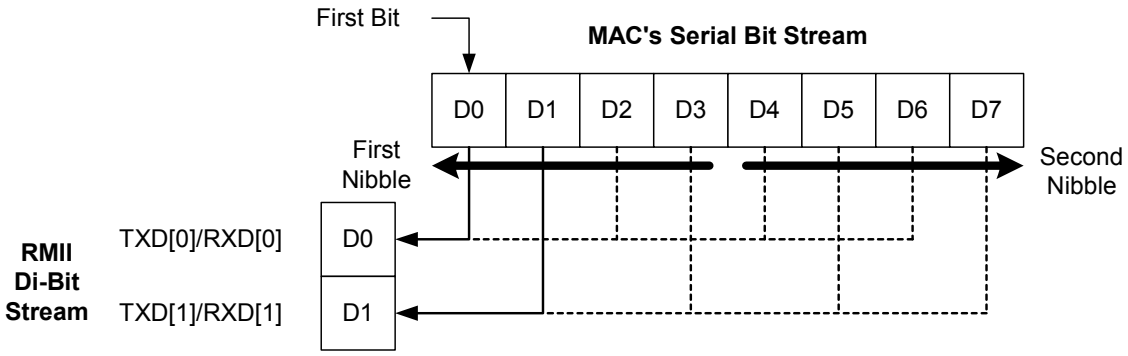
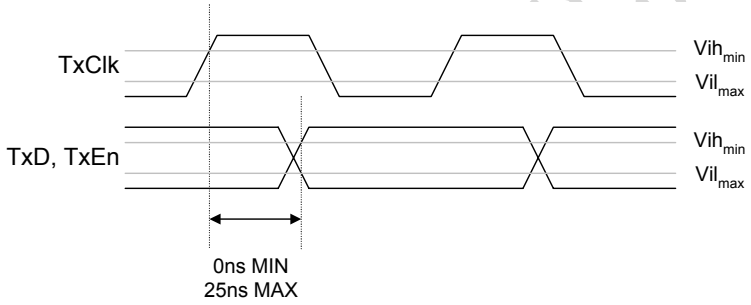


Figure 52: MII Transmit Signal Timing

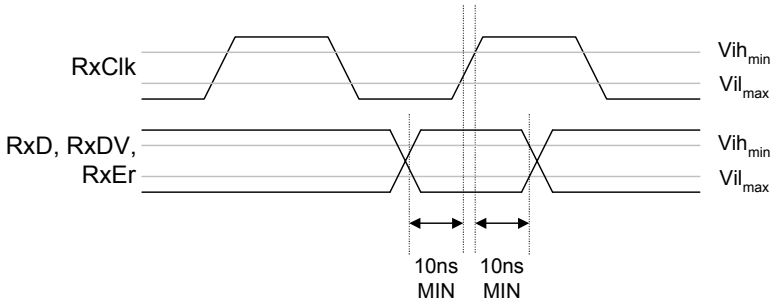


13.4.1.11 10/100 Mbps MII Reception

Frame reception starts with the assertion of CRS (while the port is not transmitting) by the PHY.

Once RxDV is asserted, the port begins sampling incoming data on pins RxD[3:0] on the rising edge of RxClk. Reception ends when RxDV is deasserted by the PHY. The last nibble sampled by the port is the nibble present on RxD[3:0] on the last RxClk rising edge in which RxDV is still asserted. During reception RxDV is continuously asserted. If, while RxDV is asserted, RxEr is asserted, it designates current packet as corrupted. When no reception takes place, RxDV should remain deasserted. The input setup time should be a minimum of 10ns and the input hold time must be a minimum of 10ns and shown in Figure 53.

Figure 53: MII Receive Signal Timing



13.4.1.12 10/100 Mbps Full-Duplex Operation

When operating in Full-duplex mode the port can transmit and receive frames simultaneously.

In full-duplex mode, the CRS signal is associated with received frames only and has no effect on transmitted frames. The Col signal is ignored while in Full-duplex mode. Transmission starts when TxEn goes active. Transmission starts regardless of the state of CRS. Reception starts when the CRS signal is asserted indicating traffic on the receive port of the PHY.

13.4.1.13 Back Pressure

The port implements a back pressure algorithm, which is only for use when the port is operating in half duplex mode. It is enabled through the Port Command register's FJ bit [15] (see Table 557 on page 378).

While in backpressure mode, the port transmits a JAM pattern for a programmable period of time (JAM_LENGTH). The IPG between two consecutive JAM patterns (or between the last transmit and the first JAM) is also a programmable value (JAM_IPG). The values are set in Serial_Parameters register.

13.4.1.14 Flow Control

IEEE 802.3x flow control is enabled while in full-duplex mode. Activating this mode is done by setting the Port Configuration Extend FCTL bit [12] (see Table 556 on page 375) or by enabling auto-negotiation for Flow-Control, see Section 13.4.1.4 "Auto-Negotiation for Flow Control" on page 362.

The port supports 802.3x flow-control (PAUSE packets, in the standard term), if it is operating in full-duplex and if the Port Configuration Extend register's FCTL bit is set to '1'.

When the port receives a PAUSE packet, it does not transmit a new packet for a period of time specified in this PAUSE packet.

A received packet is recognized as flow control PAUSE, if it was received without errors and is either of the following:

- DA = 01-80-C2-00-00-01 and type=88-08 and MAC_Control_Opcode=01
- DA = (The port address) and type=88-08 and MAC_Control_Opcode=01. The 48-bit port address is in the registers Source_Address_Low, Source_Address_High. This address is also used as source address for PAUSE packets that the port generates (to DA=01-80-C2-00-00-01)

PAUSE packets are sent by the port when instructed to do so by the CPU. This is done by setting Port Command register's FJ bit [15], see Table 557 on page 378.

13.4.2 MII Serial Management Interface (SMI)

The Ethernet unit has an integrated MII Serial Management Interface (SMI) logic for controlling MII compliant PHYs. This interface consists of two signals: serial data (MDIO); and, clock (MDC).

These signals enable control and status parameters to be passed between the PHYs and the port logic (or CPU). Multiple PHY devices can be controlled using this simple 2-pin interface.

Typically, the SMI unit continuously queries the PHY devices for their link status, without the need for CPU intervention. The PHY addresses for the link query operation are programmable per port in the PHY_Address register.

A CPU can write/read to/from all PHY addresses/registers by writing and reading to/from the SMI control register. The SMI allows the CPU to directly control a MII compatible PHY device via the SMI control register. This enables the driver software to program the PHY into specific operation mode such as Full Duplex, Loopback, Power Down, 10/100 speed selection as well as control of the PHY device's Auto-Negotiation function, if it exists. The CPU writes commands to the SMI register and the SMI unit performs the actual data transfer via MDIO, which is a bi-directional data pin. These serial data transfers are clocked by the MDC clock output.

13.4.2.1 MII Management Frame Structure

The GT–64240A's SMI cycles support the MII management frame structure.

Frames transmitted on the MII management interface have a structure that is shown in Table 546 and the order of bit transmission is from left to right.

Table 546: MII Management Frame Format

	PRE	ST	OP	PhyAd	RegAd	TA	Data	IDLE
READ	1...1	01	10	AAAAA	RRRRR	Z0	D...D(16)	Z
WRITE	1...1	01	01	AAAAA	RRRRR	10	D...D(16)	Z

The format of the bit transmission's parts is as follows:

Table 547: Bit Transmission Parts

Part	Description
PRE (Preamble)	At the beginning of each transaction, the port sends a sequence of 32 contiguous logic one bits on MDIO with 32 corresponding cycles on MDC to provide the PHY with a pattern that it can use to establish synchronization.
ST (Start of Frame)	A Start of Frame pattern of 01.
OP (Operation Code)	10 - Read; 01 - Write.
PhyAd (PHY Address)	A 5 bit address of the PHY device (32 possible addresses). The first PHY address bit transmitted by the port is the MSB of the address.

Table 547: Bit Transmission Parts

Part	Description
RegAd (Register Address)	A 5 bit address of the PHY register (32 possible registers in each PHY). The first register address bit transmitted by the port is the MSB of the address. The port always queries the PHY device for status of the link by reading register 1 bit 2.
TA (Turn Around)	The turnaround time is a 2 bit time spacing between the Register Address field and the Data field of the SMI frame to avoid contention during a read transaction. During a Read transaction the PHY should not drive MDIO in the first bit time and drive ‘0’ in the second bit time. During a write transaction, the port drives a ‘10’ pattern to fill the TA time.
Data (Data)	The data field is 16 bits long. The PHY drives the data field during Read transactions. The port drives the data field during write transactions. The first data bit transmitted and received shall be bit 15 of the PHY register being accessed.
IDLE (Idle)	The IDLE condition on MDIO is a high impedance state. The MDIO driver is disabled and the PHY should pull-up the MDIO line to a logic one.

13.4.3 SMI Timing Requirements

When the MDIO signal is driven by the PHY, it is sampled synchronously with respect to the rising edge of MDC. Per IEEE 802.3 specification, the MDC to output delay must be a minimum of 0ns and a maximum of 300ns as shown in Figure 10. Further, when the MDIO signal is driven by the port, it has a minimum of 10ns setup time and minimum of 10ns hold time as shown in Figure 54 and Figure 55.

Figure 54: MDIO Output Delay

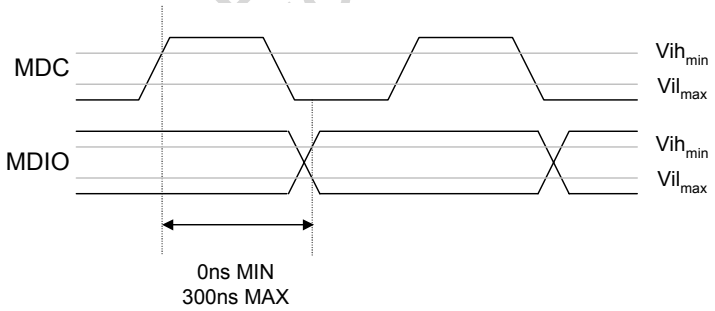
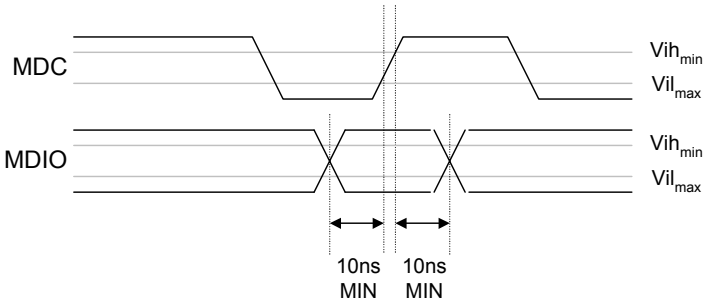


Figure 55: MDIO Setup and Hold Time



13.4.3.1 Link Detection and Link Detection Bypass (ForceLinkPass)

Typically, the port continuously queries the PHY devices for their link status without CPU intervention.

The PHY addresses used for the link query are determined by the PHY_Address register and are programmable for each port. The port alternately reads register 1 from the PHYs and updates the internal link bits according to the value of bit 2 of register 1. In the case of “link down” (i.e. bit 2 is ‘0’), that port will enter link test fail state.

In this state, all of the port’s logic is reset. The port exits from link test fail state only when the “link is up” (i.e. bit 2 of register 1 is read from the port’s PHY as ‘1’).

There is an option to disable the link detection mechanism by forcing the link state of a specific port. This is done by setting Port Configuration Extend register’s FLP bit [11](see Table 556 on page 375).

13.5 Fast Ethernet Unit Registers

Table 548: Ethernet Unit Register Map

Register	Offset	Page
Ethernet PHY Address Register (EPAR)	0x2000	page 372
Ethernet SMI Register (ESMIR)	0x2010	page 372

Table 549: Ethernet0 Register Map

Register	Offset	Page
Ethernet0 Port Configuration Register (E0PCR)	0x2400	page 373
Ethernet0 Port Configuration Extend Register (E0PCXR)	0x2408	page 375
Ethernet0 Port Command Register (E0PCMR)	0x2410	page 378
Ethernet0 Port Status Register (E0PSR)	0x2418	page 378
Ethernet0 Serial Parameters Register (E0SPR)	0x2420	page 379
Ethernet0 Hash Table Pointer Register (E0HTPR)	0x2428	page 381

Table 549: Ethernet0 Register Map (Continued)

Register	Offset	Page
Ethernet0 Flow Control Source Address Low (E0FCSAL)	0x2430	page 381
Ethernet0 Flow Control Source Address High (E0FCSAH)	0x2438	page 381
Ethernet0 SDMA Configuration Register (E0SDCR)	0x2440	page 382
Ethernet0 SDMA Command Register (E0SDCMR)	0x2448	page 383
Ethernet0 Interrupt Cause Register (E0ICR)	0x2450	page 384
Etehrnet0 Interrupt Mask Register (E0IMR)	0x2458	page 387
Ethernet0 First Rx Descriptor Pointer 0 (E0FRDP0)	0x2480	page 349
Ethernet0 First Rx Descriptor Pointer 1 (E0FRDP1)	0x2484	
Ethernet0 First Rx Descriptor Pointer 2 (E0FRDP2)	0x2488	
Ethernet0 First Rx Descriptor Pointer 3 (E0FRDP3)	0x248c	
Ethernet0 Current Rx Descriptor Pointer 0 (E0CRDP0)	0x24a0	
Ethernet0 Current Rx Descriptor Pointer 1 (E0CRDP1)	0x24a4	
Ethernet0 Current Rx Descriptor Pointer 2 (E0CRDP2)	0x24a8	
Ethernet0 Current Rx Descriptor Pointer 3 (E0CRDP3)	0x24ac	
Ethernet0 Current Tx Descriptor Pointer 0 (E0CTDP0)	0x24e0	page 345
Ethernet0 Current Tx Descriptor Pointer 1 (E0CTDP1)	0x24e4	
IP Differentiated Services CodePoint to Priority0 low (DSCP2P0L)	0x2460	page 387
IP Differentiated Services CodePoint to Priority0 high (DSCP2P0H)	0x2464	page 388
IP Differentiated Services CodePoint to Priority1 low (DSCP2P1L)	0x2468	page 388
IP Differentiated Services CodePoint to Priority1 high (DSCP2P1H)	0x246c	page 388
VLAN Priority Tag to Priority (VPT2P)	0x2470	page 388
Ethernet0 MIB Counters	0x2500 - 0x25ff	page 391

Table 550: Ethernet1 Register Map

Register	Offset	Page
Ethernet1 Port Configuration Register (E1PCR)	0x2800	page 373
Ethernet1 Port Configuration Extend Register (E1PCXR)	0x2808	page 375
Ethernet1 Port Command Register (E1PCMR)	0x2810	page 378

Table 550: Ethernet1 Register Map (Continued)

Register	Offset	Page
Ethernet1 Port Status Register (E1PSR)	0x2818	page 378
Ethernet1 Serial Parameters Register (E1SPR)	0x2820	page 379
Ethernet1 Hash Table Pointer Register (E1HTPR)	0x2828	page 381
Ethernet1 Flow Control Source Address Low (E1FCSAL)	0x2830	page 381
Ethernet1 Flow Control Source Address High (E1FCSAH)	0x2838	page 381
Ethernet1 SDMA Configuration Register (E1SDCR)	0x2840	page 382
Ethernet1 SDMA Command Register (E1SDCMR)	0x2848	page 383
Ethernet1 Interrupt Cause Register (E1ICR)	0x2850	page 384
Ethernet1 Interrupt Mask Register (E1IMR)	0x2858	page 387
Ethernet1 First Rx Descriptor Pointer 0 (E1FRDP0)	0x2880	page 349
Ethernet1 First Rx Descriptor Pointer 1 (E1FRDP1)	0x2884	
Ethernet1 First Rx Descriptor Pointer 2 (E1FRDP2)	0x2888	
Ethernet1 First Rx Descriptor Pointer 3 (E1FRDP3)	0x288c	
Ethernet1 Current Rx Descriptor Pointer 0 (E1CRDP0)	0x28a0	
Ethernet1 Current Rx Descriptor Pointer 1 (E1CRDP1)	0x28a4	
Ethernet1 Current Rx Descriptor Pointer 2 (E1CRDP2)	0x28a8	
Ethernet1 Current Rx Descriptor Pointer 3 (E1CRDP3)	0x28ac	
Ethernet1 Current Tx Descriptor Pointer 0 (E1CTDP0)	0x28e0	page 345
Ethernet1 Current Tx Descriptor Pointer 1 (E1CTDP1)	0x28e4	
IP Differentiated Services CodePoint to Priority0 low (DSCP2P0L)	0x2860	page 387
IP Differentiated Services CodePoint to Priority0 high (DSCP2P0H)	0x2864	page 388
IP Differentiated Services CodePoint to Priority1 low (DSCP2P1L)	0x2868	page 388
IP Differentiated Services CodePoint to Priority1 high (DSCP2P1H)	0x286c	page 388
VLAN Priority Tag to Priority (VPT2P)	0x2870	page 388
Ethernet1 MIB Counters	0x2900 - 0x29ff	page 391

Table 551: Ethernet2 Register Map

Register	Offset	Page
Ethernet2 Port Configuration Register (E2PCR)	0x2c00	page 373
Ethernet2 Port Configuration Extend Register (E2PCXR)	0x2c08	page 375
Ethernet2 Port Command Register (E2PCMR)	0x2c10	page 378
Ethernet2 Port Status Register (E2PSR)	0x2c18	page 378
Ethernet2 Serial Parameters Register (E2SPR)	0x2c20	page 379
Ethernet2 Hash Table Pointer Register (E2HTPR)	0x2c28	page 381
Ethernet2 Flow Control Source Address Low (E2FCSAL)	0x2c30	page 381
Ethernet2 Flow Control Source Address High (E2FCSAH)	0x2c38	page 381
Ethernet2 SDMA Configuration Register (E2SDCR)	0x2c40	page 382
Ethernet2 SDMA Command Register (E2SDCMR)	0x2c48	page 383
Ethernet2 Interrupt Cause Register (E2ICR)	0x2c50	page 384
Ethernet2 Interrupt Mask Register (E2IMR)	0x2c58	page 387
Ethernet2 First Rx Descriptor Pointer 0 (E2FRDP0)	0x2c80	page 349
Ethernet2 First Rx Descriptor Pointer 1 (E2FRDP1)	0x2c84	
Ethernet2 First Rx Descriptor Pointer 2 (E2FRDP2)	0x2c88	
Ethernet2 First Rx Descriptor Pointer 3 (E2FRDP3)	0x2c8c	
Ethernet2 Current Rx Descriptor Pointer 0 (E2CRDP0)	0x2ca0	
Ethernet2 Current Rx Descriptor Pointer 1 (E2CRDP1)	0x2ca4	
Ethernet2 Current Rx Descriptor Pointer 2 (E2CRDP2)	0x2ca8	
Ethernet2 Current Rx Descriptor Pointer 3 (E2CRDP3)	0x2cac	
Ethernet2 Current Tx Descriptor Pointer 0 (E2CTDP0)	0x2ce0	page 345
Ethernet2 Current Tx Descriptor Pointer 1 (E2CTDP1)	0x2ce4	
IP Differentiated Services CodePoint to Priority0 low (DSCP2P0L)	0x2c60	page 387
IP Differentiated Services CodePoint to Priority0 high (DSCP2P0H)	0x2c64	page 388
IP Differentiated Services CodePoint to Priority1 low (DSCP2P1L)	0x2c68	page 388
IP Differentiated Services CodePoint to Priority1 high (DSCP2P1H)	0x2c6c	page 388
VLAN Priority Tag to Priority (VPT2P)	0x2c70	page 388
Ethernet2 MIB Counters	0x2d00 - 0x2dff	page 391

Table 552: IP Differentiated Services Register Map

Register	Offset	Page
IP Differentiated Services CodePoint to Priority0 low (DSCP2P0L)	0x2c60	page 387
IP Differentiated Services CodePoint to Priority0 high (DSCP2P0H)	0x2c68	page 388
IP Differentiated Services CodePoint to Priority1 low (DSCP2P1L)	0x2c70	page 388
IP Differentiated Services CodePoint to Priority1 high (DSCP2P1H)	0x2c78	page 388
VLAN Priority Tag to Priority (VPT2P)	0x2c80	page 388

13.5.1 Ethernet Unit Registers

Table 553: PHY Address Register, Offset: 0x2000

Bits	Field Name	Function	Initial Value
4:0	PhyAD0	PHY device address for port 0.	0x4
9:5	PhyAD1	PHY device address for port 1.	0x5
14:10	PhyAD2	PHY device address for port 2.	0x6
31:15	Reserved	Reserved.	0x0

Table 554: SMI Register (SMIR), Offset: 0x2010

Bits	Field Name	Function	Initial Value
15:0	Data	• SMI READ operation Two transactions are required: (1) write to the SMI register with OpCode = 1, PhyAd, RegAd with the Data being any value; (2) read from the SMI register. When reading back the SMI register, the Data is the addressed Phy register contents if the ReadValid bit (#27) is '1'. The Data remains undefined as long as ReadValid is 0. • SMI WRITE operation One transaction is required. Write to the SMI register with OpCode = 0, PhyAd, RegAd with the Data to be written to the addressed Phy register.	0x0
20:16	PhyAd	PHY Device Address	0x0
25:21	RegAd	PHY Device Register Address	0x0

Table 554: SMI Register (SMIR), Offset: 0x2010

Bits	Field Name	Function	Initial Value
26	OpCode	0 - Write 1 - Read	0x0
27	ReadValid	1 - Indicates that the Read operation has been completed for the addressed RegAd register and that the data is valid on the Data field.	0x0
28	Busy	1 - Indicates that an operation is in progress and that CPU must not write to the SMI register at this time.	0x0
31:29	N/A	Must be written as '0' for any write to the SMI register.	0x0

13.5.2 Ethernet Registers

Table 555: Port Configuration Register (PCR)

- Ethernet0 Offset: 0x2400
- Ethernet1 Offset: 0x2800
- Ethernet2 Offset: 0x2c00

Bits	Field Name	Function	Initial Value
0	PM	Promiscuous mode 0 - Normal mode Frames are only received if the destination address is found in the hash table. 1 - Promiscuous mode Frames are received regardless of their destination address. Errored frames are discarded unless the Port Configuration register's PBF bit is set.	0x0
1	RBM	Reject Broadcast Mode 0 - Receive broadcast address 1 - Reject frames with broadcast address Overridden by the promiscuous mode.	0x0
2	PBF	Pass Bad Frames 0 - Normal mode 1 - Pass bad Frames The Ethernet receiver passes to the CPU errored frames (like fragments and collided packets) that are normally rejected. **NOTE:** Frames are only passed if they successfully pass address filtering.	0x0
6:3	Reserved	Reserved.	0x0

Table 555: Port Configuration Register (PCR) (Continued)

- Ethernet0 Offset: 0x2400
- Ethernet1 Offset: 0x2800
- Ethernet2 Offset: 0x2c00

Bits	Field Name	Function	Initial Value
7	EN	Enable 0 - Disabled 1 - Enable When enabled, the ethernet port is ready to transmit/receive.	0x0
9:8	LPBK	Loop Back Mode 00 - Normal mode 01 - Internal loop back mode Tx data is looped back to the Rx lines. No transition is seen on the interface pins. 10 - External loop back mode Tx data is looped back to the Rx lines and also transmitted out to the MII interface pins. 11 - Reserved	0x0
10	FC	Force Collision 0 - Normal mode. 1 - Force Collision on any Tx frame. For RxM test (in Loop-back mode).	0x0
11		Reserved.	0x0
12	HS	Hash Size 0 - 8K address filtering (256KB of memory space required). 1 - 1/2K address filtering (16KB of memory space required).	0x0
13	HM	Hash Mode 0 - Hash Function 0 1 - Hash Function 1	0x0
14	HDM	Hash Default Mode 0 - Discard addresses ***not found*** in address table. 1 - Pass addresses ***not found*** in address table.	0x0
15	HD	Duplex Mode 0 - Half Duplex 1 - Full Duplex **NOTE:** Valid only when auto-negotiation for duplex mode is disabled.	0x0
27:16	Reserved	Reserved.	0x0

Table 555: Port Configuration Register (PCR) (Continued)

- Ethernet0 Offset: 0x2400
- Ethernet1 Offset: 0x2800
- Ethernet2 Offset: 0x2c00

Bits	Field Name	Function	Initial Value
30:28	ISLEn	0x6 ISL Enabled 0x0 ISL Disabled **NOTE:** When ISL is enabled, bit 31 must set to 0.	
31	ACCS	Accelerate Slot Time 0 - Normal mode 1 - Reserved	0x0

Table 556: Port Configuration Extend Register (PCXR)

- Ethernet0 Offset: 0x2408
- Ethernet1 Offset: 0x2808
- Ethernet2 Offset: 0x2c08

Bits	Field Name	Function	Initial Value
0	IGMP	IGMP Packets Capture Enable 0 - IGMP packets are treated as normal Multicast packets. 1 - IGMP packets on IPv4/Ipv6 over Ethernet/802.3 are trapped and sent to high priority Rx queue.	0x0
1	BPDU	BPDU Capture Enable 0 - BPDU (Bridge Protocol Data Unit) packets are treated as normal Multicast packets. 1 - BPDU packets are trapped and sent to high priority Rx queue.	0x0
2	PAR	Partition Enable When more than 61 collisions occur while transmitting, the port enters Partition mode. It waits for the first good packet from the wire and then goes back to Normal mode. Under Partition mode it continues transmitting, but it does not receive. 0 - Normal mode 1 - Partition mode	0x0

Table 556: Port Configuration Extend Register (PCXR) (Continued)

- Ethernet0 Offset: 0x2408
- Ethernet1 Offset: 0x2808
- Ethernet2 Offset: 0x2c08

Bits	Field Name	Function	Initial Value
5:3	PRIOtx	Priority weight in the round-robin between high and low priority Tx queues. 000 - 1 pkt transmitted from HIGH, 1 pkt from LOW. 001 - 2 pkt transmitted from HIGH, 1 pkt from LOW. 010 - 4 pkt transmitted from HIGH, 1 pkt from LOW. 011 - 6 pkt transmitted from HIGH, 1 pkt from LOW. 100 - 8 pkt transmitted from HIGH, 1 pkt from LOW. 101 - 10 pkt transmitted from HIGH, 1 pkt from LOW. 110 - 12 pkt transmitted from HIGH, 1 pkt from LOW. 111 - All pkt transmitted from HIGH, 0 pkt from LOW. LOW is served only if HIGH is empty. **NOTE:** If the HIGH queue is emptied before finishing the count, the count is reset until the next first HIGH comes in.	0x0
7:6	PRIOrx	Default Priority for Packets Received on this Port 00 - Lowest priority 11 - Highest priority	0x0
8	PRIOrx_ Override	Override Priority for Packets Received on this Port 0 - Do not override 1 - Override with <PRIOrx> field	0x0
9	DPLXen	Enable Auto-negotiation for Duplex Mode 0 - Enable 1 - Disable	0x0
10	FCTLen	Enable Auto-negotiation for 802.3x Flow-control 0 - Enable When enabled, ‘1’ is written (through SMI access) to the PHY’s register 4 bit 10 to advertise flow-control capability. 1 - Disable Only enables flow control after the PHY address is set by the CPU. When changing the PHY address the flow control auto-negotiation must be disabled.	0x1
11	FLP	Force Link Pass 0 - Force Link Pass 1 - Do NOT Force Link pass	0x1

Table 556: Port Configuration Extend Register (PCXR) (Continued)

- Ethernet0 Offset: 0x2408
- Ethernet1 Offset: 0x2808
- Ethernet2 Offset: 0x2c08

Bits	Field Name	Function	Initial Value
12	FCTL	Flow-Control Mode 0 - Disable IEEE 802.3x flow-control 1 - Enable IEEE 802.3x flow-control **NOTE:** Only valid when auto negotiation for flow control is disabled.	0x0
13	Reserved	Reserved.	0x0
15:14	MFL	Max Frame Length Maximum packet allowed for reception (including CRC): 00 - 1518 bytes 01 - 1536 bytes 10 - 2048 bytes 11 - 64K bytes	0x0
16	MIBclrMode	MIB Counters Clear Mode 0 - Clear 1 - No effect	0x0
17	MIBctrMode	Reserved.	0x0
18	Speed	Port Speed 0 - 10Mbit/Sec 1 - 100Mbit/Sec **NOTE:** Only valid if SpeedEn bit is set.	0x0
19	SpeedEn	Enable Auto-negotiation for Speed 0 - Enable 1 - Disable	0x0
20	RMIIen	RMII enable 0-Port functions as MII port 1-Port functions as RMII port	0x0
21	DSCPen	DSCP enable 0-IP DSCP field decoding is disabled. 1-IP DSCP field decoding is enabled.	0x0
31:22	Reserved	Reserved.	0x0

Table 557: Port Command Register (PCMR)

- Ethernet0 Offset: 0x2410
- Ethernet1 Offset: 0x2810
- Ethernet2 Offset: 0x2c10

Bits	Field Name	Function	Initial Value
14:0	Reserved	Reserved.	0x0
15	FJ	Force Jam / Flow Control When in half-duplex mode, the CPU uses this bit to force collisions on the Ethernet segment. When the CPU recognizes that it is going to run out of receive buffers, it can force the transmitter to send jam frames, forcing collisions on the wire. To allow transmission on the Ethernet segment, the CPU must clear the FJ bit when more resources are available. When in full-duplex and flow-control is enabled, this bit causes the port's transmitter to send flow-control PAUSE packets. The CPU must reset this bit when more resources are available.	0x0
31:16	Reserved	Reserved.	0x0

Table 558: Port Status Register (PSR)

- Ethernet0 Offset: 0x2418
- Ethernet1 Offset: 0x2818
- Ethernet2 Offset: 0x2c18

Bits	Field Name	Function	Initial Value
0	Speed	Indicates Port Speed 0 - 10Mbit/s 1 - 100Mbit/s **NOTE:** This bit is read-only.	0x0
1	Duplex	Indicates Port Duplex Mode 0 - Half duplex 1 - Full duplex **NOTE:** This bit is read-only.	0x0
2	Fctl	Indicates Flow-control Mode 0 - Flow-control mode enabled. 1 - Flow-control mode disabled. **NOTE:** This bit is read-only.	0x0
3	Link	Indicates Link Status 0 - Link is down 1 - Link is up **NOTE:** This bit is read-only.	0x0

Table 558: Port Status Register (PSR) (Continued)

- Ethernet0 Offset: 0x2418
- Ethernet1 Offset: 0x2818
- Ethernet2 Offset: 0x2c18

Bits	Field Name	Function	Initial Value
4	Pause	Indicates that the port is in flow-control disabled state. This bit is set when an IEEE 802.3x flow-control PAUSE (XOFF) packet is received (assuming that flow-control is enabled and the port is in full-duplex mode). Reset when XON is received, or when the XOFF timer has expired. **NOTE:** This bit is read-only.	0x0
5	TxLow	Tx Low Priority Status Indicates the status of the low priority transmit queue: 0 - Stopped 1 - Running **NOTE:** This bit is read-only.	0x0
6	TxHigh	Tx High Priority Status Indicates the status of the high priority transmit queue: 0 - Stopped 1 - Running **NOTE:** This bit is read-only.	0x0
7	TxinProg	Tx in Progress Indicates that the port's transmitter is in an active transmission state. **NOTE:** This bit is read-only.	0x0
31:8	Reserved	Reserved.	0x0

Table 559: Serial Parameters Register (SPR)

- Ethernet0 Offset: 0x2420
- Ethernet1 Offset: 0x2820
- Ethernet2 Offset: 0x2c20

Bits	Field Name	Function	Initial Value
1:0	JAM_LENGTH	Two bits to determine the JAM Length (in Backpressure) as follows: 00 = 12K bit-times 01 = 24K bit-times 10 = 32K bit-times 11 = 48K bit-times	11 (48K bit time)

Table 559: Serial Parameters Register (SPR) (Continued)

- Ethernet0 Offset: 0x2420
- Ethernet1 Offset: 0x2820
- Ethernet2 Offset: 0x2c20

Bits	Field Name	Function	Initial Value
6:2	JAM_IPG	Five bits to determine the JAM IPG. The step is four bit-times. The JAM IPG varies between 4 bit time to 124.	01000 (32 bit time)
11:7	IPG_JAM_TO_DATA	Five bits to determine the IPG JAM to DATA. The step is four bit-times. The value may vary between 4 bit time to 124.	10000 (64 bit time)
16:12	IPG_DATA	Inter-Packet Gap (IPG) The step is four bit-times. The value may vary between 12 bit time to 124. **NOTE:** These bits may be changed only when the Ethernet ports is disabled.	11000 (96 bit time)
21:17	Data_Blind	Data Blinder The number of nibbles from the beginning of the IPG, in which the IPG counter is restarted when detecting a carrier activity. Following this value, the port enters the Data Blinder zone and does not reset the IPG counter. This ensures fair access to the medium. Value must be written in hexadecimal format. The default is 10 hex (64 bit times - 2/3 of the default IPG). The step is 4 bit-times. Valid range is 3 to 1F hex nibbles. **NOTE:** These bits may be only changed when the Ethernet port is disabled.	10000 (64 bit time)
22	Limit4	The number of consecutive packet collisions that occur before the collision counter is reset. 0- The port resets its collision counter after 16 consecutive retransmit trials and restarts the Backoff algorithm. 1- The port resets its collision counter and restarts the Back-off algorithm after 4 consecutive transmit trials.	0x0
31:23	Reserved	Reserved.	0x0

Table 560: Hash Table Pointer Register (HTPR)

- Ethernet0 Offset: 0x2428
- Ethernet1 Offset: 0x2828
- Ethernet2 Offset: 0x2c28

Bits	Field Name	Function	Initial Value
31:0	HTP	32-bit pointer to the address table. Bits [2:0] must be set to zero.	0x0

Table 561: Flow Control Source Address Low (FCSAL)

- Ethernet0 Offset: 0x2430
- Ethernet1 Offset: 0x2830
- Ethernet2 Offset: 0x2c30

Bits	Field Name	Function	Initial Value
15:0	SA[15:0]	Source Address The least significant bits of the source address for the port. This address is used for Flow Control.	0x0
31:16	Reserved	Reserved.	0x0

Table 562: Flow Control Source Address High (FCSAH)

- Ethernet0 Offset: 0x2438
- Ethernet1 Offset: 0x2838
- Ethernet2 Offset: 0x2c38

Bits	Field Name	Function	Initial Value
31:0	SA[47:16]	Source Address The most significant bits of the source address for the port. This address is used for Flow Control.	0x0

Table 563: SDMA Configuration Register (SDCR)

- Ethernet0 Offset: 0x2440
- Ethernet1 Offset: 0x2840
- Ethernet2 Offset: 0x2c40

Bits	Field Name	Function	Initial Value
1:0	Reserved	Reserved.	0x0
5:2	RC	Retransmit Count Sets the maximum number of retransmits per packet. After executing retransmit for RC times, the Tx SDMA closes the descriptor with a Retransmit Limit error indication and processes the next packet. When RC is set to '0', the number of retransmits is unlimited. In this case, the retransmit process is only terminated if CPU issues an Abort command.	0x0
6	BLMR	Big/Little Endian Receive Mode The DMA supports Big or Little Endian configurations on a per channel basis. The BLMR bit only affects data transfer to memory. 0 - Big Endian 1 - Little Endian	0x1
7	BLMT	Big/Little Endian Transmit Mode The DMA supports Big or Little Endian configurations on a per channel basis. The BLMT bit only affects data transfer from memory. 0 - Big Endian 1 - Little Endian	0x1
8	POVR	PCI Override When set, causes the SDMA to direct all its accesses in PCI_0 direction and overrides normal address decoding process.	0x0
9	RIFB	Receive Interrupt on Frame Boundaries When set, the SDMA Rx generates interrupts only on frame boundaries (i.e. after writing the frame status to the descriptor).	0x0
11:10	Reserved	Reserved.	0x0
13:12	BSZ	Burst Size Sets the maximum burst size for SDMA transactions: 00 - Burst is limited to 1 64bit words. 01 - Burst is limited to 2 64bit words. 10 - Burst is limited to 4 64bit words. 11 - Burst is limited to 8 64bit words.	0x0
31:14	Reserved	Reserved.	0x0

Table 564: SDMA Command Register (SDCMR)

- Ethernet0 Offset: 0x2448
- Ethernet1 Offset: 0x2848
- Ethernet2 Offset: 0x2c48

Bits	Field Name	Function	Initial Value
6:0	Reserved	Reserved.	0x0
7	ERD	Enable Rx DMA. Set to ‘1’ by the CPU to cause the SDMA to start a receive process. Cleared when the CPU issues an Abort Receive command.	0x0
14:8	Reserved	Reserved.	0x0
15	AR	Abort Receive Set to ‘1’ by the CPU to abort a receive SDMA operation. When the AR bit is set, the SDMA aborts its current operation and moves to IDLE. No descriptor is closed. The AR bit is cleared upon entering IDLE. After setting the AR bit, the CPU must poll the bit to verify that the abort sequence is completed.	0x0
16	STDH	Stop Tx High Set to ‘1’ by the CPU to stop the transmission process from the high priority queue at the end of the current frame. An interrupt is generated when the stop command has been executed. Writing ‘1’ to STDH resets TxDH bit. Writing ‘0’ to this bit has no effect.	0x0
17	STDL	Stop Tx Low Set to ‘1’ by the CPU to stop the transmission process from the low priority queue at the end of the current frame. An interrupt is generated when the stop command has been executed. Writing ‘1’ to STDL resets TxDL bit. Writing ‘0’ to this bit has no effect.	0x0
22:18	Reserved	Reserved.	0x0
23	TxDH	Start Tx High Set to ‘1’ by the CPU in order to cause the SDMA to fetch the first descriptor and start a transmit process from the high priority Tx queue. Writing ‘1’ to TxDH resets STDH bit. Writing ‘0’ to this bit has no effect.	0x0

Table 564: SDMA Command Register (SDCMR) (Continued)

- Ethernet0 Offset: 0x2448
- Ethernet1 Offset: 0x2848
- Ethernet2 Offset: 0x2c48

Bits	Field Name	Function	Initial Value
24	TxDL	Start Tx Low Set to '1' by the CPU to cause the SDMA to fetch the first descriptor and start a transmit process from the low priority Tx queue. Writing '1' to TxDL resets STDL bit. Writing '0' to this bit has no effect.	0x0
30:25	Reserved	Reserved.	0x0
31	AT	Abort Transmit Set to '1' by the CPU to abort a transmit DMA operation. When the AT bit is set, the SDMA aborts its current operation and moves to IDLE. No descriptor is closed. Cleared upon entering IDLE. After setting AT bit, the CPU must poll it in order to verify that the abort sequence is completed.	0x0

Table 565: Interrupt Cause Register (ICR)

- Ethernet0 Offset: 0x2450
- Ethernet1 Offset: 0x2850
- Ethernet2 Offset: 0x2c50

Bits	Field Name	Function	Initial Value
0	RxBuffer	Rx Buffer Return Indicates an Rx buffer returned to CPU ownership or that the port finished reception of a Rx frame in either priority queues. **NOTE:** In order to get a Rx Buffer return per priority queue, use bit 19:16. This bit is set upon closing any Rx descriptor which has its EI bit set. To limit the interrupts to frame (rather than buffer) boundaries, the user must set SDMA Configuration register's RIFB bit. When the RIFB bit is set, an interrupt generates only upon closing the first descriptor of a received packet, if this descriptor has it EI bit set.	0x0
1	Reserved	Reserved.	0x0

Table 565: Interrupt Cause Register (ICR) (Continued)

- Ethernet0 Offset: 0x2450
- Ethernet1 Offset: 0x2850
- Ethernet2 Offset: 0x2c50

Bits	Field Name	Function	Initial Value
2	TxBufferHigh	Tx Buffer for High priority Queue Indicates a Tx buffer returned to CPU ownership or that the port finished transmission of a Tx frame. **NOTE:** This bit is set upon closing any Tx descriptor which has its EI bit set. To limit the interrupts to frame (rather than buffer) boundaries, the user must set EI only in the last descriptor.	0x0
3	TxBufferLow	Tx Buffer for Low Priority Queue Indicates a Tx buffer returned to CPU ownership or that the port finished transmission of a Tx frame. **NOTE:** This bit is set upon closing any Tx descriptor which has its EI bit set. To limit the interrupts to frame (rather than buffer) boundaries, the user must set EI only in the last descriptor.	0x0
5:4	Reserved	Reserved.	0x0
6	TxEndHigh	Tx End for High Priority Queue Indicates that the Tx DMA stopped processing the high priority queue after stop command, or that it reached the end of the high priority descriptor chain.	0x0
7	TxEndLow	Tx End for Low Priority Queue Indicates that the Tx DMA stopped processing the low priority queue after stop command, or that it reached the end of the low priority descriptor chain.	0x0
8	RxError	Rx Resource Error Indicates a Rx resource error event in either priority queues. **NOTE:** To get a Rx Resource Error Indication per priority queue, use bit 23:20. event	0x0
9	Reserved	Reserved.	0x0
10	TxErrorHigh	Tx Resource Error for High Priority Queue Indicates a Tx resource error event during packet transmission from the high priority queue.	0x0
11	TxErrorLow	Tx Resource Error for Low Priority Queue Indicates a Tx resource error event during packet transmission from the low priority queue.	0x0
12	RxOVR	Rx Overrun Indicates an overrun event that occurred during reception of a packet.	0x0

Table 565: Interrupt Cause Register (ICR) (Continued)

- Ethernet0 Offset: 0x2450
- Ethernet1 Offset: 0x2850
- Ethernet2 Offset: 0x2c50

Bits	Field Name	Function	Initial Value
13	TxUdr	Tx Underrun Indicates an underrun event that occurred during transmission of packet from either queue.	0x0
15:14	Reserved	Reserved.	0x0
16	RxBuffer Queue[0]	Rx Buffer Return in Priority Queue[0] Indicates a Rx buffer returned to CPU ownership or that the port completed reception of a Rx frame in a receive priority queue[0]	0x0
17	RxBuffer Queue[1]	Rx Buffer Return in Priority Queue[1] Indicates a Rx buffer returned to CPU ownership or that the port completed reception of a Rx frame in a receive priority queue[1].	0x0
18	RxBuffer-Queue[2]	Rx Buffer Return in Priority Queue[2] Indicates a Rx buffer returned to CPU ownership or that the port completed reception of a Rx frame in a receive priority queue[2].	0x0
19	RxBuffer-Queue[3]	Rx Buffer Return in Priority Queue[3] Indicates a Rx buffer returned to CPU ownership or that the port completed reception of a Rx frame in a receive priority queue[3].	0x0
20	RxError-Queue[0]	Rx Resource Error in Priority Queue[0] Indicates a Rx resource error event in receive priority queue[0].	0x0
21	RxError-Queue[1]	Rx Resource Error in Priority Queue[1] Indicates a Rx resource error event in receive priority queue[1].	0x0
22	RxError-Queue[2]	Rx Resource Error in Priority Queue[2] Indicates a Rx resource error event in receive priority queue[2].	0x0
23	RxError-Queue[3]	Rx Resource Error in Priority Queue[3] Indicates a Rx resource error event in receive priority queue[3].	0x0
27:24	Reserved	Reserved.	0x0

Table 565: Interrupt Cause Register (ICR) (Continued)

- Ethernet0 Offset: 0x2450
- Ethernet1 Offset: 0x2850
- Ethernet2 Offset: 0x2c50

Bits	Field Name	Function	Initial Value
28	MIIPhySTC	MII PHY Status Change Indicates a status change reported by the PHY connected to this port. Set when the MII management interface block identifies a change in PHY's register 1.	0x0
29	SMIdone	SMI Command Done Indicates that the SMI completed a MII management command (either read or write) that was initiated by the CPU writing to the SMI register.	0x0
30	Reserved	Reserved.	0x0
31	EtherIntSum	Ethernet Interrupt Summary This bit is a logical OR of the (unmasked) bits [30:4] in the Interrupt Cause register.	0x0

Table 566: Interrupt Mask Register (IMR)

- Ethernet0 Offset: 0x2458
- Ethernet1 Offset: 0x2858
- Ethernet2 Offset: 0x2c58

Bits	Field Name	Function	Initial Value
31:0	Various	Mask bits for the Interrupt Cause register.	0x0

13.5.3 IP Differentiated Service Registers

Table 567: IP Differentiated Services CodePoint to Priority0 low (DSCP2P0L)

- Ethernet0 Offset: 0x2460
- Ethernet1 Offset: 0x2860
- Ethernet2 Offset: 0x2c60

Bits	Field Name	Function	Initial Value
31:0	Priority0 low	The LSB priority bits for DSCP[31:0] entries.	0x0

Table 568: IP Differentiated Services CodePoint to Priority0 high (DSCP2P0H)

- Ethernet0 Offset: 0x2464
- Ethernet1 Offset: 0x2864
- Ethernet2 Offset: 0x2c64

Bits	Field Name	Function	Initial Value
31:0	Priority0 high	The LSB priority bits for DSCP[63:32] entries.	0x0

Table 569: IP Differentiated Services CodePoint to Priority1 low (DSCP2P1L)

- Ethernet0 Offset: 0x2468
- Ethernet1 Offset: 0x2868
- Ethernet2 Offset: 0x2c68

Bits	Field Name	Function	Initial Value
31:0	Priority1 low	The MSB priority bits for DSCP[31:0] entries.	0x0

Table 570: IP Differentiated Services CodePoint to Priority1 high (DSCP2P1H)

- Ethernet0 Offset: 0x246c
- Ethernet1 Offset: 0x286c
- Ethernet2 Offset: 0x2c6c

Bits	Field Name	Function	Initial Value
31:0	Priority1 high	The MSB priority bit for DSCP[63:32] entries.	0x0

Table 571: VLAN Priority Tag to Priority (VPT2P)

- Ethernet0 Offset: 0x2470
- Ethernet1 Offset: 0x2870
- Ethernet2 Offset: 0x2c70

Bits	Name	Description	Initial Value
7:0	Priority0	The LSB priority bits for VLAN Priority[7:0] entries.	0xcc
15:8	Priority1	The MSB priority bits for VLAN Priority[7:0] entries.	0xf0
31:16	Reserved	Reserved.	0x0

13.5.4 Defining a Priority Queue to the IP DSCP or VLAN Entry

To define a priority queue to the IP DSCP or VLAN entry, the entry's priority 0 and priority 1 bits must be defined.

Table 572 and Table 573 describe the writing of IP DSCP and VLAN entries respectively for a few set examples. Table 574 describes three example cases for mixed priority queueing.

Table 572: Writing IP DSCP Priority Example

IP DSCP Value	Priority MSB Bit	Priority LSB Bit
0	DSCP2P1L[0	DSCP2P0L[0
16	DSCP2P1L[16]	DSCP2P0L[16]
31	DSCP2P1L[31]	DSCP2P0L[31]
32	DSCP2P1H[0]	DSCP2P0H[0]
48	DSCP2P1H[16]	DSCP2P0H[16]
63	DSCP2P1H[31]	DSCP2P0H[31]

Table 573: Writing VLAN Priority Example

VLAN Priority Value	Priority MSB Bit	Priority LSB Bit
0	VPT2P[8]	VPT2P[0]
4	VPT2P[12]	VPT2P[4]
7	VPT2P[15]	VPT2P[7]

Table 574: Writing IP DSCP and VLAN Priority Example

Case	IPDSCP	All Others	VLAN Tag Packets
A	0x0 and 0x3f (63) directed to priority queue 3	directed to priority queue 0	ignored
B	<0x20 (32) directed to priority queue 2	directed to priority queue 1	directed to priority queue 3
C	0x1f (31) and 0x20 (32) directed to priority queue 3 0x0 and 0x3f (63)directed to priority queue 0<0x1f (31) directed to priority queue 1	directed to priority queue 2	>3 and IP DSCP ≠ 0x1f or 0x20 directed to priority queue 2 other tags are ignored

Table 575: Writing IP DSCP and VLAN Priority Register mapping Example

Register	Case A	Case B	Case C
2P0L	0x00000001	0x00000000	0xFFFFFFFE
DSCP2P0H	0x80000000	0xFFFFFFFF	0x00000001
DSCP2P1L	0x00000001	0xFFFFFFFF	0x80000000

Table 575: Writing IP DSCP and VLAN Priority Register mapping Example (Continued)

Register	Case A	Case B	Case C
DSCP2P1H	0x80000000	0x00000000	0x7FFFFFFF
VPT2P	0x00000001	0x0000FFFF	0x0000F000

13.5.5 Ethernet MIB Counters

The Ethernet unit includes a set of counters that are used to count events occurring on the segment to which the port is connected to. All counters are 32 bit wide.

The CPU must read all the MIB counters during initialization to reset the counters to '0'. If the Port Configuration Extend register's MIBclrMode bit [16] is set to '0' (default), the counters are set to '0'. If MIBclrMode bit is set to '1', reading the MIB counters has no effect on their value.

NOTE: Table 576 lists definitions of terms used in the counter descriptions.

Table 576: Terms Used in MIB Counters Descriptions

Term	Definition
Packet Data Section	All data bytes in the packet following the SFD until the end of the packet.
Packet Data Length	The number of data bytes in the packet data section.
Data Octet	A single byte from the packet data section.
Nibble	4 bits (half byte) of a data octet.
Misaligned Packet	A packet with an odd number of nibbles.
Received Good Packet	A received packet which is well formed.
Received Bad Packet	A received packet which has an error such as bad CRC, Rx Error Event, Invalid size (too short or too long).
Transmitted Packet	Any transmitted packet (not including collision fragments).
Collision Event	Any collision event that is indicated by assertion of MII_COL signal within the collision window interval.
Late Collision Event	Any collision event that is indicated by assertion of MII_COL signal outside the collision window interval.
Rx Error Event	An error event that is indicated by assertion of MII_Rx_ERR signal.
Dropped Packet	A received packet which is dropped by the port due to lack of resources (e.g. no Rx buffers available).
MIBctrMode	MIBctrMode bit in the Port Configuration Extend register.
MaxFrameSize	1518, 1536, 2 K or 64Kbytes depending on the setting in the Port Configuration Extend register.

Table 577: Ethernet MIB Counters

- Ethernet0 Offset: 0x2500 - 0x25ff
- Ethernet1 Offset: 0x2900 - 0x29ff
- Ethernet2 Offset: 0x2d00 - 0x2dff

Port 1 Offset	Port 2 Offset	Port 3 Offset	Counter Name	Function	Initial Value
0x2500	0x2900	0x2d00	Bytes Received	This counter increments once for every data octet of good packets (Unicast + Multicast + Broadcast) received by the port.	-
0x2504	0x2904	0x2d04	Bytes Sent	This counter increments once for every data octet of transmitted packets sent by the port.	-
0x2508	0x2908	0x2d08	Frames Received	This counter increments once for every good packet (Unicast + Multicast + Broadcast) received by the port.	-
0x250c	0x290c	0x2d0c	Frames Sent	This counter increments once for every transmitted packet sent by the port.	-
0x2510	0x2910	0x2d10	Total Bytes Received	This counter increments once for every data octet of all received packets. This includes data octets of BAD packets, which might be automatically rejected by the port (e.g fragments). This counter reflects all the data octets received from the line. **NOTE:** A nibble is NOT counted as a whole byte.	-
0x2514	0x2914	0x2d14	Total Frames Received	This counter increments once for every received packet. This includes BAD packets. This counter reflects all packets received from the line.	-
0x2518	0x2918	0x2d18	Broadcast Frames Received	This counter increments once for every good broadcast packet received.	-
0x251c	0x291c	0x2d1c	Multicast Frames Received	This counter increments once for every good Multicast packet received. This counter does not count Broadcast packets.	-

Table 577: Ethernet MIB Counters (Continued)

- Ethernet0 Offset: 0x2500 - 0x25ff
- Ethernet1 Offset: 0x2900 - 0x29ff
- Ethernet2 Offset: 0x2d00 - 0x2dff

Port 1 Offset	Port 2 Offset	Port 3 Offset	Counter Name	Function	Initial Value
0x2520	0x2920	0x2d20	CRC Error	This counter increments once for every received packet which meets all the following conditions (i.e. logical AND of the following conditions): • Packet data length is between 64 and MaxFrameSize bytes inclusive (i.e. valid packet data length per IEEE std). • Packet has invalid CRC. • Collision Event has not been detected. • Late Collision Event has not been detected. • Rx Error Event has not been detected.	-
0x2524	0x2924	0x2d24	Oversize Frames	This counter increments once for every received packet which meets all the following conditions (i.e. logical AND of the following conditions): • Packet data length is greater than MaxFrameSize. • Packet has valid CRC. • Rx Error Event has not been detected.	-
0x2528	0x2928	0x2d28	Fragments	This counter increments once for every received packet which meets all the following conditions (i.e. logical AND of the following conditions): • Packet data length is less than 64 bytes -OR- packet without SFD and is less than 64 bytes in length. • Collision Event has not been detected. • Late Collision Event has not been detected. • Rx Error Event has not been detected. • Packet has INVALID CRC.	-

Table 577: Ethernet MIB Counters (Continued)

- Ethernet0 Offset: 0x2500 - 0x25ff
- Ethernet1 Offset: 0x2900 - 0x29ff
- Ethernet2 Offset: 0x2d00 - 0x2dff

Port 1 Offset	Port 2 Offset	Port 3 Offset	Counter Name	Function	Initial Value
0x252c	0x292c	0x2d2c	Jabber	This counter increments once for every received packet which meets all the following conditions (i.e. logical AND of the following conditions): • Packet data length is greater than MaxFrameSize. • Packet has invalid CRC. • Rx Error Event has not been detected.	-
0x2530	0x2930	0x2d30	Collision	This counter increments once for every received packet which meets both of the following conditions (i.e. logical AND of the following conditions): • Collision Event has been detected. • Rx Error Event has not been detected.	-
0x2534	0x2934	0x2d34	Late Collision	This counter increments once for every received packet which meets both of the following conditions (i.e. logical AND of the following conditions): • Late Collision Event has been detected. • Rx Error Event has not been detected.	-
0x2538	0x2938	0x2d38	Frames 64 Bytes	This counter increments once for every received and transmitted packet with size of 64 bytes. This counter does not count BAD received packets.	-
0x253c	0x293c	0x2d3c	Frames 65-127 Bytes	This counter increments once for every received and transmitted packet with size of 65 to 127 bytes. This counter does not count BAD received packets.	-
0x2540	0x2940	0x2d40	Frames 128-255 Bytes	This counter increments once for every received and transmitted packet with size of 128 to 255 bytes. This counter does not count BAD received packets.	-

Table 577: Ethernet MIB Counters (Continued)

- Ethernet0 Offset: 0x2500 - 0x25ff
- Ethernet1 Offset: 0x2900 - 0x29ff
- Ethernet2 Offset: 0x2d00 - 0x2dff

Port 1 Offset	Port 2 Offset	Port 3 Offset	Counter Name	Function	Initial Value
0x2544	0x2944	0x2d44	Frames 256-511 Bytes	This counter increments once for every received and transmitted packet with size of 256-511 bytes. This counter does not count BAD received packets.	-
0x2548	0x2948	0x2d48	Frames 512-1023 Bytes	This counter increments once for every received and transmitted packet with size of 512-1023 bytes. This counter does not count BAD received packets.	-
0x254c	0x294c	0x2d4c	Frames 1024-Max-FrameSize Bytes	This counter increments once for every received and transmitted packet with size of 1024 to MaxFrameSize bytes. This counter does not count BAD received packets.	-
0x2550	0x2950	0x2d50	Rx Error	This counter increments once for every received packet in which the Rx Error Event has been detected. When a Rx Error event occurs, the following counters do not increment: CRC Error, Oversize Frames, Fragments, Jabbers, Collision and Late Collision.	-
0x2554	0x2954	0x2d54	Dropped Frames	Reserved.	-
0x2558	0x2958	0x2d58	Out Multicast Frames	The number of Multicast frames sent by the port. This counter does not count Broadcast packets.	-
0x255c	0x295c	0x2d5c	Out Broad-cast Frames	The number of Broadcast frames sent by the port.	-
NOTE: No dedicated counters.			Out Unicast Frames	To calculate Out Unicast Frames use one of the following counters: • Frames Sent • Out Multicast Frames • Out Broadcast Frames	

Table 577: Ethernet MIB Counters (Continued)

- Ethernet0 Offset: 0x2500 - 0x25ff
- Ethernet1 Offset: 0x2900 - 0x29ff
- Ethernet2 Offset: 0x2d00 - 0x2dff

Port 1 Offset	Port 2 Offset	Port 3 Offset	Counter Name	Function	Initial Value
0x2560	0x2960	0x2d60	Undersize Frames	This counter increments once for every received packet which meets all the following conditions (i.e. logical AND of the following conditions): • Packet data length is less than 64 bytes. • Collision Event has not been detected. • Late Collision Event has not been detected. • Rx Error Event has not been detected. • Packet has valid CRC.	-

14. Multi Protocol Serial Controller (MPSC)

The GT–64240A includes two MPSCs that support:

- Bit oriented protocols (e.g. HDLC)
- Byte oriented protocols (e.g. BISYNC)
- Transparent protocols
- The UART (Start/Stop) mode.

The two MPSCs:

- Can be programmed to operate simultaneously to a guaranteed bit rate of 55Mbps.
- Have dedicated DMAs to transfer data between the serial port and memory, see Section 15. “MPSC Serial DMAs (SDMA)” on page 457.
- Can be routed out via serial interface ports which implement interfaces like EIA-232 and V.34.

14.1 Signals Routing

The two MPSCs can be physically routed to S0 and S1 ports or left unconnected. The physical routing of the MPSC signals are defined in the Main Routing Register (MRR), see Table 578 .

Table 578: MPSC Routing Register (MRR), Offset: 0Xb400

Bits	Field Name	Description	Initial Value
2:0	MR0	MPSC0 Routing 0x0 - Serial Port0 0x1 - 0x6 -Reserved 0x7 - Unconnected	0x7
5:3	Reserved	Must be 0x7.	0x7
8:6	MR1	MPSC1 Routing 0x0 - Serial Port1 0x1 - 0x6 - Reserved 0x7 - Unconnected	0x7
30:9	Reserved	Reserved.	0x3fff
31	Dont_Stop_Clock		0x0

The MPSCs’ receive and transmit clocks use the baud rate generators or serial clock input signals. The routing of these signals is defined in the Rx Clock Routing Register (RCRR) and the Tx Clock Routing Register (TCRR).

NOTE: If using BRG as the Rx clock source, the SCLK pin becomes an output and drives the Rx clock. If using BRG or the SCLK pin as the Tx clock source, the TSCLK pin becomes an output and drives the Tx clock.

Table 579: Rx Clock Routing Register (RCRR), Offset 0xb404

Bits	Field Name	Description	Initial Value
3:0	CRR0	MPSC0 RX Clock Routing 0x0 - BRG0 0x1 - BRG1 0x2 - 0x7 - Reserved 0x8 - SCLK0 0x9 - 0xf - Reserved	0x0
7:4	Reserved	Must be 0.	0x0
11:8	CRR1	MPSC1 RX Clock Routing 0x0 - BRG0 0x1 - BRG1 0x2 - 0x7 - Reserved 0x8 - SCLK1 0x9 - 0xf - Reserved	0x0
31:12	Reserved	Reserved.	0x0

Table 580: Tx Clock Routing Register (TCRR), Offset 0xb408

Bits	Field Name	Description	Initial Value
3:0	CRT0	MPSC0 TX Clock Routing 0x0 - BRG0 0x1 - BRG1 0x2 - 0x7 - Reserved 0x8 - SCLK0 0x9 - TSCLK0 0xa - 0xf - Reserved	0x0
7:4	Reserved	Must be 0. 0x - 0x7 - Reserved	0x0

Table 580: Tx Clock Routing Register (TCRR), Offset 0xb408 (Continued)

Bits	Field Name	Description	Initial Value
11:8	CRT1	MPSC1 TX Clock Routing 0x0 - BRG0 0x1 - BRG1 0x2 - 0x7 - Reserved 0x8 - SCLK1 0x9 - TSCLK1 0xa - 0xf - Reserved	0x0
31:12	Reserved	Reserved.	0x0

14.2 Digital Phase Lock Loop

Each MPSC has a dedicated transmit and receive digital phase lock loop (DPLL).

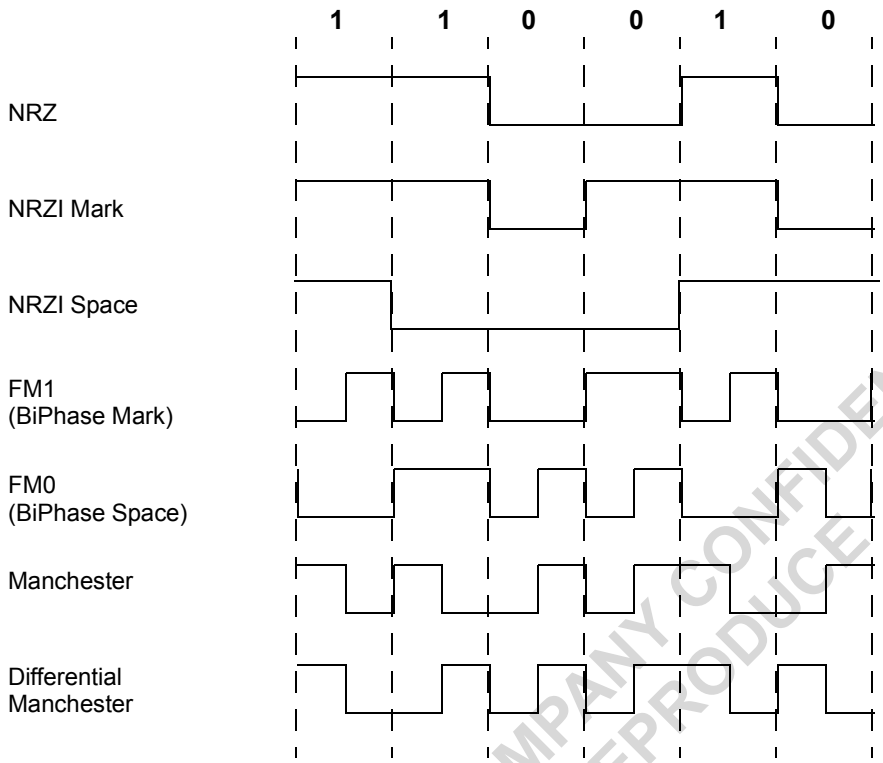
The transmit DPLL encodes the transmit bit stream to the selected code and monitors the transmit clock for glitches. If a clock glitch is detected and the Glitch Detect Enable (GDE) bit in the Main Configuration register (MMCR) is set to ‘1’, a maskable interrupt is generated.

The receive DPLL decodes the incoming bit stream according to the selected mode. If a code violation is detected (for example, no transition in Manchester code) the DE (Decoding Error) in the receive descriptor is set. The receive DPLL also performs clock recovery from the incoming bit stream and monitors the receive clock for glitches. If a clock glitch is detected and the Glitch Detect Enable (GDE) bit in the Main Configuration register (MMCR) is set to ‘1’, a maskable interrupt is generated.

14.2.1 Data Encoding/Decoding

Figure 56 shows the data encoding and decoding schemes The GT–64240A DPLL supports.

Figure 56: MPSC DPLL Encoding/Decoding Schemes



14.2.2 DPLL Clock Source

Each received DPLL uses the MPSC receive clock input and each transmit DPLL uses the MPSC transmit clock input as its source clock.

NOTE: The GT–64240A DPLLs can accept a clock source of up to 83MHz. This allows the GT–64240A to have a bit rate of up to 5MHz using a 16X clock rate scheme.

14.2.3 Receive DPLL Clock Recovery

When a MPSC is programmed to work in UART Asynchronous mode, the DPLL encoding must be set to NRZ and the clock sampling rate to x8, x16, or x32 of the bit rate. The receive DPLL recognizes a start bit and synchronizes the clock to it.

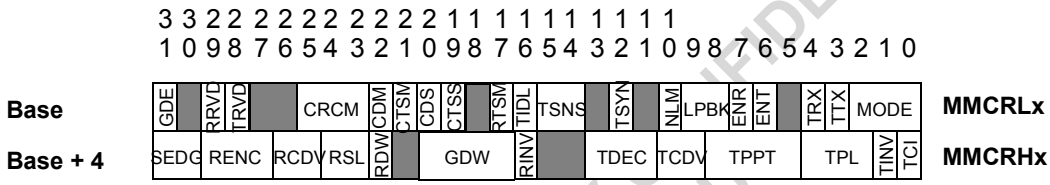
When not synchronized, the DPLL hunts for a start bit or edge. In UART mode, the DPLL hunts for start bit. In HDLC BISYNC and Transparent mode, the DPLL hunts for an edge. If hunting for a start bit (UART), the DPLL hunts for a falling edge, assuming it to be the beginning of a start bit. It then samples RxD at the middle of the bit, calculated from the falling edge of the start bit (8 ticks in x16 mode), to see that it is still ‘0’. If not, it is considered noise. A modulo 16 counter (for a 16x over-sampling rate) generates the receive clock RCLK.

In HDLC, BISYNC, and Transparent modes, the DPLL tries to lock itself on the transitions of the receive bit stream. When synchronization is achieved, the DPLL continuously monitors for rising and falling edges as defined in the MPSC Main Configuration Register (MMCR). When detecting an edge, the edge-compare logic gives the counter shift_left or shift_right commands to maintain lock on the received data.

14.3 MPSCx Main Configuration Register (MMCRx)

Each MPSC has an MPSC Main Configuration Register (MMCRx). The MMCRx is a 64 bit register used to configure common MPSC features. It is protocol independent. The MMCRx consists of two 32 bits registers, MMCRHx and MMCRLx, as shown below.

Figure 57: MPSC Main Configuration Register (MMCRx)



Unless otherwise specified:

- '1' means set
- '0' means not set

14.3.1 MPSCx Main Configuration Register Low (MMCRLx)

Table 581: MPSCx Main Configuration Register Low (MMCRLx)

- MPSC0 Offset: 0x8000
- MPSC1 Offset: 0x9000

Bits	Field Name	Function	Initial Value
2:0	MODE	Mode 000 -HDLC (default) 001 -Reserved 010 -Reserved 011 -Reserved 100 -UART 101 -BISYNC 110 -Reserved 111 -Reserved	0x0
3	TTX	Transparent Transmitter 0 - Normal Mode. (default) 1 - Transparent Mode. (Transparent Mode overrides the program mode in MODE bits.)	0x0

Table 581: MPSCx Main Configuration Register Low (MMCRLx) (Continued)

- MPSC0 Offset: 0x8000
- MPSC1 Offset: 0x9000

Bits	Field Name	Function	Initial Value
4	TRX	Transparent Receiver 0 - Normal Mode (default) 1 - Transparent Mode. (Transparent Mode overrides the program mode in MODE bits.)	0x0
5		Reserved.	0x0
6	ET	Enable Transmit 0 - Disabled The Tx channel is in Low Power Mode. 1 - Enable The Tx controller is ready for data. When the SDMA has data to transmit, it loads the data to the Tx controller that transmits the data in the selected protocol.	0x0
7	ER	Enable Receive 0 - Disabled The Rx channel is in Low Power Mode. 1 - Enable. The Rx controller is ready to receive data.	0x0
9:8	LPBK	Loop Back (for diagnostic) mode 00 -Normal Operation, no loopback (Default) 01 -Loopback 10 -Echo 11 -Loop Back + Echo In loopback mode, which is only for diagnostic purposes, the transmitted data on TxD is also fed into RxD. In this mode, the same clock source should be used for both Rx and TX. Echo mode re-transmits received data on RxD (with one clock delay) on TxD. If CD* is asserted, the receiver also receives the incoming data.	0x00

Table 581: MPSCx Main Configuration Register Low (MMCRLx) (Continued)

- MPSC0 Offset: 0x8000
- MPSC1 Offset: 0x9000

Bits	Field Name	Function	Initial Value
10	NLM	Null Modem 0 - Normal operation The MPSC uses the CD* and CTS* inputs to control the data flow 1 - Null Modem The MPSC CD* and RTS* internal signals are always asserted. The external pin status can still be read from the Event Register. **NOTE:** For information about the behavior of the Event Register in different modes, see: • Section "The ESR register holds information on the transmit/receive channel condition." on page 419. • Section 14.7.5.6 "CHR10 - BISYNC Event Status Register (ESR)" on page 433. • Section 14.8.5.7 "CHR10 - UART Event Status Register (ESR)" on page 445. • Section 14.9.2.3 "CHR10 - Transparent Event Status Register (ESR)" on page 453.	0x0
11		Reserved.	0x0
12	TSYN	Transmitter Synchronize to Receiver Setting this bit synchronizes the transmitter to receiver byte boundaries. This is particularly important in the X.21 protocol. 0 - No synchronization assumed. 1 - Transmit bit stream is synchronized to the receive bit stream. **This bit affects only a transparent transmitter.** The transmitter starts transmission nx8 bit period after the receive data arrives. If CTS* is already asserted, the transparent transmitter starts transmitting eight clocks after the receiver starts to receive data. **NOTE:** Only this bit when transmit and receive clocks are equal and TCDV and RCDV are set to '00'.	0x0
13		Reserved.	0x0

Table 581: MPSCx Main Configuration Register Low (MMCRLx) (Continued)

- MPSC0 Offset: 0x8000
- MPSC1 Offset: 0x9000

Bits	Field Name	Function	Initial Value
15:14	TSNS	Transmit Sense. Defines the number of bit times the internal sense signal will stay active after last transition on the RXD line occurs. It is useful for AppleTalk protocol to avoid the spurious CD* change interrupt that would otherwise occur during the frame synchronization sequence that precedes the opening flag. The delay is a function of RCDV (clock divider) setting. 00 (RCDV = 0) - Infinite (Carrier Sense is always active - default) 00 (RCDV ≠ 0) - Infinite (Carrier Sense is always active - default) 01 (RCDV = 0) - 14 bit times 01 (RCDV ≠ 0) - 6.5 bit times 10 (RCDV = 0) - 4 bit times (normal AppleTalk) 10 (RCDV ≠0) - 2.5 bit times (normal AppleTalk) 11 (RCDV = 0) - 3 bit times 11 (RCDV ≠ 0) - 1 bit time	0x0
16	TIDL	Transmit Idles 0 - TxD is encoded during data transmission (including preamble and flags/sync patterns). (Default.) TxD is in MARK during idle. 1 - TxD is encoded all the time, even when idles are transmitted, see Table 582 .	0x0
17	RTSM	RTS* Mode This bit may be changed on the fly. 0 - Send IDLE between frames. RTS* negated between frames and the IDLE pattern is defined by the protocol and TIDL bit. 1- Send flags/syncs between frames according to the protocol. RTS* is always asserted. Refer to Table 582 .	0x0
18	Reserved	Reserved.	0x0

Table 581: MPSCx Main Configuration Register Low (MMCRLx) (Continued)

- MPSC0 Offset: 0x8000
- MPSC1 Offset: 0x9000

Bits	Field Name	Function	Initial Value
19	CTSS	CTS* Sampling Mode 0 - Asynchronous CTS* (Default) CTS* is synchronized inside the GT–64240A. Transmission starts after synchronization is achieved with a few cycles delay to the external CTS*. 1 - Synchronous CTS* CTS* is synchronized to the Rx clock. **NOTE:** Synchronous CTS* must be used for ISDN D channels.	0x0
20	CDS	CD* Sampling mode 0 - Asynchronous CD* (Default) CD* is synchronized internally in the GT–64240A and then data is received. 1 - Synchronous CD*. CD* is synchronized to the Rx clock.	0x0
21	CTSM	CTS* Operating Mode 0 - Normal mode (Envelop Mode) CTS* must envelop the frame. Deassertion of CTS* during transmission causes a CTS lost error. 1- Pulse Mode Once CTS* is sampled low, synchronization has been achieved. Further transitions of CTS* have no effect. CTS* synchronization is lost when RTS* is deasserted.	0x0
22	CDM	CD* Operating Mode 0- Normal mode (Envelop Mode) CD* must envelop the frame. Deassertion of CD* during reception causes a CD lost error. 1- Pulse Mode Once CD* is sampled low, synchronization has been achieved. Further transitions of CD* have no effect.	0x0
25:23	CRCM	CRC Mode 000 - CRC16-CCITT (HDLC based protocols, e.g. X.25) (Default) 001 - CRC-16 (BISYNC) 010 - CRC32-CCITT (HDLC based protocols, e.g. LAP-D. Identical to the Ethernet CRC) 011 - Reserved 1XX- Reserved	0x010
27:26	Reserved	Reserved.	0x0

Table 581: MPSCx Main Configuration Register Low (MMCRLx) (Continued)

- MPSC0 Offset: 0x8000
- MPSC1 Offset: 0x9000

Bits	Field Name	Function	Initial Value
28	TRVD	Transmit Reverse Data 0 - Normal Mode. (Default) 1 - Reverse Data Mode. MSB is shifted out first.	0x0
29	RRVD	Receive Reverse Data 0 - Normal Mode. (Default) 1 - Reverse Data Mode. MSB is shifted in first.	0x0
30	Reserved	Reserved.	0x0
31	GDE	Glitch Detect Enable 0 - Normal mode. No glitch detect. (Default) 1 - When glitch is detect, a maskable interrupt is generated. When this bit is set, the MPSC looks for glitches in the external receive and transmit clocks. **NOTE:** The GT–64240A tries to clean the input clocks by receiving them via a Schmitt trigger input buffer.	0x0

•

The following table summarizes the relationship between the TIDL and RTSM

Table 582: TIDL/RTSM Relationship

RTSM/TIDL	TxD	RTS*	TxD	RTS*
00	'1' Not Encoded	1	Data Encoded	0
01	'1' Encoded	1	Data Encoded	0
10	Flags/Not Encoded	0	Data Encoded	0
11	Flags/Encoded	0	Data Encoded	0

14.3.2 MPSCx Main Configuration Register High (MMCRHx)

Table 583: MPSCx Main Configuration Register High (MMCRHx)

- MPSC0 Offset: 0x8004
- MPSC1 Offset: 0x9004

Bits	Field Name	Function	Initial Value
0	TCI	Transmit Clock Invert 0 - Normal operation (Default.) Data is shifted out on the falling edge. 1 -The internal transmit clock is inverted by the MPSC before it is used. This allows the MPSC to clock data out half a cycle earlier on the rising edge of the clock.	0x0
1	TINV	Transmit Bit Stream Inversion 0 - No invert 1 - Invert the data before it is sent to the DPLL Setting TINV to '1' generates FM1 from FM0, NRZI mark from NRZI space, etc. It also inverts the bit stream in NRZ mode.	0x0
4:2	TPL	Transmit Preamble Length Determines the number of preamble bytes the transmitter sends before it starts to transmit data. The send pattern is defined by the TPPT bits. 000 - No Preamble (Default) 001 - 1 byte 010 - 2 bytes 011 - 4 bytes 100 - 6 bytes 101 - 8 bytes 110 - 16 bytes 111 - Reserved	0x0
8:5	TPPT	Transmit Preamble Pattern Defines a character sent as a preamble sequence. Two TPPT characters form a preamble byte. The number of preamble bytes sent is defined by the TPL field. The receiving DPLL uses the preamble pattern to lock on the receiving signal.	0x0

Table 583: MPSCx Main Configuration Register High (MMCRHx) (Continued)

- MPSC0 Offset: 0x8004
- MPSC1 Offset: 0x9004

Bits	Field Name	Function	Initial Value
10:9	TCDV	Transmit Clock Divider Defines the transmit clock divider. The transmit bit rate is the rate of the clock entering the MPSC Tx machine (from external pin or a BRG) divided by the TCDV field. For FM0, FM1, Manchester, and Differential Manchester, one of the 8x, 16x, or 32x options must be set. 00 - 1x clock mode (Default. For NRZ and NRZI only.) 01 - 8x clock mode 10 - 16x clock mode 11 - 32x clock mode	0x0
13:11	TDEC	Transmit Encoder Specifies the encoding method for the dedicated Tx channel DPLL. 000 - NRZ (default) 001 - NRZI (mark, can be set to Space by setting TINV bit) 010 - FM0 (can be set to FM1 by setting the TINV bit) 011 - Reserved 100 - Manchester 101 - Reserved 110 - Differential Manchester 111 - Reserved	0x0
15:14	Reserved	Reserved.	0x0
16	RINV	Receive Bit Stream Inversion. 0 - No invert 1 - Inverts the data before it is sent from the DPLL to the MPSC data path. Setting RINV to '1' decodes FM1 and NRZI mark when the RENC field is programed to FM0 and NRZI space etc. It also inverts the received bit stream in NRZ mode.	0x0
20:17	GDW	Clock Glitch Width When the GDE bit is set, the MPSC considers Tx/Rx clock pulses that are narrower than GDW system clocks as a glitch.	0x0
21	Reserved	Reserved.	0x0

Table 583: MPSCx Main Configuration Register High (MMCRHx) (Continued)

- MPSC0 Offset: 0x8004
- MPSC1 Offset: 0x9004

Bits	Field Name	Function	Initial Value
22	RDW	Receive Data Width 0 - Normal mode The MPSC data path is 16-bits wide. Upon receiving 16-bits, the data is transferred into the SDMA FIFOs. The buffers must be 64-bit word aligned and DMA bursts enabled. **NOTE:** Normal Mode must be used for HDLC based protocols. 1 - Low latency operation Data is transferred to the FIFOs after 8-bits are received. Logical FIFO width is one byte. **NOTE:** This mode allows byte aligned buffers and must be chosen for BISYNC and UART modes. DMA bursts are disabled. The SDMA writes one byte per DRAM access. Setting RDW also bypasses the receive FIFO threshold. The SDMA arbitrates for DMA access as soon as the FIFO has one byte in it.	0x0
24:23	RSYL	Receive Sync Length (BISYNC and Transparent Modes) 00 - External sync (CD* assertion) 01 - 4-bit sync 10 - 8-bit sync (MonoSYNC) 11 - 16-bit sync (BISYNC)	0x0
26:25	RCDV	Receive Clock Divider Defines the receive clock divider. The receive bit rate is the rate of the clock entering the MPSC Rx machine (from external pin or a BRG) divided by the RCDV field. For FM0, FM1, Manchester, and Differential Manchester, one of the 8x, 16x, or 32x options must be set. 00 - 1x clock mode (Default. For NRZ and NRZI only.) 01 - 8x clock mode 10 - 16x clock mode 11 - 32x clock mode	0x0

Table 583: MPSCx Main Configuration Register High (MMCRHx) (Continued)

- MPSC0 Offset: 0x8004
- MPSC1 Offset: 0x9004

Bits	Field Name	Function	Initial Value
29:27	RENC	Receive Encoder Specifies the encoding method for the dedicated Rx channel DPLL. 000 - NRZ (default) 001 - NRZI (Mark, can be set to Space by setting RINV bit) 010 - FM0 (can be set to FM1 by setting the RINV bit) 011 - Reserved 100 - Manchester 101 - Reserved 110 - Differential Manchester 111 - Reserved	0x0
31:30	SEDG	Synchronization Clock Edge The clock edge used by the DPLL for adjusting the receive sample point due to drift in the receive signal. 00 - Both rising and falling edges. (Default.) 01 - Rising edge 10 - Falling edge 11 - No adjustment	0x0

14.4 MPSCx Protocol Configuration Registers (MPCRx)

Each MPSC has a dedicated Protocol Configuration Register (MPCRx).

The MPCRx registers are located at base+08 relative to the corresponding MPSC Main Configuration Register (MMCRx). The functionality of the MPCRx is protocol dependent. Detailed descriptions of the MPCRs are given in the following protocol sections.

14.5 Channel Registers (CHxRx)

Each MPSC and the ethernet controller has ten dedicated Channel Registers (CHxRx) to program the MPSC or ethernet controller.

The CHxRx registers are located at base+0xC0 through base+0x30 relative to the corresponding MPSC Main Configuration Register (MMCRx). The functionality of the CHxRx is protocol dependent. Detailed descriptions of the CHRs are given in the following protocol sections.

14.6 HDLC Mode

14.6.1 HDLC Receive/Transmit Operation

In HDLC mode, an MPSC performs the following protocol functions:

- Flag generation and stripping
- Bit stuffing and stripping
- Address recognition (up to 16 bit addresses)
- CRC generation and checking
- Line condition monitoring
- LocalTalk preamble generation
- LocalTalk trailing abort generation

Figure 58: Typical HDLC Frame

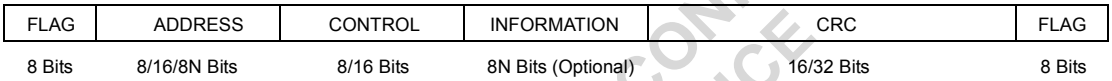
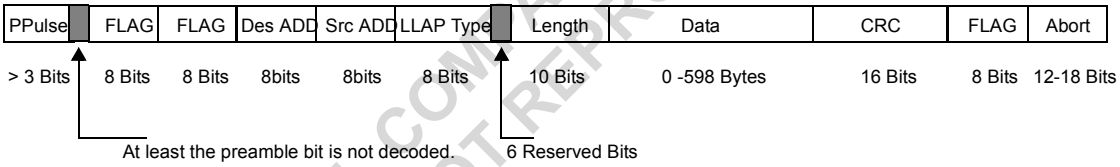


Figure 59: Typical LocalTalk Frame



14.6.2 SDMAx Command/Status Field for HDLC Mode

When an MPSC is in HDLC mode, the Command/Status field in the corresponding SDMAx descriptor has the following format:

Table 584: SDMAx Command/Status Field for HDLC Mode

Bit	Rx - Function	Tx - Function
31	O -Owner	O - Owner
30	AM - Auto Mode	AM - Auto Mode
29:24	Reserved	Reserved
23	EI - Enable Interrupt	EI - Enable Interrupt
22	Reserved	GC - Generate CRC
21:18	Reserved	Reserved
17	F - First	F - First

Table 584: SDMAx Command/Status Field for HDLC Mode (Continued)

Bit	Rx - Function	Tx - Function
16	L - last	L - Last
15	ES - Error Summary ES = CE || CDL || DE || NO || ABR || OR || MFL || SF[1]	Error Summary ES = CTSL || UR || RL[1]
14	Reserved	Reserved
13:10	Reserved	RC-Retransmit Count (LAN HDLC mode only)
9	Reserved	COL - Collision Occurred
8	SF - Short Frame	RL - Retransmit Limit Error
7	MFL - Max Frame Length Err	Reserved
6	OR - Data Overrun/Residue[2]	UR - Data Underrun
5	Residue[1]	Reserved
4	ABR - Abort Sequence/Residue[0]	Reserved
3	NO - Non Octet Frame	D-deferred. Transmission was deferred due to busy channel.
2	DE - Decoding Error	Reserved
1	CDL - CD Loss	CTSL - CTS Loss
0	CE - CRC Error	Reserved

1. "||" means logical OR.

14.6.3 MPSCx Protocol Configuration Register (MPCRx) for HDLC

Figure 60: MPSCx Protocol Configuration Register (MPCRx) for HDLC

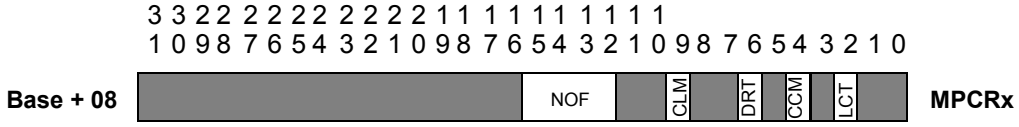


Table 585: MPSCx Protocol Configuration Register (MPCRx) for HDLC

- MPSC0 Offset: 0x8008
- MPSC1 Offset: 0x9008

Bits	Field Name	Function	Initial Value
1:0	Reserved	Reserved.	0x0
2	LCT	Local Talk When set, the following LocalTalk support is added to the HDLC controller: • Two abort sequences will be generated at the end of frame following its closing flag. • A preamble will be generated. No encoding will be done for the last preamble bit When working with LocalTalk, the FM0 Encoding Scheme should be set by writing '010' to RENC and TDEC in the MMCRx. The user should also set TPPT to 0xF and TPL to '1' (one byte preamble). The last preamble bit is not decoded. This must be done for LocalTalk RTS frames. Setting TPL to '0' leads to a frame without preamble. This can be used with LocalTalk data frames. Setting TPL to other values leads to unpredictable results.	0x0
3	Reserved	Reserved.	0x0

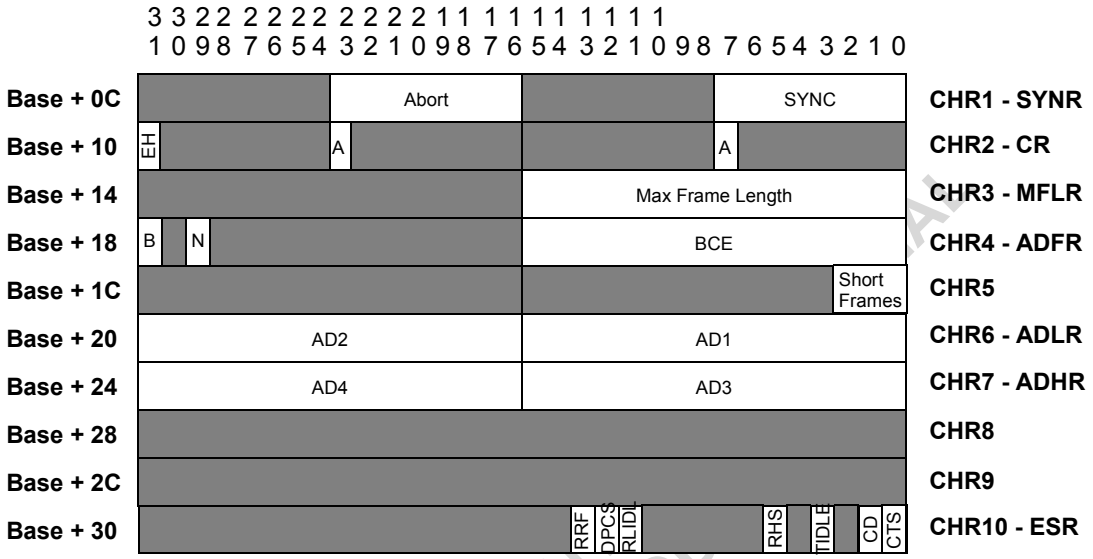
Table 585: MPSCx Protocol Configuration Register (MPCRx) for HDLC (Continued)

- MPSC0 Offset: 0x8008
- MPSC1 Offset: 0x9008

Bits	Field Name	Function	Initial Value
4	CCM	CRC Compliance Mode. In HDLC, the TX side uses bit stuffing to prevent a data/CRC pattern from looking like an HDLC control flag. The CCM tells the Rx side how to handle frames that were received with mistakes in bit stuffing, when they occur immediately before the end flag. This is a borderline condition that may or may not present a problem in actual systems. 0 - Compatible Mode If the Rx side receives a frame that is missing a stuffed bit that is supposed to be immediately before the End Flag, then mark in the descriptor that the frame has a good CRC, and pass the good CRC along to the buffer. 1 - Compliance Mode If the Rx side receives a frame that is missing a stuffed bit that is supposed to immediately proceed the End Flag, then mark in the descriptor that the frame has a bad CRC, and pass the errored CRC to the buffer.	
5	Reserved	Reserved.	0x0
6	DRT	Disable Rx on Tx When DRT is set to '1' the Rx path is closed during Tx. This is useful in multidrop configurations when a user doesn't want to receive its own frames.	0x0
8:7	Reserved	Reserved.	0x0
9	CLM	Collision Mode When set to '1', the MPSC transceiver tries to retransmit a frame after a CTS lost. This mode allows automatic collision resolution for an ISDN LAP-D type channel.	0x0
11:10	Reserved	Reserved.	0x0
15:12	NOF	Number of Flags Specifies the number of flags transmitted between consecutive frames. Setting NOF to '0' specifies shared flag mode. In shared flag mode, the closing flag of a frame is used as the opening flag of the following frame. This setting also puts the receiver in back-to-back mode. The default value is '1'.	0x1
31:16	Reserved	Reserved.	0x0

14.6.4 Channel Registers (CHxRx) for HDLC Mode

Figure 61: Channel Registers (CHxRx) for HDLC



Unless otherwise is specified:

- '1' means set.
- '0' means not set.
- '0' is the default value after reset.

Table 586: CHR1 - Sync/Abort Register (SYNR)

- MPSC0 Offset: 0x800c
- MPSC1 Offset: 0x900c

Bits	Field Name	Function	Initial Value
7:0	SYNC	Holds the synchronization pattern for the receive machine and opening/closing flag/sync-pattern for the transmit machine. This is an HDLC flag so no additional programing is needed for the HDLC protocol. **NOTE:** After reset it holds the value of 7E in the SYNC field	FE
15:8	Reserved	Reserved.	0x0
23:16	Abort_Pattern	The abort pattern is transmitted upon receiving an abort command.	0x0
31:24	Reserved	Reserved.	0x0

Table 587: CHR2 - Command Register (CR)

- MPSC0 Offset: 0x8010
- MPSC1 Offset: 0x9010

Bits	Field Name	Function	Initial Value
N/A	TD	Transmit Demand Fetch a descriptor and start transmission. Issued through the SDMAx Command Register.	
N/A	Stop	Stop Complete frame transmission and stop. (Go to IDLE). Issued through the SDMAx Command Register.	
6:0	Reserved	Reserved.	0x0
7	A	Abort Transmission Abort transmission immediately and go to IDLE. The descriptor is not closed or incremented. **NOTE:** Command is not synchronized to byte.	0x0
22:8	Reserved	Reserved.	0x0
23	A	Abort Reception Abort receive immediately and go to IDLE. The descriptor is not closed or incremented. The processor must issue enter hunt command after abort command in order to enable reception. The bit is cleared upon entering IDLE state. After executing an Abort Reception, the CPU must disable the Tx SDMA channel. The CPU then needs to execute a normal initialization process to the MPSC.	0x0
30:24	Reserved	Reserved.	0x0
31	EH	Enter Hunt Upon receiving the Enter Hunt command, the receive machine moves to HUNT state and continuously searches for an opening flag. If enter hunt mode command is issued during frame reception, the current descriptor is closed with CRC error[1]. The EH bit is cleared upon entering Hunt state.	0x0

1. The reception process for this purpose begins after proper address recognition is allowed. Before achieving an address match, the receiver goes to Enter Hunt state without closing the descriptor.

NOTES: The ET bit in the Main Configuration Register must be set to '1' before issuing the following Transmit Demand, Stop Transmission, or Abort Transmission commands.

The ER bit in the Main Configuration Register must be set to '1' before issuing the Enter Hunt or Abort Reception commands.

When the ET or ER bits are deasserted, the MPSCx transmit/receive channel is in low power mode (NO CLOCK). Issuing one of the above commands in this state will lead to unpredictable results.

Table 588: CHR3 - Maximum Frame Length Register (MFLR)

- MPSC0 Offset: 0x8014
- MPSC1 Offset: 0x9014

Bits	Field Name	Function	Initial Value
15:0	FLBR	Frame Length Buffer Register Holds the maximum allowed frame length. When a frame exceeds the number written in the FLBR, the remainder of the frame is discarded. The HDLC controller waits for a closing flag and then returns the frame status with bit 7 (MFLE) set to '1'.	0xFFFF
31:16	Reserved	Reserved.	0x0

Table 589: CHR4 - Address Filtering Register (ADFR)

- MPSC0 Offset: 0x8018
- MPSC1 Offset: 0x9018

Bits	Field Name	Function	Initial Value
15:0	BCE	Bit Comparison Enable Bits Setting '1' in one of the BCE bits enables the address comparison for this bit: • For 16-bit LAP-D like address recognition, write 0xFFFF in ADFR. • For 8-bit HDLC/LAP-B like address recognition, write 0x00FF in ADFR. • For reception of a predefined address group, write '0' to the appropriate bits to disable address comparison on these bits.	0x0
28:16	Reserved	Reserved.	0x0
29	N	Null Enable Enables the reception of HDLC NULL address (0x0000 or 0x00 depending on the BCE setting)	0x0

Table 589: CHR4 - Address Filtering Register (ADFR) (Continued)

- MPSC0 Offset: 0x8018
- MPSC1 Offset: 0x9018

Bits	Field Name	Function	Initial Value
30	Reserved	Reserved.	0x0
31	B	Broadcast Enable Enables the reception of HDLC broadcast address (0xFFFF or 0xFF, depending on the BCE setting).	0x0

Table 590: CHR5 - Short Frame Register (SHFR)

- MPSC0 Offset: 0x801c
- MPSC1 Offset: 0x901c

Bits	Field Name	Function	Initial Value
2:0	SHFR	Short Frame Register Setting SHFR to '1' enables the Short Frame Error report. Short Frames are frames with byte count less than 3+SHFR.	0x0
31:3	Reserved	Reserved.	0x0

Table 591: CHR6 - Address 1 and 2 Register (ADLR)

- MPSC0 Offset: 0x8020
- MPSC1 Offset: 0x9020

Bits	Field Name	Function	Initial Value
15:0	AD1	Address 1 A 16-bit address that can be used for receive address recognition.	0x0
31:16	AD2	Address 2 A 16-bit address used for receive address recognition.	0x0

Table 592: CHR7 - Address 3 and 4 Register (ADHR)

- MPSC0 Offset: 0x8024
- MPSC1 Offset: 0x9024

Bits	Field Name	Function	Initial Value
15:0	AD3	Address 3 A 16-bit address that can be used for receive address recognition.	0x0
31:16	AD4	Address 4 A 16-bit address that can be used for receive address recognition.	0x0

Table 593: CHR8 - Reserved

- MPSC0 Offset: 0x8028
- MPSC1 Offset: 0x9028

Bits	Field Name	Function	Initial Value
31:0	Reserved	Reserved. **NOTE:** Do not access this register in the HDLC mode.	0x0

Table 594: CHR9 - Reserved

- MPSC0 Offset: 0x802c
- MPSC1 Offset: 0x902c

Bits	Field Name	Function	Initial Value
31:0	Reserved	Reserved. **NOTE:** Do not access this register in the HDLC mode.	0x0

The ESR register holds information on the transmit/receive channel condition.

CHR10 can be read by the CPU for channel condition resolution. Some changes in the channel condition can generate maskable interrupts, as shown below.

Table 595: CHR10 - Event Status Register (ESR)

- MPSC0 Offset: 0x8030
- MPSC1 Offset: 0x9030

Bits	Field Name	Function	Initial Value
0	CTS	Clear To Send Signal Generates an interrupt when this signal is deasserted during transmit.	0
1	CD	Carrier Detect Signal Generates and interrupt when this signal is deasserted during receive.	0
2	Reserved	Reserved.	0x0
3	TIDLE	Tx in IDLE state. Generates an interrupt upon entering IDLE state.	0x0
4	Reserved	Reserved.	0x0
5	RHS	Rx Hunt State 0 - Rx out of hunt state 1 - Rx in hunt state	0x0
10:6	Reserved	Reserved.	0x0
11	RLIDL	1 = Rx IDLE Line	0x0
12	DPCS	1 = DPLL Carrier Sense.	0x0
13	RRF	1 = Rx Receiving Flags.	0x0
31:14	Reserved	Reserved.	0x0

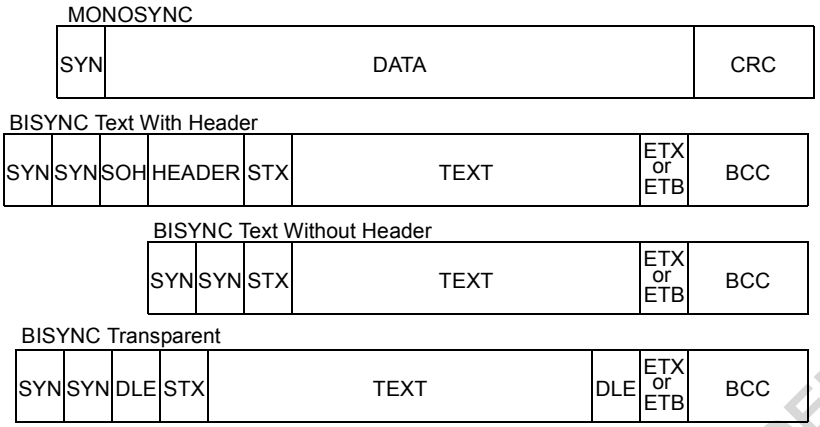
14.7 BISYNC Mode

The GT–64240A BISYNC controller was designed to reduce CPU overhead by executing most of the protocol requirements for BISYNC/MonoSYNC mode without CPU interference.

When Auto Transparent mode is enabled, the GT–64240A automatically switches to the transparent receive mode upon receiving a DLE STX sequence.

Other features are controlled by programming the bank of control registers.

Figure 62: Typical BISYNC/MonoSYNC Frames



14.7.1 BISYNC Transmit Operation

In BISYNC mode an MPSC handles the following protocol functions:

- Leading SYNC character transmission before a buffer with F bit set.
- Optional 32-bit transmission before the SYNC transmission.
- DLE transmission before a buffer with the TD bit set.
- BCC generation:
  - BCC (CRC-16, VRC/LRC and VRC/CRC-16) is calculated.
  - Buffers with BCE set to ‘0’ are excluded from BCC calculation.
  - CRC reset is controlled from the RC bit in Tx descriptor.
  - The calculation of BCC is sent or discarded according to the GC bit in the Tx descriptor.
- Automatic stuffing of DLE when transmitting a transparent buffer (buffer with TR bit set).
- SYNC transmission if underrun occurs.

BISYNC transmission is descriptor chain oriented. Transmission starts when the CPU issues a Transmit Demand command and continues until the channel’s SDMA reaches a NULL pointer or a ‘not owned’ descriptor.

14.7.2 BISYNC Receive Operation

There are two major operating modes in the BISYNC receiver.

Table 596: BISYNC Receiver Operating Modes

Mode	Function
Normal Mode	The CPU must monitor each received byte and manage each BISYNC operation (e.g., moving into transparent mode) manually.
Auto Transparent Mode	The GT–64240A handles transparent mode automatically. This mode reduces the CPU burden since it can monitor the incoming data buffer-by-buffer and not byte-by-byte.

14.7.2.1 BISYNC Normal Receive Mode

In Normal Mode, the BISYNC receiver handles the following protocol functions:

- BISYNC, MonoSYNC, NibbleSYNC or External SYNC synchronization.
- Auto SYNC stripping in text mode.
- Auto DLE-SYNC stripping in transparent text mode.
- Auto SYNC stripping after receiving DLE ITB in transparent mode.
- Automatic exit of transparent mode after receiving DLE-ETX/ETB (if RTR bit in the MPCRx was cleared).
- Marking of buffers that contain transparent data by setting the TB bit in the descriptor.
- BCC generation:
  - BCC (CRC-16, VRC/LRC and VRC/CRC-16) is calculated.
  - In transparent text mode, CRC-16 always overrides the VRC.
  - SYNC (DLE-SYNC) is not included in the BCC calculation.
- Buffer closing at the reception of ETX, ETB, ITB and ENQ.
- Maintaining SYNC (stay in text mode) after ITB.
- Protocol correctness checking:
  - Test for '1' padding at the end of block reception. (The CPU should ignore a padding error reported after ITB, and can use it when testing for proper NAK or EOT.)
  - Test for DLE-CTL after receiving DLE-ITB in transparent text mode. If another sequence arrives (except SYNCs), buffer is closed with DLE error.

14.7.2.2 BISYNC Auto Transparent Receive Mode

In Auto Transparent Mode, the BISYNC receiver handles the following protocol functions:

- BISYNC, MonoSYNC, NibbleSYNC, or External SYNC synchronization.
- Auto SYNC stripping in text mode.
- Auto DLE-SYNC stripping in transparent text mode.
- Auto SYNC stripping after receiving DLE ITB in transparent mode.
- Automatic switch to transparent mode after receiving DLE-STX.
- Automatic exit of transparent mode after receiving DLE-ETX/ETB.
- Marking of buffers that contain transparent data by setting the TB bit in the descriptor.
- BCC generation:
  - BCC (CRC-16, VRC/LRC and VRC/CRC-16) is calculated.
  - In transparent text mode, CRC-16 always overrides the VRC.
  - SYNC (DLE-SYNC) is not included in the BCC calculation.
  - Opening STX/SOH (DLE-STX) are discarded from BCC calculations.
- Buffer closing at the reception of ETX, ETB, ITB, and ENQ.
- Maintaining SYNC (stay in text mode) after ITB.
- Buffer closing after SYN-SYN-DLE-CHAR (when char is not STX).

- Protocol correctness checking:
  - Test for ‘1’ padding at the end of block reception. (The CPU should ignore a padding error reported after ITB, and can use it when testing for proper NAK or EOT.)
  - Test for DLE-CTL (CTL is a control character with B or H set) after receiving DLE-ITB in transparent text mode. If another sequence arrives (except SYNCs), buffer is closed with a DLE error.

The BISYNC receive process is block oriented. A block starts after a buffer was closed due to control character reception, overrun, protocol error, parity error, or line error (i.e. CD deassertion).

The first descriptor in a block is marked with F bit set to ‘1’. The last descriptor in block is marked with L bit set to ‘1’. The last descriptor also includes the actual status report for the block. Intermediate descriptors can be recognized by having both F and L bit set to ‘0’.

14.7.3 SDMAx Command/Status Field for BISYNC Mode

When an MPSC is in BISYNC mode the Command/Status field in the corresponding SDMAx descriptor has the following format:

Table 597: SDMAx Command/Status Field for BISYNC Mode

Bits	Rx - Function	Tx - Function
0	CE - CRC/LRC Error	Reserved
1	CDL - CD Loss	CTSL - CTS Loss
2	DE - Decoding Error	Reserved
3	DLE - DLE Error. While in transparent mode, this indicates a DLE was received and the following byte was not a valid control character.	Reserved
4	PR - Parity Error. Last byte in buffer has parity error.	Reserved
5	Reserved	Reserved
6	OR - Data Overrun	Reserved
7:8	Reserved	Reserved
9	PDR - Pad Report. This is set if there were no four consecutive ‘1’s after the block reception.	Reserved
10	Reserved	Reserved
11	TB - Transparent Buffer. Buffer contains transparent data.	Reserved
12	Reserved	Reserved
13	C - Last bytes in buffer is a user defined control character.	Reserved
14	B - Last bytes in buffer are BCC.	Reserved

Table 597: SDMAx Command/Status Field for BISYNC Mode (Continued)

Bits	Rx - Function	Tx - Function
15	ES - Error Summary ES = CDL || DE || DLE || PR || OR	ES - Error Summary ES = CTSL
16	L - Last	L - Last **NOTE:** Transmit Bit 22 is used only if L bit is set to '1'. If L bit is set to '0', no BCC is sent at the end of this buffer transmission.
17	F - First	F - First
18	Reserved	TR - Transparent mode. • 0 - Normal mode. SYNC will be sent in case of underrun • 1 - Transparent Mode. DLE-SYNC will be sent in case of underrun. CRC-16 will be used.
19	Reserved	TD - Transmit DLE before transmitting the buffer. This bit is valid only for transparent buffers. The preceding DLE is not included in the BCC calculations.
20	Reserved	BCE - BCC Enable • 0 - Buffer must be excluded from BCC calculations • 1 - Buffer must be included in BCC calculation
21	Reserved	RC - Reset BCC • 0 - BCC/LRC is accumulated. • 1 - BCC/LRC is reset.
22	Reserved	GC - Generate BCC/LRC.
23	EI - Enable Interrupt	EI - Enable Interrupt
29:24	Reserved	Reserved
30	AM - Auto Mode	AM - Auto Mode
31	O - Owner	O - Owner

14.7.4 MPSCx Protocol Configuration Register (MPCRx) for BISYNC

Figure 63: MPSCx Protocol Configuration Register (MPCRx) for BISYNC

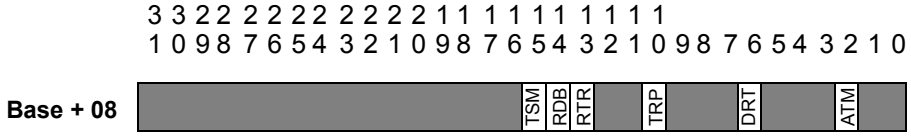


Table 598: MPSCx Protocol Configuration Register (MPCRx) for BISYNC

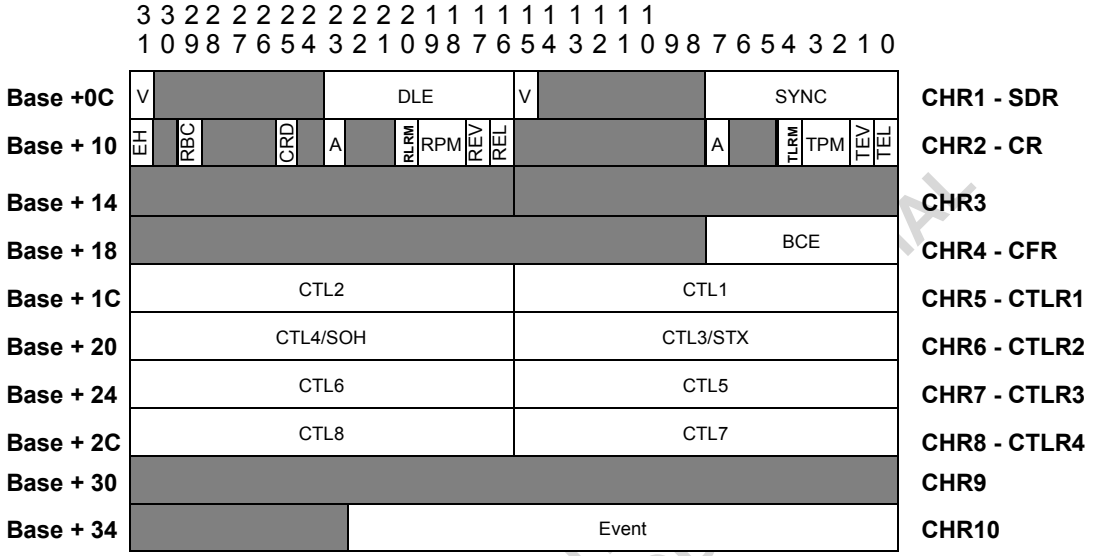
Bits	Field Name	Function	Initial Value
1:0	Reserved	Reserved.	0x0
2	ATM	Auto Transparent Mode 0 - Normal Mode. 1 - Receiver switches to transparent mode after receiving DLE-STX and exits transparent mode upon receiving a DLE-ETB or DLE-ETX sequence. When switching to transparent mode, new buffers are opened for transparent data. When the ATR bit is set to '1' the following characters should be programed into CTL1-8: • CTL3 - STX • CTL4 - SOH **NOTE:** When entering transparent mode either automatically or by issuing an RTR command, the Receiver will strip automatically leading DLEs. The TB bit in the descriptor will be set to signal the software that the buffer contains transparent data.	0x0
5:3	Reserved	Reserved.	0x0
6	DRT	Disable Rx on Tx When DRT is set to '1' the Rx path is closed during Tx. This is useful in a multidrop configuration when a user doesn't want to receive its own frames.	0x0
9:7	Reserved	Reserved.	0x0
10	TRP	Trailing Pad When set, the BISYNC transmitter sends a PAD character (0xFF) at the end of each outgoing frame (i.e. after a buffer with L bit set.)	0x0
12:11	Reserved	Reserved.	0x10

Table 598: MPSCx Protocol Configuration Register (MPCRx) for BISYNC (Continued)

Bits	Field Name	Function	Initial Value
13	RTR	Receive Transparent Mode 0 - The receiver is placed in normal mode with sync stripping and control character recognition operative. 1 - The receiver is placed in transparent mode. Syncs DLEs and control characters are recognized only after leading DLE characters. CRC16 is calculated even in VRC/LRC mode while in transparent mode. **NOTE:** When entering transparent mode either automatically or by issuing an RTR command, the receiver automatically strips leading DLEs. The TB bit in the descriptor is set to signal the software that the buffer contains transparent data.	0x0
14	RDB	Receive Discard From BCC When this bit is set, the received byte is not included in the BCC. The software must set this bit within the byte time window that starts when the character is in the Rx machine internal buffer. (The software can use the BISYNC interrupts for proper synchronization.) This bit is used in software to control BISYNC. The GT–64240A clears the RDBCC bit after discarding the required byte from BCC.	0x0
15	TSM	Tx SYNC Mode 0 - Two SYNC characters are transmitted. 1 - 32 SYNC characters are transmitted. **NOTE:** The Tx machine sends at least two bytes even in MonoSYNC or NibbleSYNC modes.	0x0
31:16	Reserved	Reserved.	0x0

14.7.5 Channel Registers (CHxRx) for BISYNC Mode

Figure 64: Channel Registers (CHxRx) for BISYNC



Unless otherwise is specified:

- '1' means set
- '0' means unset.
- '0' is the default value after reset.

14.7.5.1 CHR1 - SYNC/DLE Register (SDR)

CHR1[7:0] holds the SYNC character and CHR1[23:16] holds the DLE character for the channel. After reset it holds the value of 7E in the SYNC field and FE in the DLE field. The user must write the appropriate values before enabling the Rx/Tx machines.

If bit 15 is set, the BISYNC receive machine discards the SYNC patterns received in a middle of a message.

NOTE: This usually happens when the transmitter experiences underrun.

If bit [15] is '0' the SYNC characters is transferred to the receive buffer.

If bit 31 is '1', the first DLE received in transparent mode is discarded. If bit 31 is '0', the BISYNC receiver is not discard DLE in transparent mode.

A BISYNC transmitter always stuffs the leading DLE before transmitting the DLE that is part of a transparent buffer (transmit descriptor with TR bit set). In order to send DLE ETX, for example, the CPU must either prepare a buffer that contains DLE ETX and set TR='0', or prepare a buffer with ETX and program the transmitter to send a leading DLE by setting the TD bit in the descriptor.

A BISYNC transmitter always transmits SYNC-SYNC at the beginning of a frame. This is true in MonoSYNC and NibbleSYNC modes.

When a BISYNC transmitter experiences underrun it transmits continuous SYNC patterns in text mode or DLE-SYNC in transparent mode. The BISYNC transmitter exits this state upon receiving new data or when the CPU issues a Stop or Abort command.

The receiver SYNC length is programmable. The actual length is determined according to the value of the RSYL bits in the MMCRx. If the RSYL bits equal #00b, the synchronization is done externally and the receiver will start receiving when CD* is asserted.

In NibbleSync mode, bits [7:4] are used by the receiver for sync recognition. Bits [3:0] should return the SYNC pattern in order to assure proper SYNC transmission.

14.7.5.2 CHR2 - Command Register (CR)

Table 599: CHR2 - Command Register (CR)

Bits	Field Name	Function	Initial Value
NA	TD	Transmit Demand Fetch a descriptor and start transmission. Issued through the SDMAx Command Register.	
NA	Stop	Stop Transmission Complete frame transmission and stop. (Go to IDLE). Issued through the SDMAs Command Register.	
0	TEL	Tx Enable Longitudinal Redundancy Check 0 - LRC is disabled. 1 - LRC is enabled. (TEL default value is 0 and the CPU must write “1” to it in order to enable LRC). When set, TEL **overrides** the CRC mode that was programed in the CRCM field in the MMCRx.	0x0
1	TEV	Tx Enable Vertical Redundancy Check (Parity Bit) 0 - VRC is disabled. 1 - VRC is enabled. (TEV default value is ‘0’ and the CPU must write ‘1’ to it in order to enable VRC).	0x0
3:2	TPM	Transmit Parity Mode 00 - Odd 01 - Low (always “0”) 10 - Even 11 - High (always “1”)	0x0
4	TLRM	Transmit Longitudinal Redundancy Mode 0 - Odd 1 - Even	0x1
6:5	Reserved	Reserved.	0x0
7	A	Abort Transmission Abort transmission immediately and go to IDLE. The descriptor is not closed or incremented. **NOTE:** Command is not synchronized to byte.	0x0

Table 599: CHR2 - Command Register (CR) (Continued)

Bits	Field Name	Function	Initial Value
15:8	Reserved	Reserved.	0x0
16	REL	Rx Enable Longitudinal Redundancy Check. 0 - LRC is disabled. 1 - LRC is enabled. This is the normal mode for BISYNC. When set, REL **overrides** the CRC mode that was programed in the CRCM field in the MMCRx.	0x0
17	REV	Rx Enable Vertical Redundancy Check (parity bit). 0 - VRC (parity) is disabled. 1 - VRC is enabled. This is the normal mode for BISYNC.	0x0
19:18	RPM	Receive Parity Mode 00 - Odd 01 - Low (always '0') 10 - Even 11 - High (always '1')	0x0
20	RLRM	Receive Longitudinal Redundancy Mode 0 - Odd 1 - Even	0x1
22:21	Reserved	Reserved.	0x0
23	A	Abort Reception Abort receive immediately and go to IDLE. The descriptor is not closed or incremented. The processor must issue an enter hunt command after an abort command to enable reception. The A bit is cleared upon entering IDLE state.	0x0
24	Reserved	Reserved.	0x0
25	CRD	Close Rx Descriptor When the CPU issues a CRD command the current receive descriptor is closed and the following received data is SDMA'd into a new buffer. If there is no active receive in process, no action takes place.	0x0
28:26	Reserved	Reserved.	0x0
29	RBC	Reset BCC The CPU issues an RBC command to manually reset the CRC-LRC/VRC generator. The BCC calculation starts with the next byte. The GT–64240A clears the RBC bit after resetting BCC.	0x0

Table 599: CHR2 - Command Register (CR) (Continued)

Bits	Field Name	Function	Initial Value
30	Reserved	Reserved.	0x0
31	EH	Enter Hunt Upon receiving an enter hunt command, the receive machine moves to a hunt state and continuously searches for an opening SYNC or external SYNC. If an enter hunt mode command is issued during frame reception, the current descriptor is closed with a CRC error. The EH bit is cleared upon entering a hunt state.	0x0

The ET bit in the Main Configuration Register must be set to ‘1’ before issuing any of the following commands:

- Transmit Demand.
- Stop Transmission.
- Abort Transmission.

The ER bit in the Main Configuration Register must be set to ‘1’ before issuing any of the following commands:

- Enter Hunt
- Reset BCC
- Close Rx Descriptor
- Abort Reception.

When the ET or ER bits are deasserted, the MPSCx transmit/receive channel is in low power mode (NO CLOCK).

NOTE: Issuing one of the above commands in this state will lead to unpredictable results.

Setting TEL=’0’, TEV=’1’ and CRCM=’001’, or setting REL=’0’, REV=’1’ and CRCM=’001’, will set the BISYNC transmitter/receiver to work in VRC+CRC16 mode. The calculated parity bit is considered part of the data that the CRC-16 checks.

When a BISYNC transmitter transmits a transparent buffer, it automatically switches to the CRC that was programmed in the CRCM field in MMCRx. When a receiver enters transparent mode, it automatically switches to the CRC that was programed in CRCM field in MMCRx. In both cases, CRCM must be programed to ‘001’ in order to meet the BISYNC CRC-16 specifications.

14.7.5.3 CHR4 - Control Filtering Register (CFR)

Bits 7:0 of the CFR register are the Bit Comparison Enable bits. Setting ‘1’ in one of the BCE bits enables the control comparison for this bit

14.7.5.4 CHR5-8 - BISYNC Control Character Registers

Figure 65 shows a BISYNC control register format.

The CHAR field holds the pattern for the control character while bits 8-15 are used to control the GT–64240A behavior when the control character is recognized.

Figure 65: BISYNC Control Character Register Format

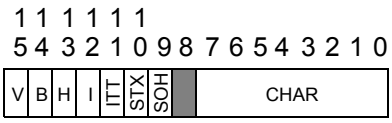


Table 600: BISYNC Control Character Register Format

Bits	Field Name	Function	Initial Value
7:0	CHAR	The Control Character To Sync On **NOTE:** Bit 7 must be programmed according to the parity method in use. See Table 599.	0x0
8		Reserved.	0x0
9	SOH	SOH Character 0 - Normal Mode 1 - SOH character. In Auto Transparent mode the characters following SOH including STX are part of the BCC calculations.	0x0
10	STX	STX Character 0 - Normal character 1 - STX character In Auto Transparent mode, an STX character is expected after the first DLE in order to enter transparent mode.	0x0
11	ITT	Ignore While Receiving in Text Mode 0 - Normal control character. 1 - Ignore this character after entering text mode (i.e. after receiving SYN-SYN-STX/SOH).	0x0
12	I	Interrupt 0 - No interrupt. 1 - Generate interrupt upon receiving this CHAR.	0x0
13	H	Hunt 0 - Close buffer and maintain SYNC. 1 - Close buffer and move to HUNT state.	0x0

Table 600: BISYNC Control Character Register Format (Continued)

Bits	Field Name	Function	Initial Value
14	B	BCC Next 0 - Close buffer. 1 - BCC is next. Receive BCC and than close buffer.	0x0
15	V	Valid. 0 - Entry is not valid. 1 - Entry is valid.	0x0

The BISYNC Control Character programming recommendations for Auto Transparent Mode and CPU Controlled Operation are shown in the following tables.

Table 601: Auto Transparent Programming

Control Character	V	B	H	I	ITT	STX	SOH
STX[1]	1	0	0	X	1	1	0
SOH[2]	1	0	0	X	1	0	1
ETX	1	1	1	X	0	0	0
ITB	1	1	0	X	0	0	0
ETB	1	1	1	X	0	0	0
ENQ	1	0	1	X	0	0	0
EOT	1	0	1	X	1	0	0
NACK	1	0	1	X	1	0	0

1. CTL3 must be use to hold STX

2. CTL4 must be use to hold SOH

Table 602: CPU Controlled Operation

Control Character	V	B	H	I	ITT	STX	SOH
ETX	1	1	1	X	0	0	0
ITB	1	1	0	X	0	0	0
ETB	1	1	1	X	0	0	0
ENQ	1	0	1	X	0	0	0
EOT	1	0	1	X	1	0	0

Table 602: CPU Controlled Operation (Continued)

Control Character	V	B	H	I	ITT	STX	SOH
NACK	1	0	1	X	1	0	0
Other Entry							
Other Entry							

14.7.5.5 CHR9 - Reserved

This register is reserved.

Do not access this register in the BISYNC mode.

14.7.5.6 CHR10 - BISYNC Event Status Register (ESR)

The ESR register holds information on the transmit/receive channel condition.

CHR10 can be read by the CPU for channel condition resolution. Some changes in the channel condition can generate maskable interrupts, as shown below.

Table 603: CHR10 - BISYNC Event Status Register (ESR)

Bits	Field Name	Event
0	CTS	Clear To Send Signal[1]
1	CD	Carrier Detect Signal[2]
2		Reserved
3	TIDLE	Tx in Idle State[3]
5	RHS	Rx in HUNT state
6-10		Reserved
11	RLIDL	1 = Rx IDLE Line[4]
12	DPCS	1 = DPLL Carrier Sense
13-15		Reserved.
16-23	RCRn	Received Control Character n When the BISYNC receiver recognizes a control character it sets the corresponding RCRn bit. Bit 16 (RCR1) corresponds to CTL1. Bit 23 (RCR8) corresponds to CTL8. RCRn bits are cleared by writing '1' to the bit. RCRn is set if the corresponding control character arrives, and both its Valid bit and Interrupt bit are also set (e.g., bit 16 will be set if CTL1 arrives, and both CTL1's "V" bit is set, and CTL1's "I" bit is also set.)

1. Interrupt is generated when signal is deasserted during transmit

2. Interrupt is generated when signal is deasserted during receive

3. Interrupt is generated upon entering IDLE state

4. Interrupt is generated upon change in line status

NOTE: PERR is set in transparent mode during SYN stripping when a non DLE or SYN character is received. This is a protocol violation. The receiver moves to hunt mode and a maskable interrupt is generated. The received character is discarded.

14.8 UART Mode

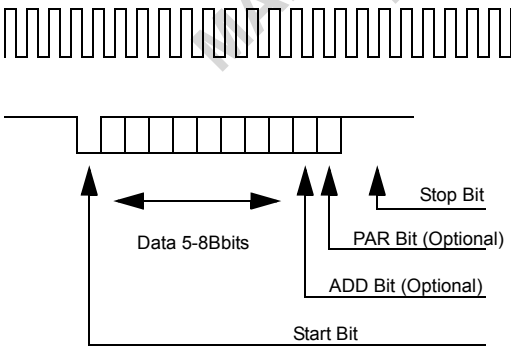
14.8.1 UART Receive/Transmit Operation

In UART mode an MPSC performs the following protocol functions:

- Start/Stop bit framing.
- Programmable data lengths (5-8 bits).
- Synchronous and asynchronous support.
- Message oriented data support.
- Parity detection and generation.
- Frame error, noise error, break, and idle detection.
- Support for HDLC over asynchronous control-octet transparency protocol.
- Multidrop operation with address recognition of up to two different addresses.

Figure 66 shows a typical UART frame format. A frame with a start bit is followed by 5-8 data bits. The address and parity bits are optional.

Figure 66: Typical UART Frame



At the end of a frame there are 1–2 stop bits before the transmitter can start to transmit the next frame. If there is nothing to transmit, a continuous ‘1’ is transmitted. This indicates that the line is idle.

The GT–64240A's UART samples each bit three times near its central point to define the bit value. A new start bit can be recognized only after the last stop bit sample is received. For example, at a 16x clock rate, the receiver can receive a start bit after a 9/16 bit time long stop bit.

When in UART mode, the RDW bit in the MMCRx should be set to configure the MPSCx data path to 8 bits.

A UART transceiver can work in asynchronous or is synchronous modes.

14.8.1.1 Asynchronous Mode

In Asynchronous mode, the DPLL sampling rate is set to 8x, 16x, or 32x of the data rate. The DPLL is synchronized by the falling edge of the start bit. If no error occurs, it maintains synchronization until the last bit in a frame is received.

Each bit is sampled three times around it's middle point. The bit value is determined by a majority vote. This feature helps to filter out noise from received data.

14.8.1.2 Isochronous Mode

In Isochronous mode, the DPLL sampling rate will be 1x the data rate. The receive data must be synchronized to the receive clock.

14.8.2 SDMAx Command/Status Field for UART Mode

When an MPSC is in UART mode the Command/Status field in the corresponding SDMAx descriptor has the following format:

Table 604: SDMAx Command/Status Field for UART Mode

Bit	Rx - Function	Tx - Function
0	PE - Parity Error. Last byte in buffer has parity error.	Reserved
1	CDL - CD Loss	CTSL - CTS Loss
2	Reserved	Reserved
3	FR - Framing Error	Reserved
5:4	Reserved	Reserved
6	OR - Data Overrun	Reserved
8:7	Reserved	Reserved
9	BR - Break Received while receiving data into this buffer	Reserved
10	MI - Max Idle. Buffer was closed due to Max_Idle timer expiration. **NOTE:** When this bit is set, the status of bit 0 is disregarded.	Reserved
11	A - Address. First byte in the buffer is an address. (Valid only in multidrop mode, '00' in point to point mode.)	Reserved

Table 604: SDMAx Command/Status Field for UART Mode (Continued)

Bit	Rx - Function	Tx - Function
12	AM - Address match. This bit will be set to '1' when a match occurred even if the V bit of the address is disabled.	Reserved
13	CT - The last byte in the buffer was precede by a transparency control octet.	Reserved
14	C - The last byte in a buffer is a user define control character.	Reserved
15	ES - Error Summary ES = PE || CDL || FR || OR	ES - Error Summary ES = CTSL
16	L - Last	L- Last
17	F - First	F - First
18	Reserved	P - Preamble. When set, the UART will send an IDLE preamble before buffer data. If data length is 0, only preamble IDLE will be send.
19	Reserved	A - Address. When set, buffer content will be sent with address bit on. Valid only in multidrop mode.
20	Reserved	NS - No Stop Bit. When set, data will be sent without stop bit.
22:21	Reserved	Reserved
23	EI - Enable Interrupt	EI - Enable Interrupt
29:24	Reserved	Reserved
30	AM - Auto Mode	AM - Auto Mode
31	O - Owner	O - Owner

14.8.3 MPSCx Protocol Configuration Register (MPCRx) for UART Mode

Figure 67: MPSCx Protocol Configuration Register (MPCRx) for UART Mode

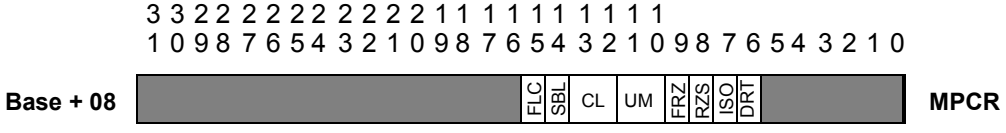


Table 605: MPSCx Protocol Configuration Register (MPCRx) for UART Mode

Bits	Field Name	Function	Initial Value
5:0	Reserved	Reserved.	0x0
6	DRT	Disable Rx on Tx. When DRT is set to '1' the Rx path is closed during Tx. This is useful in multidrop configurations when a user doesn't want to receive its own frames	0x0
7	ISO	Isochronous Mode 0 - Asynchronous Mode Start and stop bits are expected. RENC in the MMCRx should be programmed to NRZ and RCDV should be programmed to x8, x16 or x32 mode. (x16 is recommended for most applications). 1 - Isochronous Mode The receive bit stream is assumed to be synchronous to the receive clock. RCDV should be programmed to x1 mode.	0x0
8	RZS	Receive Zero Stop Bit 0 - Normal Mode At least one stop bit is expected. 1 - Zero Stop Bit The receiver continues reception when a stop bit is missing. If a '0' is received when stop bit is expected, this bit is considered a start bit. The FE (Framing Error) bit is set and the next bit to be received is considered to be data.	0x0
9	FRZ	Freeze Tx 0 - Restart Tx after freeze (normal operation). Transmission continues from the place it stopped. 1 - Freeze Tx at the end of the current character.	0x0

Table 605: MPSCx Protocol Configuration Register (MPCRx) for UART Mode (Continued)

Bits	Field Name	Function	Initial Value
11:10	UM	UART Mode 00 - Normal Mode Multidrop is disabled and IDLE line wake up is selected. A UART receiver wakes up after entering hunt mode upon receiving an IDLE character (all one character). 01 - Multi Drop Mode In multidrop mode, there is an additional Address/Data bit in each character. Upon receiving an address character, the UART receiver compares it to two 8-bit addresses stored in it's channel registers. If a match occurs, the receiver transfers the address and the following characters into a new buffer. If there is a no match, the character is discarded and the receiver is set to the hunt mode. If none of the addresses is valid (V bit in both address register is set to '0'), there is always a match and all the characters are transferred into the DRAM. Addresses are always be placed in a new buffer (Regardless of the V bit). The receiver receives characters until a new address is received, an abort character is received, an enter hunt command is issued, or until max idle counter expiration. Upon max idle counter expiration, the receiver is set to the hunt mode. 10 - Reserved. 11 - Reserved.	0x0
13:12	CL	Character Length 00 - 5 data bits 01 - 6 data bits 10 - 7 data bits 11 - 8 data bits	0x01
14	SBL	Stop Bit Length 0 - One stop bit 1 - Two stop bits	0x0
15	FLC	Flow Control 0 - Normal Mode The CTSM bit in the MMCRx determines the CTS* pin behavior. 1 - Asynchronous Mode When CTS* is negative, transmission stops at the end of the current character. When CTS* is asserted again, the transmission starts from the place it stopped. No CTS* lost is reported. Line is IDLE (MARK) during CTS* deassertion period.	0x0
31:16	Reserved	Reserved.	0x0

NOTE: When CD* is deasserted during frame reception UART behavior is different for multidrop and normal modes. In normal mode the UART hunts for an IDLE character (hunting starts when CD* is asserted again) before receiving valid start bit. In this mode, transmitting from a GT–64240A model to another should be with the 'P' bit in the buffer descriptor set. In multidrop mode, the UART receiver hunts for a start bit as soon as CD* is asserted again.

14.8.4 UART Stop Bit Reception and Framing Error

The UART receiver always expects to find a stop bit at the end of a character. If no stop bit is detected, the Framing Error (FE) bit is set in the receive descriptor. After a framing error, the reception process is controlled by the RZS and UM bits in the UART MPCRx. The various options are summarized in the table bellow.

Table 606: UART Stop Bit Reception and Framing Error

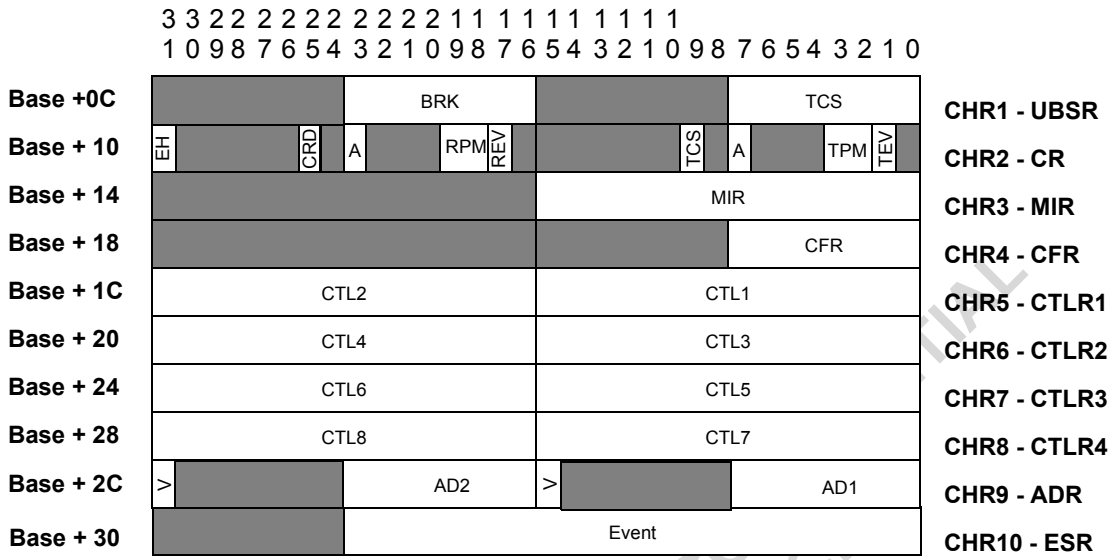
UM	RZS	Operation	Break Recognition
00	0	Go to hunt after missing a stop bit. The receiver is enabled after receiving a new IDLE char.	Single Break
00	1	The receiver tries to synchronize itself. The missing stop bit is considered as the following start bit and the reception process continues.	Two Break Sequence
01	0	Goes to hunt after missing stop bit. The receiver is enabled after receiving new address character.	Single Break
01	1	The receiver tries to synchronize itself. The missing stop bit is considered as the following start bit and the reception process continues.	Two Break Sequence

14.8.5 Channel Registers (CHxRx) for UART Mode

The MPSCx Channel Registers (CHxRx) are protocol dependent.

Figure 68 shows the CHxRx format in UART mode.

Figure 68: Channel Registers (CHxRx) for UART Mode



Unless otherwise is specified:

- '1' means set.
- '0' means unset.
- '0' is the default value after reset.

14.8.5.1 CHR1 - UART Break/Stuff Register (UBSR)

The UART Break/Stuff register has two fields: Break Count (BRK)(CHR1[23:16]) and Control Stuff Character (TCS) (CHR1[7:0]).

With the BRK field, the UART transmitter will starts to transmit break characters after receiving an abort command. The number for the break character to send is programmed into the BRK field.

For example, when BRK equals '0', no break character is transmitted. When BRK equals '1', one break character is transmitted.

A break character is a character with all '0's including it's stop bit.

Upon issuing a TCS command, the transmitter sends a TCS character after the current transmitting character. This allows a transmitter to bypass the normal pipeline when a special control character must be send (e.g. XON/XOFF).

Upon receiving a break character, the UART stops the reception process and moves to the hunt state. In a point to point configuration, the receiver is hunting for a new IDLE character. In a multidrop configuration, the receiver hunts for a new address character.

When the UART is in RZS=0 mode after receiving a break sequence, the descriptor is closed with BR bit (bit 9) set. In addition, a "break descriptor" also has the FE bit (bit 3) set and, if in odd parity, the PE bit (bit 0) is also set.

When the UART in RZS=1 mode, two consecutive break sequences are needed for proper break recognition. The first break character is not recognized. Instead, the UART receiver closes the descriptor with the FE bit (bit 3) set

and, if in odd parity, the PE bit (bit 0) will also be set. The second break will be recognized as a break and a descriptor will be closed with the BR bit (bit 9) set. In addition, a “break descriptor” will also have the FE bit (bit 3) set and, if in odd parity, the PE bit (bit 0) will also be set.

Table 607: CHR1 - Break/Stuff Register (UBSR)

Bits	Field Name	Function	Initial Value
7:0	TCS	Control Stuff Character	
15:8	Reserved	Reserved.	
23:16	BRK	Break Count	

14.8.5.2 CHR2 - Command Register (CR)

Table 608: CHR2 - Command Register (CR)

Bits	Field Name	Function	Initial Value
N/A	TD	Transmit Demand Fetch a descriptor and start transmission. Issued through the SDMAx Command Register.	
N/A	Stop	Stop Complete frame transmission and stop. (Go to IDLE). Issued through the SDMAx Command Register.	
0	Reserved	Reserved.	0x0
1	TEV	Tx Enable Vertical Redundancy Check 0 - VRC (parity) is disabled. 1 - VRC is enabled.	0x0
3:2	TPM	Transmit Parity Mode 00 - Odd 01 - Low (always 0) 10 - Even 11 - High (always 1)	0x0
6:4	Reserved	Reserved.	0x0

Table 608: CHR2 - Command Register (CR) (Continued)

Bits	Field Name	Function	Initial Value
7	A	Transmit Abort Aborts the transmission immediately (on byte boundaries) and goes to IDLE. The descriptor is not closed or incremented. After receiving an abort command, the GT–64240A halts the transmit process and starts sending a break sequence according to the BRK field in CHR1. **NOTE:** Command is not synchronized to byte.	0x0
8	Reserved	Reserved.	0x0
9	TTCS	Transmit TCS Character. The TCS character is transmitted after the current transmitted character. The transmitter then continues with the normal Tx sequence. The TCS command can be used to send out of band characters such as XOFF and XON.	0x0
16:10	Reserved	Reserved.	0x0
17	REV	Rx Enable Vertical Redundancy Check 0 - VRC (parity) is disabled. 1 - VRC is enabled.	0x0
19:18	RPM	Receive Parity Mode. 00 - Odd 01 - Low (always '0') 10 - Even 11 - High (always '1')	0x0
22:20	Reserved	Reserved.	0x0
23	A	Receive Abort Abort receive immediately and go to IDLE. The descriptor is not closed or incremented. The processor must issue a enter hunt command after an abort command in order to enable reception. The A bit is cleared upon entering IDLE state.	0x0
24	Reserved	Reserved.	0x0
25	CRD	Close Rx Descriptor When the CPU issues a CRD command, the current receive descriptor is closed and subsequent received data is DMA'd into a new buffer. If there is no active receive process, no action takes place. The GT–64240A clears the CRD bit upon closing the buffer status.[1]	0x0
30:26	Reserved	Reserved.	0x0

Table 608: CHR2 - Command Register (CR) (Continued)

Bits	Field Name	Function	Initial Value
31	EH	Enter Hunt Upon receiving an enter hunt command, the receive machine moves to a hunt state and continuously searches for an opening character. An opening character is considered an IDLE char in point to point mode (UM=00) or a matched address in multidrop mode. The EH bit is cleared upon entering a hunt state.	0x0

1. Usually, it takes a few cycles from the time the CRD bit is closed until the SDMAx actually closes the buffer. The SDMAx generates a maskable interrupt when closing a buffer if programed to do so.

The ET bit in the Main Configuration Register must be set to '1' before issuing any of the following commands:

- Transmit Demand
- Stop Transmission
- Transmit TCS Character
- Abort Transmission

The ER bit in the Main Configuration Register must be set to '1' before issuing any of the following commands:

- Enter Hunt
- Close Rx Descriptor
- Abort Reception

When the ET or ER bits are deasserted, the MPSCx transmit/receive channel is in low power mode (NO CLOCK).

NOTE: Issuing one of the above commands in this state leads to unpredictable results.

The CRCM in the MMCR must be set to 011 for LRC/VRC mode.

14.8.5.3 CHR3 - Max Idle Register (MIR)

This 16-bit value (CHR3[15:0]) defines the number of IDLE characters the receiver waits before it closes a descriptor and a maskable interrupt is generated.

When set to '0', the counter is disabled.

The counter is preloaded every time a non-IDLE character is received.

14.8.5.4 CHR4 - Control Filtering Register (CFR)

Bits 7:0 of the CFR register are the Bit Comparison Enable bits.

Setting a '1' in one of the BCE bits enables the control comparison for this bit.

Bit 29 (Z) of the CFR register is the UART enable bit for Debug Port mode. Setting this bit has meaning only when the CFR field is set to 0x0.

14.8.5.5 CHR5-8 - UART Control Character Registers

Figure 69 shows a UART control register format.

The CHAR field holds the pattern for the control character while bits 8-15 are used to control the GT–64240A's behavior when the control character is recognized.

Figure 69: UART Control Character Register Format

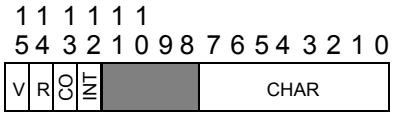


Table 609: UART Control Character Register Format

Bits	Field Name	Function	Initial Value
7:0	CHAR	The control character to sync on.	0x0
11:8	Reserved	Reserved.	0x0
12	INT	Interrupt 0 - No interrupt. 1 - Generate interrupt upon receiving this CHAR. In UART as Debug Port mode, an interrupt is generated on every character received.	0x0
13	CO	Control Octet (ISO 3309 Control Octet) Upon receiving a control octet, the control octet is discarded and the 6th bit (i.e. bit 5 in CHR5) of the following octet is complemented. The current buffer is closed with the CO bit asserted. **NOTE:** When the CO bit is set, the CHAR field must be programmed with '10111110' in order to be ISO-3309 compatible.	0x0
14	R	Reject 0 - Receive character and close the buffer. 1 - Reject character. The character is discarded, the buffer is closed and a maskable interrupt is generated.	0x0
15	V	Valid. 0 - Entry is not valid 1 - Entry is valid In UART as Debug Port mode, must be set to '1' to enable storing the character received to the RCRn field.	0x0

14.8.5.6 CHR9 - Address Register (ADR)

CHR9 holds the UART addresses for multidrop operation. The GT–64240A UART supports up to 2 addresses.

Upon receiving an address, the UART transfers the previous frame status to the SDMA. This causes the SDMA to close the previous frame descriptor and to locate the address in a new buffer.

There are two modes for address recognition operation. The first mode, setting of '1', allows the address and following characters to be transferred to the SDMA only if there is a match. The second mode, setting of '0', allows all frames to be passed to the SDMA. The CPU can use the M bit in the last frame descriptor to check if a match occurred.

14.8.5.7 CHR10 - UART Event Status Register (ESR)

The ESR register holds information on the transmit/receive channel condition. CHR10 can be read by the CPU for channel condition resolution.

Some changes in the channel condition can generate maskable interrupts, as shown in Table 610.

Table 610: CHR10 - UART Event Status Register (ESR)

Bits	Field Name	Event
0	CTS	Clear To Send Signal[1]
1	CD	Carrier Detect Signal[2]
2		Reserved.
3	TIDLE	Tx in Idle State[3]
4		Reserved.
5	RHS	Rx Hunt State Mode 0 - Rx in hunt state. 1 - Rx out of hunt state.
6		Reserved.
7	RLS	Rx Line Status
10:8		Reserved.
11	RLIDL	1 = Rx IDLE Line[4]
15:12		Reserved.
23:16	RCRn	Received Control Char n When the UART receiver recognizes a control character it sets the corresponding RCRn bit. (bit 16 (RCR1) corresponds to CTL1... bit 23 (RCR8) corresponds to CTL8). RCRn bits are cleared by write a 1 to the bit. In the UART as Debug Port mode, these bits contain the received byte.

1. Interrupt is generated when signal is deasserted during transmit.

2. Interrupt is generated when signal is deasserted during receive.

3. Interrupt is generated upon entering IDLE state.

4. Interrupt is generated upon change in line status.

14.8.6 Operating the MPSC as a Debug Port

14.8.6.1 UART Transmit (Tx): Using TCS to Generate Messages

To transmit data when UART is in Debug Port mode, the system can use the TTCS mechanism for transmitting data to the port before the SDRAM is accessible. The system can generate messages by loading the data to be transmitted to the MPSC CHR1 register's TCS field bits [7:0] (see Table 607 on page 441)and then forcing a transmission of this data by issuing a TTCS command in the MPSC CHR2 register (see Table 614 on page 452).

14.8.6.2 UART Receive (Rx): Using CFR to Capture Messages

To receive data when UART is in Debug Port mode, the system can use the UART in Debug Port mode enable bit [29] in the MPSC CFR (CHR4) register (see Section 14.8.5.4 "CHR4 - Control Filtering Register (CFR)" on page 443).

Setting this bit together with a value of 0x0 in the CFR field of the same register, enables the UART as Debug Port mode. In this mode, the MPSC receives and stores the data in the MPSC CHR10 register's RCRn field bits [23:16] (see Table 610 on page 445). The data is received/stored according to the Control Character setting of the MPSC CHR5 register's CTL1 field of (see Section 14.8.5.5 "CHR5-8 - UART Control Character Registers" on page 444). Setting the Valid bit [15] enables storage of received data. Setting the Interrupt bit [12] enables maskable interrupt generation on every byte received. In this case, bit [6] of the relevant MPSC Mask register (see Table 616 on page 454) should be set to '1'. After reading the received data in the RCRn field, the field must be cleared by writing '1' to all the set bits.

NOTE: If an interrupt was generated, bit [6]of the relevant MPSC Cause register must be cleared by writing '0' to this bit.

14.8.6.3 Sample Code (for MPSC0):

```
// declare all variables and set their values to 0.

// set the UART control characters in char5 :
// CHAR - is 8f ,
// Reserved - 0
// INT - 1
// CO - 0
// R - 0
// V - 1
SetUartCtl(0,"8F",0,1,0,0,1);
MPSC[0].BCE = 0x0; /* CFR = 0x00  */

MMCRL.MODE = 0x4; /* UART Mode */
```

```
MMCRL.ER = 1; /* enable recive */
MMCRL.ET = 1; /* enable transmit */
MMCRH.RCDV = 1; /* x8 clock mode */

Write(MPSC0_MAIN_CONFIGURATION_LOW,MMCRL);
Write(MPSC0_MAIN_CONFIGURATION_HIGH,MMCRH); /* activate mpsc */

MPCR.ISO = 0; /* asynchronous */
MPCR.SBL = 1; /* 0; */ /* one stop bit */
MPCR.CL = 3;   /* number of data bits 8 */
Write(MPSC0_PROTOCOL_CONFIGURATION,MPCR); /* config uart */

CHR4.CFR = 0;  /* MPSC as UART switch chr4[7:0](cfr)=0 */
CHR4.Z = 1;  /* MPSC as UART switch chr4[29](z)=1 */
Write(CHANNEL0_REGISTER4,CHR4); /* start MPSC as UART mode */

CHR5.CTL[0].V = 1; /* enables storing of the received data */
CHR5.CTL[0].INT = 1; /* enable maskable interrupt */
Write(CHANNEL0_REGISTER5,CHR5); /* config MPSC as UART */

MPCS0MR.RCC =1 /* enable bit 6 of MPSC Interrupt Cause register */
Write(MPSC0_MASK,MPCS0MR); /* enable MPSC as UART byte recieved interrupt */

CHR2.EH = 1; /* enter hunt command */
Write(CHANNEL0_REGISTER2,CHR2); /* start hunting */

Byte = 0x55;

for (i=0; i<1; ) { /* loop forever */

    CHR1.TCS = Byte; /* set byte to transmit */
    Write(CHANNEL0_REGISTER1,CHR1); /* config MPSC as UART transmit byte */

    CHR2.TCS = 1; /* MPSC as UART transmit command */
    Write(CHANNEL0_REGISTER2,CHR2); /* start MPSC as UART transmit */

    Read(CHANNEL0_REGISTER2,&CHR2); /* poll tcs command */
    while (CHR2 & 0x00000020 == 1) { /* check TCS bit */
      Read(CHANNEL0_REGISTER2,&CHR2);
    }

    Read(MPSC0_CAUSE,&MPSC0CR); /* poll interrupt bit */
    while (MPSC0CR & 0x00000040 == 0) {/* check RCC interrupt bit */
      Read(MPSC0_CAUSE,&MPSC0CR);
```

```
        }

        Read(CHANNEL0_REGISTER10,&CHR10); /* read recieved value */
        Byte = (CHR10 >> 16) & 0xff; /* extract byte value */

        Write(CHANNEL0_REGISTER10,CHR10); /* clear  RCRn field in the CHR10 register*/
        Write (MPSC0_CAUSE,MPSC0CR & 0xffffffbf); /* clear interrupt bit */

} /* end loop */
```

14.9 Transparent Mode

In transparent mode, the GT–64240A does not perform any protocol dependent data processing.

However, it gives the processor hardware assistance for bit reception, using the GT–64240A's powerful SDMA engines, and some assistance in synchronization, interrupt generation, and frame construction. The CPU also uses the built-in CRC engine for CRC generation and checking. In any case, CRC bits are transferred into memory for CPU use.

In transparent mode, the channel is fully configured from the MMCRx and no mode is defined by the channel registers.

A transparent channel is synchronous.If it is not serviced on time, underrun and overrun errors can occur.

The receiver can use external sync using the CD* input or synchronize itself on a SYNC sequence according to the RSYL bits in the MMCRx.

14.9.1 SDMAx Command/Status Field for Transparent Mode

When an MPSC is in Transparent mode the Command/Status field in the corresponding SDMAx descriptor has the following format:

Table 611: SDMAx Command/Status Field for Transparent Mode

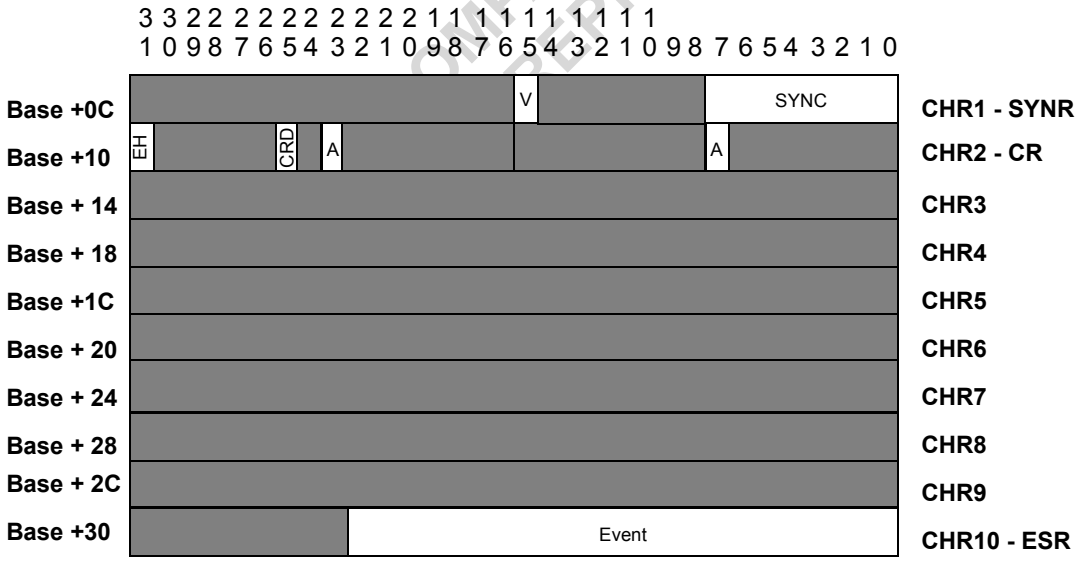
Bit	Rx - Function	Tx - Function
0	CE - CRC/LRC Error	Reserved.
1	CDL - CD Loss	CTSL - CTS Loss
2	DE - Decoding Error	Reserved.
3	Reserved.	Reserved.
4	Reserved.	Reserved.
5	Reserved.	Reserved.
6	OR - Data Overrun	UR - Data Underrun
14:7	Reserved.	Reserved.

Table 611: SDMAx Command/Status Field for Transparent Mode (Continued)

Bit	Rx - Function	Tx - Function
15	ES - Error Summary ES = CE || CDL || DE || OR	ES - Error Summary ES = CTSL || UR
16	L - Last	L - Last
17	F - First	F - First
21:18	Reserved.	Reserved.
22	Reserved.	GC - Generate BCC/LRC.
23	EI - Enable Interrupt	EI - Enable Interrupt
29:24	Reserved.	Reserved.
30	AM - Auto Mode	AM - Auto Mode
31	O - Owner	O - Owner

14.9.2 Channel Registers (CHxRx) for Transparent Mode

Figure 70: Channel Registers (CHxRx) for Transparent Mode



Unless otherwise is specified:

- '1' means set.
- '0' means unset.
- '0' is the default value after reset.

14.9.2.1 CHR1 - SYNC Register (SYNR)

The SYNC Register holds the synchronization for the channel receiver. After reset it holds the value of 7E in the SYNC field. The user should right the appropriate values before enabling the Rx/Tx machines.

There are two basic synchronization options for a transparent channel: Transparent Mode Synchronization and Transmitter Synchronization. The Transparent Mode Synchronization has two synchronization options, selected by setting RSYL[24:23] in the MMCRx.

The Transparent Mode Synchronization has two synchronization options. They are also selected by setting the RSYL [24:23] bits in the MMCRx.

NOTE: For more information about setting RSYL[24:23], see Table 583 on page 407.

Table 612: Transparent Mode Synchronization Options

Synchronization Option	Function
External Synchronization	(RSYL = '00') The receiver starts to receive data whenever CD* is asserted and stops receiving data when CD* is deasserted (if CDM=0) or when the CPU issues an Enter Hunt Command.
Sync Hunt	RSYL = '01', '10', or '11' (nibble, byte or two bytes sync) The receiver hunts for the sync pattern, as defined by RSYL. When the synch pattern is recognized, the receiver starts to receive data. The receive process stops when CD* is deasserted and CDM=0 or when the CPU issues an enter hunt command. If bit 15 is set, there is no transfer of the SYNC characters to the receiver. The syncs are stripped until the first data character is received, and are not calculated in the packet CRC. If bit 15 is reset, sync characters that appear after the sync pattern is recognized are regarded as data. On the transmitter side, in sync hunt mode, two sync characters are always sent at the beginning of a frame. **NOTE:** When RSYL equals 01, the Sync pattern is defined by bits [7:4] of the Sync Register.

There are two mode of transmit synchronization in transparent mode. They are selected by setting TSYN[12] in the MMCRx, see Table 613 on page 451.

Table 613: Transmitter Mode Synchronization Options

Synchronization Option	Function
TSYN = 0	Synchronization is achieved whenever CTS* is asserted.
TSYN=1	Synchronization is achieved after receiver synchronization and CTS* is asserted. The transmitter always starts to transmit on the receive byte boundaries. In external synchronization, when CTS* is asserted, the transmitter starts to transmit 8 bits after CD* assertion. In sync hunt mode, when CTS* is asserted, the transmitter starts to transmit 8 bits after sync recognition. If CTS* is deasserted after the receiver gains synchronization, the transmitter waits to the byte boundary before it starts to transmit.

14.9.2.2 CHR2 - Command Register (CR)

Table 614: CHR2 - Command Register (CR)

Bits	Field Name	Function	Initial Value
N/A	TD	Transmit Demand Fetch a descriptor and start transmission. Issued at SDMAx Command Register.	
N/A	Stop	Stop Complete frame transmission and stop. (Go to IDLE). Issued at SDMAx Command Register.	
6:0	Reserved	Reserved.	0x0
7	A	Abort Transmission Aborts the transmission immediately (on byte boundaries) and goes to IDLE. The descriptor is not closed or incremented. **NOTE:** Command is not synchronized to byte.	0x0
22:8	Reserved	Reserved.	0x0
23	A	Abort Reception Abort receive immediately and go to IDLE. The descriptor is not closed or incremented. The processor must issue an enter hunt command after an abort command in order to enable reception. The A bit is cleared upon entering IDLE state.	0x0
24	Reserved	Reserved.	0x0
25	CRD	Close Rx Descriptor When the CPU issues a CRD command the current receive descriptor is closed and the following received data is SDMA'd into a new buffer. If there is no active receive in progress, no action takes place.	0x0
30:26	Reserved	Reserved.	0x0
31	EH	Enter Hunt Upon receiving an enter hunt command, the receive machine moves to a hunt state and continuously searches for an opening sync or an external sync. If the enter hunt command is received during a frame reception, the current descriptor is closed with a CRC error. The EH bit is cleared upon entering a hunt state.	0x0

The ET bit in the Main Configuration Register must be set to ‘1’ before issuing any of the following commands:

- Transmit Demand
- Stop Transmission
- Abort Transmission

The ER bit in the Main Configuration Register must be set to ‘1’ before issuing any of the following commands:

- Enter Hunt
- Close Rx Descriptor
- Abort Reception

When the ET or ER bits are deasserted, the MPSCx transmit/receive channel is in low power mode (NO CLOCK).

NOTE: Issuing one of the above commands in this state leads to unpredictable results.

14.9.2.3 CHR10 - Transparent Event Status Register (ESR)

The ESR register holds information on the transmit/receive channel condition. CHR10 can be read by the CPU for channel condition resolution. Some changes in the channel condition can generate maskable interrupts, as shown Table 615.

Table 615: CHR10 - Transparent Event Status Register (ESR)

Bits	Field Name	Event
0	CTS	Clear To Send Signal[1]
1	CD	Carrier Detect Signal[2]
2		Reserved.
3	TIDLE	Tx in Idle State[3]
4		Reserved.
5	RHS	Rx in HUNT state
11:6		Reserved.
12	DPCS	1 = DPLL Carrier Sense
15:13		Reserved.
23:16	RCRn	Received Control Character n When the transparent receiver recognizes a control character it sets the corresponded RCRn bit (bit 16 (RCR1) corresponds to CTL1... bit 23 (RCR8) correspond to CTL8). RCRn bits are cleared by writing ‘1’ to the bit.

1. Interrupt is generated when signal is deasserted during transmit
2. Interrupt is generated when signal is deasserted during receive
3. Interrupt is generated upon entering IDLE state

14.10 MPSC Cause and Mask Registers

Table 616: MPSC Cause and Mask Register

- MPSC0 Cause Offset: 0xb804
- MPSC0 Mask Offset: 0xb884
- MPSC1 Cause Offset: 0xb80c
- MPSC1 Mask Offset: 0xb88c

Bits	Field Name	Description	Initial Value
0	MpscxRx	MPSCx Normal Rx Interrupt Summary Logical OR of (unmasked) bits 4-7 below. This bit is read only.	0x0
1	MpscxRxErr	MPSCx Rx Error Interrupt Summary Logical OR of (unmasked) bits 8-11 below. This bit is read only.	0x0
2		Reserved.	0x0
3	MpscxTxErr	MPSCx Tx Error Interrupt Summary Logical OR of (unmasked) bits 13-15 below. This bit is read only.	0x0
4	MpscxRLSC	MPSCx Rx Line Status Change (from to IDLE)	0x0
5	MpscxRHNT	MPSCx Rx Entered HUNT State	0x0
6	MpscxRFSC/ MpscxRCC	MPSCx Rx Flag Status Change (HDLC mode) MPSCx Received Control Character (Bisync, Uart modes)	0x0
7	MpscxRCSC	MPSCx Rx Carrier Sense Change (DPLL decoded carriers sense)	0x0
8	MpscxROVR	MPSCx Rx Overrun	0x0
9	MpscxRCDL	MPSCx Rx Carrier Detect Loss	0x0
10	MpscxRCKG	MPSCx Rx Clock Glitch	0x0
11	MpscxBPER	MPSCx Bisync Protocol Error (valid only in Bisync mode)	0x0
12	MpscxTEIDL	MPSCx Tx Entered IDLE State	0x0
13	MpscxTUDR	MPSCx Tx Underrun	0x0
14	MpscxTCTSL	MPSCx Tx Clear To Send Loss	0x0
15	MpscxTCKG	MPSCx Tx Clock Glitch	0x0
31:16	Reserved	Reserved.	0x0

NOTE: When a mask bit is set to ‘1’, the corresponding cause bit is also enabled.

14.11 MPSC Registers

Table 617: MPSC Signals Routing Register Map

Register	Offset	Page
MPSC Routing Register (MRR)	0xb400	page 397
Rx Clock Routing Register (RCRR)	0xb404	page 398
Tx Clock Routing Register (TCRR)	0xb408	page 398

Table 618: MPSCs Interrupts Register Map

Register	Offset	Page
MPSC0 Cause	0xb804	page 454
MPSC0 Mask	0xb884	
MPSC1 Cause	0xb80c	
MPSC1 Mask	0xb88c	

Table 619: MPSC0 Register Map

Register	Offset	Page
Main Configuration Low (MMCRL0)	0x8000	page 401
Main Configuration High (MMCRH0)	0x8004	page 407
Protocol Configuration (MPCR0)	0x8008	page 413
Channel0 Register1 (CH0R1)	0x800c	The functionality of each CHxRx is protocol dependent. Detailed descriptions of the CHRs are given in the following sections: • Section 14.6 “HDLC Mode” on page 411 • Section 14.7 “BISYNC Mode” on page 420 • Section 14.8 “UART Mode” on page 434 • Section 14.9 “Transparent Mode” on page 448
Channel0 Register2 (CH0R2)	0x8010	
Channel0 Register3 (CH0R3)	0x8014	
Channel0 Register4 (CH0R4)	0x8018	
Channel0 Register5 (CH0R5)	0x801c	
Channel0 Register6 (CH0R6)	0x8020	
Channel0 Register7 (CH0R7)	0x8024	
Channel0 Register8 (CH0R8)	0x8028	
Channel0 Register9 (CH0R9)	0x802c	
Channel0 Register10 (CH0R10)	0x8030	
Channel0 Register11 (CH0R11)	0x8034	

Table 620: MPSC1 Register Map

Register	Offset	Page
MPSC1 Main Configuration Low (MMCRL1)	0x9000	page 401
MPSC1 Main Configuration High (MMCRH1)	0x9004	page 407
MPSC1 Protocol Configuration (MPCR1)	0x9008	page 413
Channel1 Register1 (CH1R1)	0x900c	The functionality of each CHxRx is protocol dependent. Detailed descriptions of the CHRs are given in the following sections: • Section 14.6 “HDLC Mode” on page 411 • Section 14.7 “BISYNC Mode” on page 420 • Section 14.8 “UART Mode” on page 434 • Section 14.9 “Transparent Mode” on page 448
Channel1 Register2 (CH1R2)	0x9010	
Channel1 Register3 (CH1R3)	0x9014	
Channel1 Register4 (CH1R4)	0x9018	
Channel1 Register5 (CH1R5)	0x901c	
Channel1 Register6 (CH1R6)	0x9020	
Channel1 Register7 (CH1R7)	0x9024	
Channel1 Register8 (CH1R8)	0x9028	
Channel1 Register9 (CH1R9)	0x902c	
Channel1 Register10 (CH1R10)	0x9030	
Channel1 Register11 (CH1R11)	0x9034	

15. MPSC Serial DMAs (SDMA)

15.1 Overview

There are two SDMA channels on the GT–64240A that are dedicated for moving data between the serial communications channels (MPSCs) and memory buffers. Each SDMA channel consists of a DMA engine for receiving and one for transmitting.

Each SDMA channel has two dedicated FIFOs for data buffering (for a total of 32 FIFOs). All FIFOs are 256 bytes deep.

For receive operations, the MPSC moves received data into the dedicated FIFO of the corresponding SDMA. Then, using descriptors set up by the user, the SDMA moves the data into memory buffers. For transmit operations, the SDMA uses descriptors set up by the user to move data out of buffers into the dedicated FIFO. The MPSC moves the data down to the serial communications link.

The SDMA channel descriptors use a chained data structure. They work without CPU interference after appropriate initialization. SDMA channels can be programed to generate interrupts on buffer or frame boundaries.

When enabled, he receive SDMAs run freely and expect to find a valid descriptor, when one is required. When a receive SDMA channel accesses an invalid descriptor, the receive SDMA process halts with a resource error status indication.

When enabled, the transmit SDMAs run freely until the end of the descriptor chain is reached. When a transmit SDMA accesses an invalid descriptor and the last descriptor was not marked as an end of frame descriptor, the transmit SDMA process halts with resource error status indication.

The SDMAs arbitrate for accessing the descriptors and buffers. A standard round-robin scheme is used for arbitration between them.

15.2 SDMA Descriptors

All SDMA data transfers are done via a chained link of descriptors. The following rules must be followed when using the GT–64240A SDMA descriptors:

- Descriptor length is 4LW and it must be 4LW aligned (i.e. Descriptor_Address[3:0]=0000).
- Descriptors may reside anywhere in CPU address space except NULL address, which is used to indicate end of descriptor chain.
- In normal mode (HDLC and Transparent) RX buffers associated with RX descriptors must be 64-bit aligned. Minimum size for RX buffers is 8 bytes. In low latency, or byte, mode (BISYNC, UART, and Transparent) RX buffers have no alignment restrictions.
- Tx buffers associated with TX descriptors can start in any byte location.
- SDMA RX and TX buffers are limited to 64Kbytes.

Figure 71: SDMA Descriptor Format

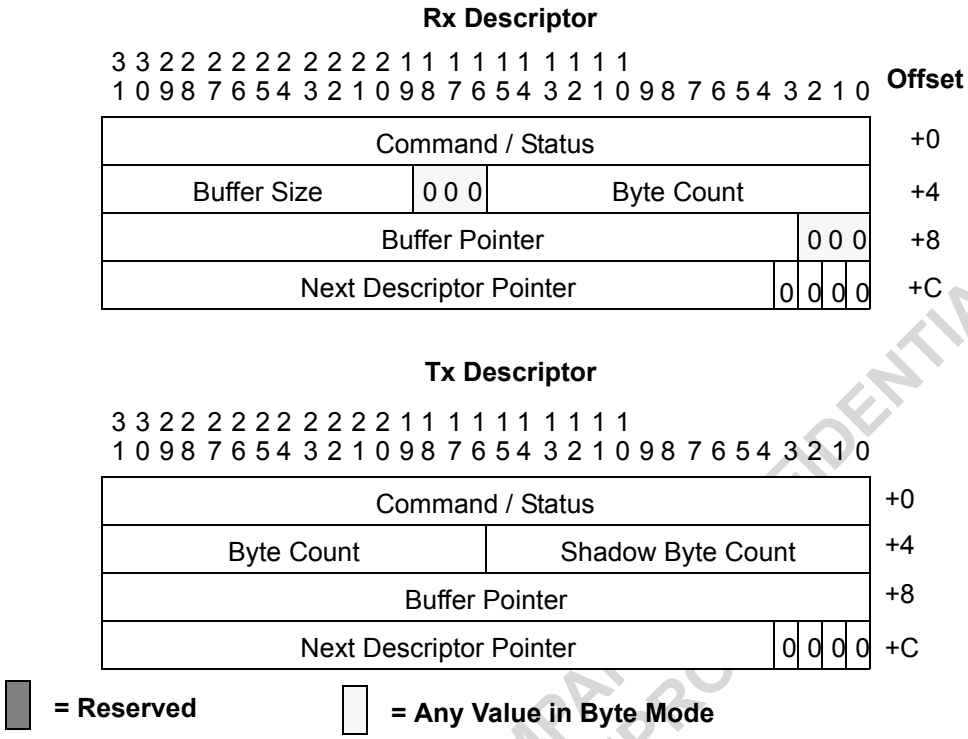


Table 621 through Table 625 provide detailed information about the descriptor fields.

Table 621: SDMA Descriptor - Command/Status word

Bits	Field Name	Function
Contains commands bits that instruct the SDMA how to process a buffer and status bits that the SDMA updates upon closing a descriptor. The CPU uses the status bits to evaluate the buffer status. Except for bits 31, 30, 23, 17, and 16, the definition of the bits vary depending on which mode is being used. See: • Section 14.6.2 “SDMAx Command/Status Field for HDLC Mode” on page 411. • Section 14.7.3 “SDMAx Command/Status Field for BISYNC Mode” on page 423. • Section 14.8.2 “SDMAx Command/Status Field for UART Mode” on page 435. • Section 14.9.1 “SDMAx Command/Status Field for Transparent Mode” on page 448.		
15:0		Determined by the mode selected.
16	L	Last Bit Indicates last buffer of a frame.
17	F	First Bit Indicates first buffer of a frame.
22:18		Determined by the mode selected.

Table 621: SDMA Descriptor - Command/Status word (Continued)

Bits	Field Name	Function
23	EI	Enable Interrupt The GT–64240A generates a maskable interrupt when closing descriptor with EI bit set. **NOTE:** If the RIFB bit is set in the SDMA configuration register, a Rx interrupt is generated only if this is the last descriptor associated with a received frame. In this case, EI bit setting is masked for intermediate descriptors.
29:24	Reserved	Determined by the mode selected.
30	AM	Auto Mode When set, the SDMA won't clear the Owner bit of the descriptor at the end of buffer processing.
31	O	Owner Bit When set to'1', the buffer can be processed by the GT–64240A. When set to '0', the buffer can be processed by the CPU. An SDMA process will halt when a descriptor with owner bit set to '0' is fetched.

Table 622: SDMA Descriptor - Buffer Size, Byte Count (Rx Descriptor)

Bits	Field Name	Function
15:0	Byte Count	The number of bytes that were actually written by the SDMA into the buffer. This number is never greater than Buffer Size. The CPU must initialize the Byte Count field with 0x0000.
31:16	Buffer Size	The buffer size field is valid only in receive descriptors and is reserved in transmit descriptors.The field is written by the CPU and read by the GT–64240A. When the buffer byte counter of a SDMA receive channel reaches the buffer size value, the SDMA will close the buffer descriptor and will move to the next buffer. Buffer Size must be a multiple of 8 when the MPSC is programmed to work in normal mode (HDLC and Transparent). Buffer Size can be arbitrary when working in low bandwidth mode (BISYNC, UART, and Transparent).

Table 623: SDMA Descriptor - Byte Count, Shadow Byte Count (Tx Descriptor)

Bits	Field Name	Function
15:0	Shadow Byte Count	The CPU must initialize this field with a value identical to the Byte Count field. The GT–64240A subtracts the number of bytes actually transmitted from this parameter. Usually the GT–64240A writes '0' in this field when closing a descriptor. However, when the transmit SDMA halts due to a transmit error, this number can be used to determine the number of bytes that were fetched into the GT–64240A. Setting both the Byte Count and Shadow Byte Count to '0' will cause the SDMA to close the descriptor and move to the next descriptor, if both or neither of the F and L bits are set. Setting Byte Count and Buffer Size to '0' in transmit descriptors with one of the F or L bits set will lead to unpredictable behavior.
31:16	Byte Count	Byte count is the number of bytes to be transmitted. Zero byte counters are not supported with retransmission. Do not use zero byte buffers with LAP-D protocol.

Table 624: SDMA Descriptor - Buffer Pointer

Bits	Field Name	Function
31:0	Buffer Pointer	32-bit pointer to the beginning of the buffer associated with the descriptor. The buffer can reside anywhere in memory or PCI address space.

Table 625: SDMA Descriptor - Next Descriptor Pointer

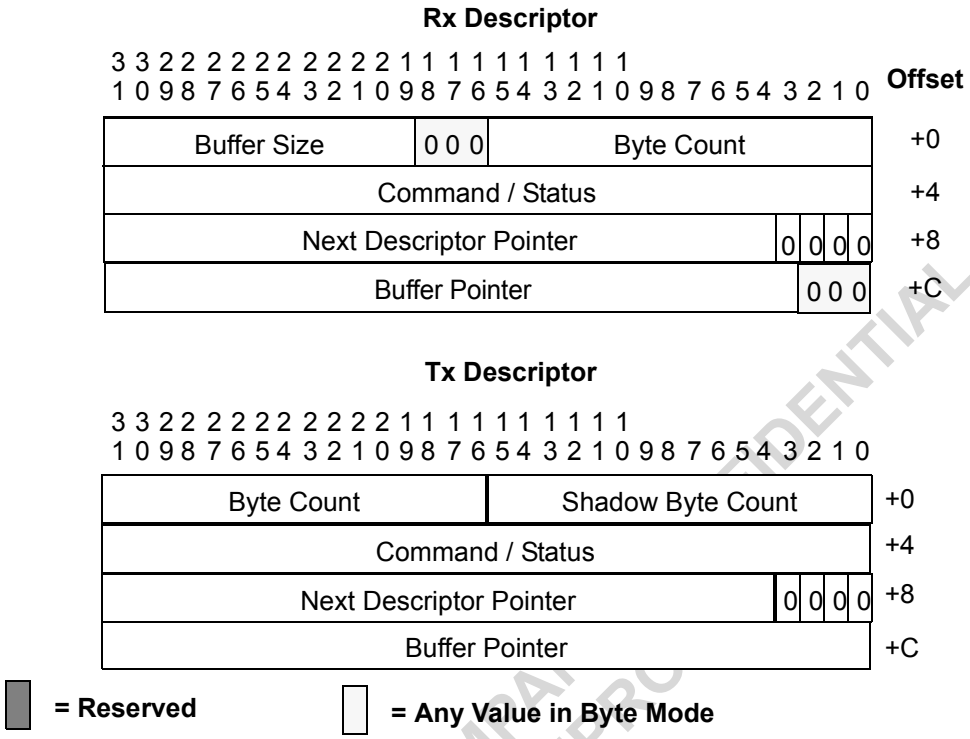
Bits	Field Name	Function
31:4	Next Descriptor Pointer	32-bit Next Descriptor pointer to the beginning of the next descriptor in the chain. A descriptor can reside anywhere in memory or PCI space. Bits [3:0] must be set to'0'. DMA operation is stopped when a NULL value in the Next Descriptor Pointer is encountered.

15.3 SDMA Descriptor Word Swapping

When the Serial DMA fetches a descriptor from DRAM, it expects the descriptor in Little Endian format. However, since the DRAM byte order matches the CPU, consideration must be made when working with a CPU configured for Big Endian.

If the software prepares the descriptors using 32-bit load/store operation, the descriptors must be word swapped when placed in DRAM. This means that the Tx and Rx descriptor must appear as described in Figure 72.

Figure 72: SDMA Descriptor Word Swapped Format



15.4 SDMA Configuration Register (SDC)

Each SDMA has a dedicated configuration register (SDCx). The SDC must be initialized before enabling the SDMA channel.

Figure 73: SDMAx Configuration Register (SDCx)

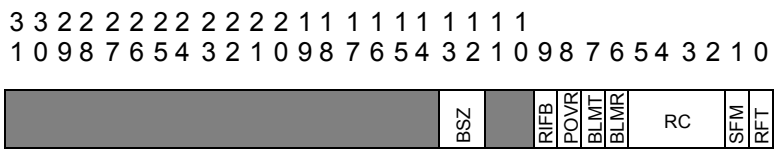


Table 626: SDMA Configuration Register (SDCx)

- Channel0 Offset: 0x4000
- Channel1 Offset: 0x6000

Bits	Field Name	Function	Initial Value
0	RFT	Receive FIFO Threshold 0 - 8 bytes 1 - Half FIFO (128 bytes) **NOTE:** When working with an 8-bit data path, the threshold is always one byte regardless of the RFT value. It is recommended that RFT bit be set to ‘0’ in this case. When RFT is set to ‘0’, the SDMA will not burst. It will transfer one word (64 bits) on each transfer.	0x0
1	SFM	Single Frame Mode 0 - Multi frame mode The GT–64240A reads as many frames as needed into the FIFO to keep the transmit FIFO full. The FIFO can handle more than one frame at a time. 1 - Single frame mode The first descriptor is not fetched before the current frame’s last descriptor is closed. **NOTE:** The SFM bit must be set to ‘1’ for HDLC Collision mode, and BISYNC protocols. It is also recommended for UART. When the SFM bit is set to ‘0’, CTS Lost cannot be reported in the correct descriptor/frame. In LAN HDLC mode SFM must be set for proper operation.	0x0

Table 626: SDMA Configuration Register (SDCx) (Continued)

- Channel0 Offset: 0x4000
- Channel1 Offset: 0x6000

Bits	Field Name	Function	Initial Value
5:2	RC	Retransmit Count In collision modes (LAP-D), after executing a backoff procedure RC times, the Tx SDMA closes the buffer with a Retransmit Limit (RL) error, a maskable interrupt is generated, and the SDMA goes to the OFF state. A new Transmit Demand command must be issued to start a new transmission process. When RC field is 0000, the GT–64240A tries to retransmit forever. The CPU needs to issue an abort command in order to stop the retransmit process.	0xf
6	BLMR	Rx Big Little/Endian Receive Mode The GT–64240A supports big or little endian configuration per channel for maximum system flexibility. The BLMR bit only affects data movements. 0 - Big endian convention 1 - Little endian convention	0x1
7	BLMT	Tx Big/Little Endian Transmit Mode The GT–64240A supports big or little endian configuration per channel for maximum system flexibility. The BLMT bit only affects data movements. 0 - Big endian convention 1 - Little endian convention	0x1
8	POVR	PCI Override When set, causes the SDMA to direct all its accesses in PCI_0 direction, overriding normal address decoding process.	0x0
9	RIFB	Receive Interrupt on Frame Boundaries When set, the SDMA Rx generates interrupts only on frame boundaries (i.e. after writing the frame status to the descriptor).	0x0
11:10	Reserved	Reserved.	0x0
13:12	BSZ	Burst Size Sets the maximum burst size for SDMA transactions: 00 - Burst is limited to 1 64bit words 01 - Burst is limited to 2 64bit words 10 - Burst is limited to 4 64bit words 11 - Burst is limited to 8 64bit words	0x0
31:14	Reserved	Reserved.	0x0

15.5 SDMA Command Register (SDCMx)

Each SDMA has a dedicated SDMA Command Register (SDCMx) register to control its DMA process.

Figure 74: SDMA Command Register (SDCMx)

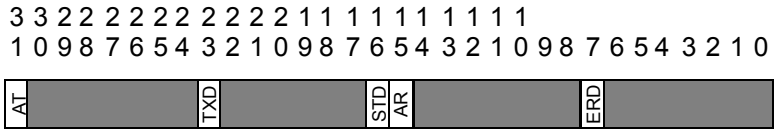


Table 627: SDMA Command Register (SDCMx)

- Channel0 Offset: 0x4008
- Channel1 Offset: 0x6008

Bits	Field Name	Function	Initial Value
6:0	Reserved	Reserved.	0x0
7	ERD	Enable Rx DMA When set to '1', the Rx SDMA will fetch the 1st descriptor and will be ready for a receive frame. The GT–64240A clears ERD when the GT–64240A receive SDMA has a resource error or when the CPU issues an abort command.	0x0
14:8	Reserved	Reserved.	0x0
15	AR	Abort Receive The CPU sets the AR bit when it needs to abort a receive SDMA channel operation. When the AR bit is set, the SDMA aborts its operation and goes to IDLE state. No descriptor is closed. The GT–64240A clears both the AR and ERD bits when entering IDLE state. The CPU must poll bit 15. When it is '0', the GT–64240A has completed the abort sequence. After an abort the CPU should write the 1st descriptor address and then set ERD bit to '1'.	0x0
16	STD	Stop Tx The SDMA stops transmission at the end of frame (i.e. at the end of buffer with L bit set to '1'). After transmitting the last buffer, the transmit SDMA goes to IDLE state. The GT–64240A clears the TXD bit when entering IDLE state. After the SDMA stops, the CPU must write the first descriptor address and than set the TXD bit to '1'. The GT–64240A signals the CPU with interrupt when the stop procedure is accomplished.	0x0
22:17	Reserved	Reserved.	0x0

Table 627: SDMA Command Register (SDCMx) (Continued)

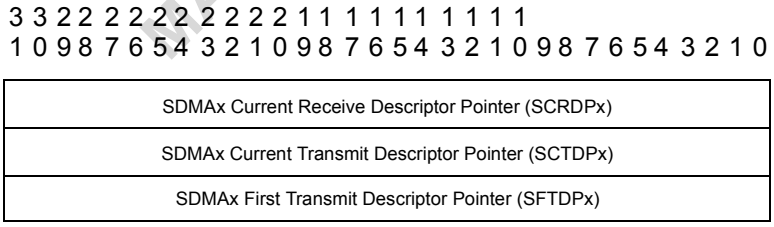
- Channel0 Offset: 0x4008
- Channel1 Offset: 0x6008

Bits	Field Name	Function	Initial Value
23	TXD	Tx Demand When this bit is set to '1', the Tx DMA will fetch the first descriptor and will start the transmission process. The GT–64240A clears TXD when it successfully ends an SDMA transmit process. It also clears TXD when a resource error occurs, when the transmit process is halted due to channel error (i.e. CTS# lost), or when the CPU issues an abort command.	0x0
30:24	Reserved	Reserved.	0x0
31	AT	Abort Transmit The CPU sets the TRD bit to '1' when it needs to abort a transmit SDMA channel operation. When the TRD bit is set, the SDMA aborts its operation and goes to IDLE state. No descriptor is closed. The GT–64240A clears both the TRD and TXD bits when entering IDLE state. The CPU must poll bit 31. When it is '0', the GT–64240A has completed the abort sequence. After an abort, the CPU must write the first descriptor address and than set TXD bit to '1'.	0x0

15.6 SDMA Descriptor Pointer Registers

Each SDMA channel has three 32-bit registers that reside in a special descriptor's Dual Port memory located in the internal address space of the GT–64240A.

Figure 75: SDMA Descriptor Pointer Registers



15.6.1 SDMA Current Receive Descriptor Pointer (SCRDP)

SCRDPx points to the current receive descriptor in memory. The CPU must write this register with the first descriptor address before enabling the SDMA receive channel. When a SDMA receive channel is enabled it will fetch the first descriptor pointed to by SCRDPx as part of its SDMA starting procedure.

15.6.2 SDMA Current Transmit Descriptor Pointer (SCTDP)

SCTDPx points to the current transmit descriptor in memory. The CPU must write this register with the first descriptor address before enabling the SDMA transmit channel. When a SDMA transmit channel is enabled it will fetch the first descriptor pointed to by SCRDPx as part of its SDMA starting procedure.

15.6.3 SDMA First Transmit Descriptor Pointer (SFTDP)

SFTDPx points to the first descriptor in a transmit frame. The CPU must write this register with the first descriptor address before enabling the SDMA transmit channel. The SDMA transmit controller uses the SFTDP when it needs to restart a transmission after collision (HDLC mode only). The GT–64240A updates the content of SFTDP each time it fetches a descriptor with the F (first) bit set to '1'.

NOTE: The CPU must write the same value to both SCTDP and SFTDP before enabling the corresponding SDMA transmit channel.

15.7 Transmit SDMA

15.7.1 Transmit SDMA Definitions

- **SOF** (Start Of Frame descriptor): Descriptor with F (First) bit set to '1'.
- **EOF** (End Of Frame descriptor): Descriptor with L (Last) bit set to '1'.

F and L bits are set by the CPU before releasing a descriptor to the GT–64240A for transmission.

A frame starts with a SOF descriptor and ends with a EOF descriptor. A frame can consist of one buffer or split over many buffers. If a frame is stored in one buffer, the associated descriptor will have both the F and L bits set to '1'. In a non-frame oriented protocol (e.g. BISYNC or UART), it is recommended that both F and L bits be set to '1' for each buffer.

15.7.2 Transmit SDMA Flow

The following steps are executed during a normal transmit SDMA process:

1. Before enabling a SDMA Tx channel the CPU must prepare a valid descriptor with the owner bit set to '1'.
2. The CPU must then write the first descriptor address to both SCTDP and SFTDP registers.
3. The CPU issues a Transmit Demand command. The SDMA controller will then fetch the first descriptor and will start the SDMA process.
4. When buffer transmission is completed, the SDMA will close the buffer descriptor by setting the correct transmit status and writing '0' in the Owner Bit, returning the buffer to the CPU.

15.7.3 Retransmit in HDLC (LAP-D) mode

When working in collision mode (see MPSC section), the GT–64240A retransmits if collision occurs before the SDMA fetches the 3rd descriptor. If the frame consists of more than two buffers, the user must assure that there is enough data in the first two buffers to compensate for this behavior. The GT–64240A can buffer up to 256 bytes in its internal Tx FIFO. This should be considered when preparing a LAP-D transmit frame.

15.7.4 Transmit SDMA Notes

The transmit SDMA process is *frame oriented*.

The Transmit SDMA does not clear the frame's first descriptor ownership bit until the last descriptor associated with this frame is closed. The transmit SDMA then writes '0' to the first descriptor Owner bit and generate an interrupt if the EI bit of the first descriptor is set.

The transmit SDMA stops the DMA process whenever it reaches a descriptor with NULL (0x00000000) value in the NDP field or when it fetches a descriptor with Owner Bit set to '0'. In such cases, the SDMA controller clears the TxD bit before returning to IDLE state.

In normal operation, the transmit SDMA never expects to find a NULL NextDescriptorPointer or Not-Owned descriptor in the middle of a frame. When this occurs, the transmit SDMA controller aborts, the TxD bit is cleared, and a Tx RESOURCE ERROR maskable interrupt is generated.

NOTE: In collision mode, if a collision occurs exactly one clock cycle after a resource error, the GT–64240A ignores the resource error and retransmit the frame.

When the CPU wants to interfere with the transmit process without corrupting the ongoing transmit process, it can issue a STOP command by writing '1' to the STD bit in the SDMA command register. The transmit SDMA controller stops after completing the transmission of the active frame.

When issuing an STD command TXD is reset to '0' upon entering IDLE state. The CPU can then issue a new Transmit Demand command to restart the SDMA process.

15.8 Receive SDMA

15.8.1 Receive SDMA Definitions

Table 628: SDMA Definitions

Term	Definition
SOF	Start Of Frame descriptor Descriptor with F (First) bit set to '1'.
EOF	End Of Frame descriptor Descriptor with L (Last) bit set to '1'.

F and L bits are set by the CPU before releasing a descriptor to the GT–64240A.

A frame starts with an SOF descriptor and ends with an EOF descriptor. A frame can be contained in one buffer or split over many buffers. If a frame is stored in one buffer, the associated descriptor will have both F and L bits set to '1'.

15.8.2 Receive SDMA Flow

The following steps are executed during a normal transmit SDMA process:

1. Before enabling a SDMA Rx channel the CPU must prepare a valid descriptor with the owner bit set to ‘1’.
2. The CPU must then write the descriptor address to the SCRDP register before enabling the receive SDMA channel.
3. The CPU writes ‘1’ to the ERD bit in the SDCM register, enabling the receive SDMA channel.
4. Normally the receive SDMA controller will then run continuously, processing received data from the MPSC.

NOTES: The receive SDMA controller never expects to encounter a descriptor with owner bit set to ‘0’ or a NULL value (0x00000000) in the NDP field. If this occurs, the receive SDMA aborts and a maskable Rx RESOURCE ERROR interrupt is generated.

Use the receive abort command for the CPU to stop the receive SDMA. It is the CPU’s responsibility to properly restart the descriptor chain.

15.9 SDMA Interrupt and Mask register (SDI and SDM)

Each SDMA channel has two maskable interrupt sources. One is for resource error events and the other one is for descriptor closed events.

15.9.1 Resource Error Interrupt

When a receive SDMA encounters a NULL descriptor pointer or a not owned descriptor, a Resource Error interrupt is generated. A Resource Error interrupt is generated whenever a transmit SDMA encounters a NULL descriptor pointer or a not-owned descriptor in a middle of a frame.

NOTE: When the GT–64240A encounters a descriptor with Owner bit set to 0, it still expects to find that all the other fields of the descriptor are legitimate. A descriptor with Owner bit set to 0, with non-legitimate fields (such as Start Of Frame descriptor with F (First) bit not set to ’1’) can lead to unpredictable behavior.

15.9.2 Descriptor/Frame Closed Interrupt

When a SDMA channel closes a descriptor with the EI (Enable Interrupt) bit set to '1', a Descriptor Closed interrupt is generated.

NOTES: In case the RIFB bit is set in the SDMA configuration register, an interrupt is generated by the Rx channel only on receive frame boundaries.

The correct operation of the frame level interrupt requires all Rx descriptors to have their EI bit set.

Table 629: SDMA Cause and Mask Register

- Cause Offset: 0xb800
- Mask Offset: 0x b880

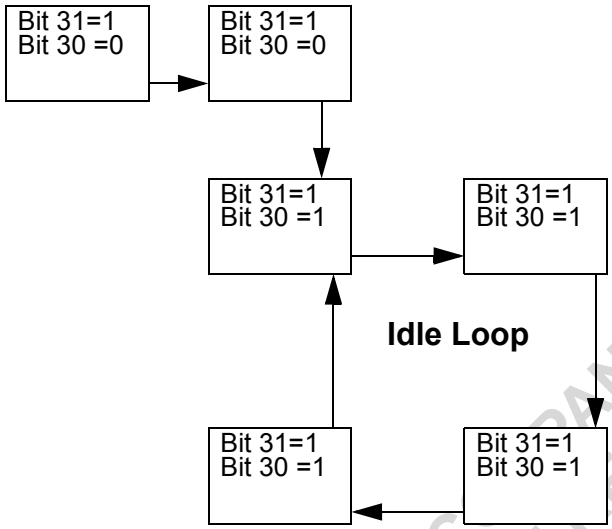
Bits	Field Name	Function	Initial Value
0	Sdma0RxBuf	SDMA Channel 0 Rx Buffer Return Indicates that SDMA0 Rx closed a descriptor and returned the associated buffer to CPU ownership.	0x0
1	Sdma0RxErr	SDMA Channel 0 Rx Error Indicates that a Rx resource error occurred.	0x0
2	Sdma0TxBuf	SDMA Channel 0 Tx Buffer Return Indicates that SDMA0 Tx closed a descriptor and returned the associated buffer to CPU ownership.	0x0
3	Sdma0TxEnd	SDMA Channel 0 Tx End Indicates that a Tx resource error occurred or that the Tx DMA moved to IDLE after a stop command. Also set when Tx retransmit limit is reached in HDLC mode.	0x0
7:4	Reserved	Reserved	0x0
8	Sdma1RxBuf	SDMA Channel 1 Rx Buffer Return Indicates that SDMA1 Rx closed a descriptor and returned the associated buffer to CPU ownership.	0x0
9	Sdma1RxErr	SDMA Channel 1 Rx Error Indicates that a Rx resource error occurred.	0x0
10	Sdma1TxBuf	SDMA Channel 1 Tx Buffer Return Indicates that SDMA1 Tx closed a descriptor and returned the associated buffer to CPU ownership.	0x0
11	Sdma1TxEnd	SDMA Channel 1 Tx End Indicates that a Tx resource error occurred or that the Tx DMA moved to IDLE after a stop command. Also, set when Tx retransmit limit is reached in HDLC mode.	0x0
31:12	Reserved	Reserved	0x0

15.10 SDMA in Auto Mode

The CPU can set bit 30 in the command/status field of transmit or receive descriptors directing the GT–64240A to work in Auto Mode.

When working with an Auto Mode descriptor, the GT–64240A SDMA works as usual except that it does not clear the Ownership bit when closing the descriptor. The CPU can use this for example to cause the GT–64240A to transmit endlessly (until CPU intervention).

Figure 76: Using Auto Mode to Create Idle Loop



15.11 SDMA Registers

Table 630: SDMA Register Map

Register	Offset	Page
Channel0 Configuration Register (SDC0)	0x4000	page 462
Channel0 Command Register (SDCM0)	0x4008	page 464
Channel0 Rx Descriptor	0x4800 - 0x480f	Not to be accessed during normal operation.
Channel0 Current Rx Descriptor Pointer (SCRDP0)	0x4810	page 465
Channel0 Tx Descriptor	0x4c00 - 0x4c0f	Not to be accessed during normal operation.
Channel0 Current Tx Descriptor Pointer (SCTDP0)	0x4c10	page 466

Table 630: SDMA Register Map (Continued)

Register	Offset	Page
Channel0 First Tx Descriptor Pointer (SFTDP0)	0x4c14	page 466
Channel1 Configuration Register (SDC1)	0x6000	page 462
Channel1 Command Register (SDCM1)	0x6008	page 464
Channel1 Rx Descriptor	0x6800 - 0x680f	Not to be accessed during normal operation.
Channel1 Current Rx Descriptor Pointer (SCRDP1)	0x6810	page 465
Channel1 Tx Descriptor	0x6c00 - 0x6c0f	Not to be accessed during normal operation.
Channel1 Current Tx Descriptor Pointer (SCTDP1)	0x6c10	page 466
Channel1 First Tx Descriptor Pointer (SFTDP1)	0x6c14	page 466

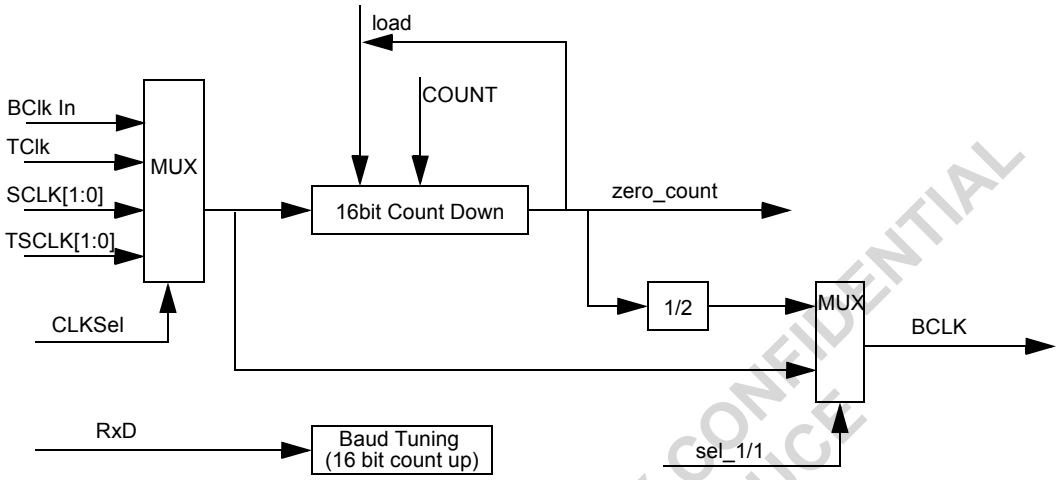
Table 631: SDMA Interrupts Register Map

Register	Offset	Page
SDMA Cause	0xb800	page 469
SDMA Mask	0xb880	

16. BAUDE RATE GENERATORS (BRG)

There are two baud rate generators (BRGs) in the GT–64240A. Figure 77 shows a BRG block diagram.

Figure 77: Baud Rate Generator Block Diagram



16.1 BRG Inputs and Outputs

There are 5 clock inputs to the baud rate generators (BRGs). One MPP pin can be programmed to function as clock input to the BRGs. Additionally, each of the serial input clocks can be used as a BRG clock. Finally, TClk is also an option.

When a BRG is enabled, it loads the Count Down Value (CDV), from the BRG configuration register, into its count down counter. When the counter expires (i.e. reaches zero), the BRG clock output, BCLK, is toggled and the counter reloads.

16.2 BRG Baud Tuning

A baud tuning mechanism can be used to adjust the generated clock rate to the receive clock rate.

When baud tuning is enabled, the baud tuning mechanism monitors for a start bit, i.e. High-to-Low transition. When a start bit is found, the baud tuning machine measures the bit length by counting up until the next Low-to-High transition. The count-up value of the BRG is then loaded into the Count Up Value (CUV) register and a maskable interrupt is generated signaling the CPU that the bit length value is available. The CPU reads the value from the CUV and adjusts the CDV to the requested value.

The CUV can be used to adjust the CDV, in the BRG configuration register, to the requested value.

16.3 BRG Registers

Table 632: BRG Registers Map

Register Name	Offset	Page
BRG0		
BRG0 Configuration Register (BCR0)	0xb200	page 473
BRG0 Baud Tuning Register (BTR0)	0xb204	page 474
BRG1		
BRG1 Configuration Register (BCR1)	0xb208	page 473
BRG1 Baud Tuning Register (BTR1)	0xb20c	page 474
BRG Interrupts		
BRG Cause Register	0xb834	page 474
BRG Mask Register	0xb8b4	

Table 633: BRGx Configuration Register (BCR)

Bits	Field Name	Function	Initial Value
15:0	CDV	Count Down Value The user programs the CDV field to define the baud rate that the BRG generates. CDV is loaded into the BRG counter every time it reaches 0. The actual baud rate is: $$BaudRate = \frac{inputClockRate}{(CDV+1) \times 2}$$ When CDV is 0x0000, the generated baud rate is equal to the input clock rate.	0x0
16	En	Enable BRG 0 - Disabled (Output clock is clamped to 0.) 1 - Enabled.	0x0
17	RST	Reset BRG 0 - No Op. 1 - Reset BRG counter to 0.	0x0

Table 633: BRGx Configuration Register (BCR) (Continued)

Bits	Field Name	Function	Initial Value
22:18	CLKS	Clock Source (input clock to the BRG) 0x0 - BclkIn (from MPP) 0x1 - Reserved 0x2 - SCLK0 (from S0 port) 0x3 - TSCLK0 (from S0 port) 0x4,0x5 - Reserved 0x6 - SCLK1 (from S1 port) 0x7 - TSCLK1 (from S1 port) 0x8 - TClk 0x9 - 0x1f - Reserved	0x10010
31:23	Reserved	Reserved.	0x0

Table 634: BRGx Baud Tuning register (BTR)

NOTE: If the BRG is written for a clock source that is inactive, this register cannot be accessed, see Table 633 bits [22:18].

Bits	Field Name	Function	Initial Value
15:0	CUV	Count Up Value **NOTE:** These bits are read only.	0x0
31:16		Reserved.	0x0

Table 635: BRG Cause and Mask Register

- Cause Offset: 0xb834
- Mask Offset: 0xb8b4

Bits	Field Name	Function	Initial Value
0	BTR0	Baud Tuning 0 interrupt	0x0
1	BTR1	Baud Tuning 1 interrupt	0x0
31:2		Reserved.	0x0

NOTE: When a mask bit is set to ‘1’, the corresponding cause bit is also enabled.

17. WATCHDOG TIMER

The GT–64240A internal watchdog timer is a 32-bit count down counter that can be used to generate a non-maskable interrupt or reset the system in the event of unpredictable software behavior.

After the watchdog is enabled, it is a free running counter that needs to be serviced periodically in order to prevent its expiration.

NOTE: WDE and WDNMI watchdog output pins are multiplexed on the MPP pins (see Section 19. “Pins Multiplexing” on page 480). The watchdog timer can be activated only after configuring two MPP pins to act as WDE and WDNMI.

17.1 Watchdog Registers

Table 636: Watchdog Configuration Register (WDC), Offset 0xb410

Bits	Field Name	Function	Initial Value
23:0	Preset_VAL	This field holds the 24 most significant bits which the watchdog counter loads each time it is enabled or serviced. After reset, this field is set to 0xFF.FFFF. The preset value is equal to {0xPreset_VAL,FF}.	0xFF.FFFF
24:25	CTL1	A write sequence of ‘01’ followed by ‘10’ into CTL1 disables/enables the watchdog.	00
27:26	CTL2	A write sequence of ‘01’ followed by ‘10’ to CTL2 services the watchdog timer.	00
28	Reserved	Reserved.	0
29	NMI	Non-Maskable Interrupt When the watchdog counter reaches a value equal to NMI_VAL, this bit is asserted. This pin can be used to drive the processor’s NMI* pin. This bit is read only.	1
30	WDE	Watchdog Expiration When the watchdog counter expires, this bit is asserted. The WDE* pin can be used to reset the entire system. This bit is read only.	1
31	EN	Enable 0 - Watchdog is disabled, counter is loaded with Preset_VAL. NMI and WDE are set to ‘1’. 1 - Watchdog is enabled. This bit is read only.	0

Table 637: Watchdog Value Register (WDV), Offset 0xb414

Bits	Field Name	Function	Initial Value
23:0	NMI_VAL	NMI_VAL are the 24 least significant bits of a 32-bit value. The upper 8 bits are always '00'. When the Watchdog counter reaches a value equal to the NMI value NMI* pin is asserted. The actual NMI value is a 32-bit number equal to {0x00,NMI_VAL}.	0x000.0000
31:24	Reserved	Reserved.	0

17.2 Watchdog Operation

After reset, the watchdog is disabled.

The watchdog must be serviced periodically in order to avoid NMI or reset (WDE*). Watchdog service is performed by writing '01' to CTL2, followed by writing '10' to CTL2. Upon watchdog service, the GT–64240A clears the NMI and WDE bits (if set) and reloads the Preset_VAL into the watchdog counter.

A write sequence of '01' followed by '10' into CTL1 disables/enables the watchdog. The watchdog's current status can be read in bit 31 of WDC. When disabled, the GT–64240A sets the NMI and WDE bits (if clear) and reloads the Preset_VAL into the watchdog counter.

Preset_VAL and NMI_VAL can be changed while the watchdog is enabled. However, Preset_VAL will affect the watchdog only after it is loaded into the watchdog counter (e.g. after watchdog service).

If the watchdog is not serviced before the counter reaches NMI_VAL, a non-maskable interrupt event occurs. a watchdog expiration event occurs. The NMI bit is reset, asserting low the NMI* pin.

In order to deassert the NMI* and/or WDE* pins, the watchdog must be serviced, disabled or the GT–64240A must be reset. The GT–64240A holds WDE* asserted for the duration of 16 system cycles after reset assertion.

18. General Purpose Port

GT–64240A contains a 32-bit General Purpose Port (GPP).

Each of the GPP pins can be assigned to act as a general purpose input or output pin and can be used to register external interrupts (when assigned as input pin). The GPP is multiplexed on the GT–64240A MPP pins (see Section 19.1 “MPP Multiplexing” on page 480 section for more information).

18.1 GPP Control Registers

The GT–64240A includes GPP I/O Control and GPP Level Control registers.

The I/O Control register determines the direction for each GPP pin. Setting a bit to ‘1’ configures the associated GPP pin to act as output pin. Setting a bit to ‘0’ configures the GPP pin as input pin.

The Level Control register determines the polarity for each GPP pin. Setting a bit to ‘1’ configures the associated GPP pin to be active low. Setting a bit to ‘0’ configures the GPP pin to be active high. The GT–64240A negates an active low input pin before latching it inside. It inverts an active low output pin before driving it outside.

18.2 GPP Value Register

The GT–64240A includes a 32-bit GPP Value register. Each GPP pin has an associated bit.

For pins configured as input pins, the associated bits are read only, and contains the value of the pins. When an input GPP pin is configured as asserted low, the value latched in GPP Value register is the negated value of the pin.

For pins configured as output pins, the associated bits are read/write. The value written to the GPP Value register bits is driven on the associated GPP output pins (inverted in case of active low pin).

18.3 GPP Interrupts

The GPP input pins can be used to register external interrupts. The GT–64240A supports both edge sensitive and level sensitive interrupts.

If the Comm Unit Arbiter Control register’s GPP_INT bit is set to ‘0’ (see Table 530 on page 334), the external interrupts are treated as edge trigger interrupts. An assertion of a GPP input pin (toggle from ‘0’ to ‘1’ in case of active high pin, from ’1’ to ’0’ in case of active low pin), results in setting the corresponding bit in GPP Interrupt Cause register.

NOTE: The GPP pin must be kept active for at least one TClk cycle to guarantee that the interrupt is registered.

If not masked by the GPP Interrupt Mask register, the GPP interrupt may cause a CPU or PCI interrupt. If a mask bit is set to ‘1’, interrupt is enabled. A mask register setting has no affect on registering GPP interrupts into the GPP Interrupt Cause register.

Interrupt is deasserted as soon as software clears the corresponding bit in the GPP Interrupt cause register (write ‘0’).

If Comm Unit Atbiter Control register's GPP_INT bit is set to '1', the external interrupts are treated as level interrupts. In this mode, an interrupt is always generated when one of the GPP Value register bits is asserted and it is not masked by the GPP Interrupt Mask register (GPP Interrupt Cause register is not used for the interrupt generation).

NOTE: In this mode, the interrupt handler clears the interrupt directly on the originating device.

18.4 General Purpose Port Registers

Table 638: GPP Register Map

Register	Offset	Page
GPP I/O Control	0xf100	page 478
GPP Level Control	0xf110	page 478
GPP Value	0xf104	page 479
GPP Interrupt Cause	0xf108	page 479
GPP Interrupt Mask	0xf10c	page 479

Table 639: GPP I/O Control, Offset: 0xf100

Bits	Field Name	Function	Initial Value
31:0	GPP I/O	GPP Input/Output Select 0 - Input 1 - Output	0x0

Table 640: GPP Level Control, Offset: 0xf110

Bits	Field Name	Function	Initial Value
31:0	GPP Level	GPP Input Level Select 0 - Active high 1 - Active low	0x0

Table 641: GPP Value, Offset: 0xf104

Bits	Field Name	Function	Initial Value
31:0	GPP Value	GPP Pins Values If the GPP pin is programed as an input pin, it's associated bit is a Read Only bit containing the GPP pin value (or negated value in case of an asserted low pin). If programed as an output pin, it is a read/write bit. It's programed value is driven on the GPP pin.	0x0

Table 642: GPP Interrupt Cause, Offset: 0xf108

Bits	Field Name	Function	Initial Value
31:0	Cause	GPP Interrupt Cause Bits Set to '1' upon GPP input pin assertion. Only cleared by the CPU or PCI writing '0'.	0x0

Table 643: GPP Interrupt Mask, Offset: 0xf10c

Bits	Field Name	Function	Initial Value
31:0	Mask	GPP Interrupts Mask If a bit is set to '1', it's associated GPP interrupt is enabled.	0x0

19. PINS MULTIPLEXING

The GT–64240A has two 15-bit E0 and E1 ports, two 7-bit S0 and S1 ports, and 32 MPP pins. Ports E0 and E1 are used as Ethernet ports, ports S0 and S1 as MPSC ports, and the MPP pins as peripheral functions.

19.1 MPP Multiplexing

The GT–64240A contains 32 Multi Purpose Pins. Each one can be assigned to a different functionality through MPP Control register. The MPP pins can be used as hardware control signals to the GT–64240A different interfaces (UMA control, DMA control, PCI arbiter signals), or as General Purpose Ports.

Table 644 shows each MPP pins' functionality as determined by the MPP Multiplex register.

Table 644: MPP Function Summary

MPP[0]	MPP[1]	MPP[2]	MPP[3]	MPP[4]	MPP[5]	MPP[6]	MPP[7]
GPP[0]	GPP[1]	GPP[2]	GPP[3]	GPP[4]	GPP[5]	GPP[6]	GPP[7]
DMAReq[0]*	DMAAck[0]*	DMAReq[1]*	DMAAck[1]*	DMAReq[2]*	DMAAck[2]*	DMAReq[3]*	DMAAck[3]*
MGNT*	MREQ*	PME0*	PME1*	PME1*	PME0*	MGNT*	MREQ*
EOT[7]	EOT[7]	EOT[6]	EOT[6]	EOT[5]	EOT[5]	EOT[4]	EOT[4]
TCEn[3]	TCTcnt[3]*	TCEn[2]	TCTcnt[2]	TCEn[1]	TCTcnt[1]	TCEn[0]	TCTcnt[0]
DBurst*	InitAct	InitAct	DBurst*	InitAct	DBurst*	DBurst*	InitAct
Int[0]*	Int[1]*	Int[2]*	Int[3]*	Int[0]*	Int[1]*	Int[2]*	Int[3]*
GNT0[0]*	REQ0[0]*	GNT0[1]*	REQ0[1]*	GNT0[2]*	REQ0[2]*	GNT0[3]*	REQ0[3]*
GNT1[0]*	REQ1[0]*	GNT1[1]*	REQ1[1]*	GNT1[2]*	REQ1[2]*	GNT1[3]*	REQ1[3]*
WDNMI*	WDE*	WDNMI*	WDE*	BClkIn	BClkIn	BClkOut0	BClkOut0
Debug[0]	Debug[1]	Debug[2]	Debug[3]	Debug[4]	Debug[5]	Debug[6]	Debug[7]
MPP[8]	**MPP[9]**	**MPP[10]**	**MPP[11]**	**MPP[12]**	**MPP[13]**	**MPP[14]**	**MPP[15]**
GPP[8]	GPP[9]	GPP[10]	GPP[11]	GPP[12]	GPP[13]	GPP[14]	GPP[15]
DMAReq[4]*	DMAAck[4]*	DMAReq[5]*	DMAAck[5]*	DMAReq[6]*	DMAAck[6]*	DMAReq[7]*	DMAAck[7]*
MGNT*	MREQ*	PME1*	PME0*	PME0*	PME1*	MGNT*	MREQ*
EOT[3]	EOT[3]	EOT[2]	EOT[2]	EOT[1]	EOT[1]	EOT[0]	EOT[0]
TCEn[7]	TCTcnt[7]	TCEn[6]	TCTcnt[6]	TCEn[5]	TCTcnt[5]	TCEn[4]	TCTcnt[4]
DBurst*	InitAct	InitAct	DBurst*	InitAct	DBurst*	DBurst*	InitAct
Int[0]*	Int[1]*	Int[2]*	Int[3]*	Int[0]*	Int[1]*	Int[2]*	Int[3]*
GNT0[4]*	REQ0[4]*	GNT0[5]*	REQ0[5]*	GNT0[4]*	REQ0[4]*	GNT0[3]*	REQ0[3]*
GNT1[4]*	REQ1[4]*	GNT1[5]*	REQ1[5]*	GNT1[4]*	REQ1[4]*	GNT1[3]*	REQ1[3]*
WDNMI*	WDE*	WDNMI*	WDE*	BClkOut0	BClkOut0	BClkIn	BClkIn
Debug[8]	Debug[9]	Debug[10]	Debug[11]	Debug[12]	Debug[13]	Debug[14]	Debug[15]

Table 644: MPP Function Summary (Continued)

MPP[16]	MPP[17]	MPP[18]	MPP[19]	MPP[20]	MPP[21]	MPP[22]	MPP[23]
GPP[16]	GPP[17]	GPP[18]	GPP[19]	GPP[20]	GPP[21]	GPP[22]	GPP[23]
DMAReq[0]*	DMAAck[0]*	DMAReq[1]*	DMAAck[1]*	DMAReq[2]*	DMAAck[2]*	DMAReq[3]*	DMAAck[3]*
MGNT*	MREQ*	PME0*	PME1*	PME1*	PME0*	MGNT*	MREQ*
EOT[7]	EOT[7]	EOT[6]	EOT[6]	EOT[5]	EOT[5]	EOT[4]	EOT[4]
TCEn[3]	TCTcnt[3]	TCEn[2]	TCTcnt[2]	TCEn[1]	TCTcnt[1]	TCEn[0]	TCTcnt[0]
DBurst*	InitAct	InitAct	DBurst*	InitAct	DBurst*	DBurst*	InitAct
Int[0]*	Int[1]*	Int[2]*	Int[3]*	Int[0]*	Int[1]*	Int[2]*	Int[3]*
GNT0[0]*	REQ0[0]*	GNT0[1]*	REQ0[1]*	GNT0[2]*	REQ0[2]*	GNT0[3]*	REQ0[3]*
GNT1[0]*	REQ1[0]*	GNT1[1]*	REQ1[1]*	GNT1[2]*	REQ1[2]*	GNT1[3]*	REQ1[3]*
WDNMI*	WDE*	WDNMI*	WDE*	BClkIn	BClkIn	BClkOut0	BClkOut0
Debug[16]	Debug[17]	Debug[18]	Debug[19]	Debug[20]	Debug[21]	Debug[22]	Debug[23]
MPP[24]	**MPP[25]**	**MPP[26]**	**MPP[27]**	**MPP[28]**	**MPP[29]**	**MPP[30]**	**MPP[31]**
GPP[24]	GPP[25]	GPP[26]	GPP[27]	GPP[28]	GPP[29]	GPP[30]	GPP[31]
DMAReq[4]*	DMAAck[4]*	DMAReq[5]*	DMAAck[5]*	DMAReq[6]*	DMAAck[6]*	DMAReq[7]*	DMAAck[7]*
MGNT*	MREQ*	PME1*	PME0*	PME0*	PME1*	MGNT*	MREQ*
MGNT*	MREQ*	PME1*	PME0*	PME0*	PME1*	MGNT*	MREQ*
EOT[3]	EOT[3]	EOT[2]	EOT[2]	EOT[1]	EOT[1]	EOT[0]	EOT[0]
TCEn[7]	TCTcnt[7]	TCEn[6]	TCTcnt[6]	TCEn[5]	TCTcnt[5]	TCEn[4]	TCTcnt[4]
DBurst*	InitAct	InitAct	DBurst*	InitAct	DBurst*	DBurst*	InitAct
Int[0]*	Int[1]*	Int[2]*	Int[3]*	Int[0]*	Int[1]*	Int[2]*	Int[3]*
GNT0[4]*	REQ0[4]*	GNT0[5]*	REQ0[5]*	GNT0[4]*	REQ0[4]*	GNT0[3]*	REQ0[3]*
GNT1[4]*	REQ1[4]*	GNT1[5]*	REQ1[5]*	GNT1[4]*	REQ1[4]*	GNT1[3]*	REQ1[3]*
WDNMI*	WDE*	WDNMI*	WDE*	BClkOut0	BClkOut0	BClkIn	BClkIn
Debug[24]	Debug[25]	Debug[26]	Debug[27]	Debug[28]	Debug[29]	Debug[30]	Debug[31]

NOTE: Since each pin might act as output or input pin, depending on its configured functionality, all MPP pins wake up after reset as GPP input pins.

19.2 Serial Ports Multiplexing

The GT–64240A has three fast Ethernet controllers, and two MPSC controllers. These controllers interface with other devices through ports E0, E1, S0, S1. The functionality of the different ports is determined via Serial Ports Multiplex register as shown in Table 645 through Table 647.

Table 645: E0 Port Select Summary

	Functionality		
Pin	MII	RMII	Initial State
E0[0]	MTxEN0	MTxEN0	3-state
E0[1]	MTxCLK0	REF_CLK	3-state
E0[2]	MTxD0[0]	MTxD0[0]	3-state
E0[3]	MTxD0[1]	MTxD0[1]	3-state
E0[4]	MTxD0[2]	MTxD2[0]	3-state
E0[5]	MTxD0[3]	MTxD2[1]	3-state
E0[6]	MCOL0	CRS_DV0	3-state
E0[7]	MRxCLK0	----	3-state
E0[8]	MRxD0[0]	MRxD0[0]	3-state
E0[9]	MRxD0[1]	MRxD0[1]	3-state
E0[10]	MRxD0[2]	MRxD2[0]	3-state
E0[11]	MRxD0[3]	MRxD2[1]	3-state
E0[12]	MRxER0	MTxEN2	3-state
E0[13]	MRxDV0	CRS_DV2	3-state
E0[14]	MCRS0	----	3-state

Table 646: E1 Port Select Summary

	Functionality	
Pin	MII	RMII
E1[0]	MTxEN1	MTxEN1
E1[1]	MTxCLK1	----
E1[2]	MTxD1[0]	MTxD1[0]
E1[3]	MTxD1[1]	MTxD1[1]
E1[4]	MTxD1[2]	----

Table 646: E1 Port Select Summary (Continued)

Pin	Functionality	
	MII	RMII
E1[5]	MTxD1[3]	----
E1[6]	MCOL1	----
E1[7]	MRxCLK1	----
E1[8]	MRxD1[0]	MRxD1[0]
E1[9]	MRxD1[1]	MRxD1[1]
E1[10]	MRxD1[2]	----
E1[11]	MRxD1[3]	----
E1[12]	MRxER1	----
E1[13]	MRxDV1	CRS_DV1
E1[14]	MCRS1	----

NOTES: E1 port has no multiplex control. It is always connected directly to Ethernet controller 1. This controller can act in MII or RMII mode.

Since E0 port pins might act as input or output pins, depending on their configurable functionality, these pins wake up after reset as 3-state input pins, to prevent contentions on the board.

Table 647: S0 Port Select Summary

Pin	Functionality	
	MPSC0	Initial State
S0[0]	TxD0	3-state
S0[1]	RxD0	3-state
S0[2]	RTS0	3-state
S0[3]	CTS0	3-state
S0[4]	CD0	3-state
S0[5]	SCLK0	3-state
S0[6]	TSCLK0	3-state

Table 648: S1 Port Select Summary

Pin	Functionality	
	MPSC1	Initial State
S1[0]	TxD1	3-state
S1[1]	RxD1	3-state
S1[2]	RTS1	3-state
S1[3]	CTS1	3-state
S1[4]	CD1	3-state
S1[5]	SCLK1	3-state
S1[6]	TSCLK1	3-state

NOTE: Since the S0 port pins might act as input or output pins, depending on their configurable functionality, these pins wake up after reset as 3-state input pins, in order to prevent contentions on the board.

The S1 port has no multiplex control. It is always connected to MPSC_1

The above multiplexing allows the following comm ports combinations:

- 2 MII ports + 2 MPSC ports
- 3 RMII ports + 2 MPSC ports

19.3 Serial Port Configuration

Since the serial ports have different configurations and part of them wake-up after rest as tri-state pins, the following pull-ups and pull-downs are necessary:

- If not using port E0, pull down all of its inputs. To minimize the number of pull downs, configure the port to MII and pull down pins E0[1] and E0[14:6].
- When E0 is configured as two RMII ports, pins E0[7] and E0[14] are not connected and must be pulled down. If only one of the two RMIIs are used (Ethernet controller 0), pins E0[13] and E0[11:10] must not be connected and must be pulled down
- If not using port E1, pull down all of its inputs. To minimize the number of pull downs, configure the port to MII and pull down pins E1[1] and E1[14:6].
- When E1 is configured as an RMII port, pins E1[1], E1[7:6], E1[12:10], and E1[14] are not connected and must be pulled down.
- If not using port S0, configure the port to MPSC and pull down pins S0[1] and S0[6:3].
- If not using port S1, configure the port to MPSC and pull down pins S1[1] and S1[6:3].

19.4 MPP Interface Registers

Table 649: GPP Interface Register Map

Register	Offset	Page
MPP Control0	0xf000	page 485
MPP Control1	0xf004	page 488
MPP Control2	0xf008	page 491
MPP Control3	0xf00c	page 494
Serial Ports Multiplex	0xf010	page 496

Table 650: MPP Control0, Offset: 0xf000

Bits	Field Name	Function	Initial Value
3:0	MPPSel0	MPP0 Select 0x0 - GPP[0] 0x1 - DMAReq[0]* 0x2 - MGNT* 0x3 - EOT[7] 0x4 - TCEn[3] 0x5 - DBurst* 0x6 - Int[0]* 0x7 - GNT0[0]* 0x8 - GNT1[0]* 0x9 - WDNMI* 0xa to 0xe - Reserved 0xf - Debug[0]	0x0
7:4	MPPSel1	MPP1 Select 0x0 - GPP[1] 0x1 - DMAAck[0]* 0x2 - MREQ* 0x3 - EOT[7] 0x4 - TCTcnt[3] 0x5 - InitAct 0x6 - Int[1]* 0x7 - REQ0[0]* 0x8 - REQ1[0]* 0x9 - WDE* 0xa to 0xe - Reserved 0xf - Debug[1]	0x0

Table 650: MPP Control0, Offset: 0xf000 (Continued)

Bits	Field Name	Function	Initial Value
11:8	MPPSel2	MPP2 Select 0x0 - GPP[2] 0x1 - DMAReq[1]* 0x2 - PME0* 0x3 - EOT[6] 0x4 - TCEn[2] 0x5 - InitAct 0x6 - Int[2]* 0x7 - GNT0[1]* 0x8 - GNT1[1]* 0x9 - WDNMI* 0xa to 0xe - Reserved 0xf - Debug[2]	0x0
15:12	MPPSel3	MPP3 Select 0x0 - GPP[3] 0x1 - DMAAck[1]* 0x2 - PME1* 0x3 - EOT[6] 0x4 - TCTcnt[2] 0x5 - DBurst* 0x6 - Int[3]* 0x7 - REQ0[1]* 0x8 - REQ1[1]* 0x9 - WDE* 0xa to 0xe - Reserved 0xf - Debug[3]	0x0
19:16	MPPSel4	MPP4 Select 0x0 - GPP[4] 0x1 - DMAReq[2]* 0x2 - PME1* 0x3 - EOT[5] 0x4 - TCEn[1] 0x5 - InitAct 0x6 - Int[0]* 0x7 - GNT0[2]* 0x8 - GNT1[2]* 0x9 - BClkIn 0xa to 0xe - Reserved 0xf - Debug[4]	0x0

Table 650: MPP Control0, Offset: 0xf000 (Continued)

Bits	Field Name	Function	Initial Value
23:20	MPPSel5	MPP5 Select 0x0 - GPP[5] 0x1 - DMAAck[2]* 0x2 - PME0* 0x3 - EOT[5] 0x4 - TCTcnt[1] 0x5 - DBurst* 0x6 - Int[1]* 0x7 - REQ0[2]* 0x8 - REQ1[2]* 0x9 - BClkIn 0xa to 0xe - Reserved 0xf - Debug[5]	0x0
27:24	MPPSel6	MPP6 Select 0x0 - GPP[6] 0x1 - DMAReq[3]* 0x2 - MGNT* 0x3 - EOT[4] 0x4 - TCEn[0] 0x5 - DBurst* 0x6 - Int[2]* 0x7 - GNT0[3]* 0x8 - GNT1[3]* 0x9 - BClkOut0 0xa to 0xe - Reserved 0xf - Debug[6]	0x0
31:28	MPPSel7	MPP7 Select 0x0 - GPP[7] 0x1 - DMAAck[3]* 0x2 - MREQ* 0x3 - EOT[4] 0x4 - TCTcnt[0] 0x5 - InitAct 0x6 - Int[3]* 0x7 - REQ0[3]* 0x8 - REQ1[3]* 0x9 - BClkOut0 0xa to 0xe - Reserved 0xf - Debug[7]	0x0

Table 651: MPP Control1, Offset: 0xf004

Bits	Field Name	Function	Initial Value
3:0	MPPSel8	MPP8 Select 0x0 - GPP[8] 0x1 - DMAReq[4]* 0x2 - MGNT* 0x3 - EOT[3] 0x4 - TCEn[7] 0x5 - DBurst* 0x6 - Int[0]* 0x7 - GNT0[4]* 0x8 - GNT1[4]* 0x9 - WDNMI* 0xa to 0xe - Reserved 0xf - Debug[8]	0x0
7:4	MPPSel9	MPP9 Select 0x0 - GPP[9] 0x1 - DMAAck[4]* 0x2 - MREQ* 0x3 - EOT[3] 0x4 - TCTcnt[7] 0x5 - InitAct 0x6 - Int[1]* 0x7 - REQ0[4]* 0x8 - REQ1[4]* 0x9 - WDE* 0xa to 0xe - Reserved 0xf - Debug[9]	0x0
11:8	MPPSel10	MPP10 Select 0x0 - GPP[10] 0x1 - DMAReq[5]* 0x2 - PME1* 0x3 - EOT[2] 0x4 - TCEn[6] 0x5 - InitAct 0x6 - Int[2]* 0x7 - GNT0[5]* 0x8 - GNT1[5]* 0x9 - WDNMI* 0xa to 0xe - Reserved 0xf - Debug[10]	0x0

Table 651: MPP Control1, Offset: 0xf004 (Continued)

Bits	Field Name	Function	Initial Value
15:12	MPPSel11	MPP11 Select 0x0 - GPP[11] 0x1 - DMAAck[5]* 0x2 - PME0* 0x3 - EOT[2] 0x4 - TCTcnt[6] 0x5 - DBurst* 0x6 - Int[3]* 0x7 - REQ0[5]* 0x8 - REQ1[5]* 0x9 - WDE* 0xa to 0xe - Reserved 0xf - Debug[11]	0x0
19:16	MPPSel12	MPP12 Select 0x0 - GPP[12] 0x1 - DMAReq[6]* 0x2 - PME0* 0x3 - EOT[1] 0x4 - TCEn[5] 0x5 - InitAct 0x6 - Int[0]* 0x7 - GNT0[4]* 0x8 - GNT1[4]* 0x9 - BClkOut0 0xa to 0xe - Reserved 0xf - Debug[12]	0x0
23:20	MPPSel13	MPP13 Select 0x0 - GPP[13] 0x1 - DMAAck[6]* 0x2 - PME1* 0x3 - EOT[1] 0x4 - TCTcnt[5] 0x5 - DBurst* 0x6 - Int[1]* 0x7 - REQ0[4]* 0x8 - REQ1[4]* 0x9 - BClkOut0 0xa to 0xe - Reserved 0xf - Debug[13]	0x0

Table 651: MPP Control1, Offset: 0xf004 (Continued)

Bits	Field Name	Function	Initial Value
27:24	MPPSel14	MPP14 Select 0x0 - GPP[14] 0x1 - DMAReq[7]* 0x2 - MGNT* 0x3 - EOT[0] 0x4 - TCEn[4] 0x5 - DBurst* 0x6 - Int[2]* 0x7 - GNT0[3]* 0x8 - GNT1[3]* 0x9 - BClkIn 0xa to 0xe - Reserved 0xf - Debug[14]	0x0
31:28	MPPSel15	MPP15 Select 0x0 - GPP[15] 0x1 - DMAAck[7]* 0x2 - MREQ* 0x3 - EOT[0] 0x4 - TCTcnt[4] 0x5 - InitAct 0x6 - Int[3]* 0x7 - REQ0[3]* 0x8 - REQ1[3]* 0x9 - BClkIn 0xa to 0xe - Reserved 0xf - Debug[15]	0x0

Table 652: MPP Control2, Offset: 0xf008

Bits	Field Name	Function	Initial Value
3:0	MPPSel16	MPP16 Select 0x0 - GPP[16] 0x1 - DMAReq[0]* 0x2 - MGNT* 0x3 - EOT[7] 0x4 - TCEn[3] 0x5 - DBurst* 0x6 - Int[0]* 0x7 - GNT0[0]* 0x8 - GNT1[0]* 0x9 - WDNMI* 0xa to 0xe - Reserved 0xf - Debug[16]	0x0
7:4	MPPSel17	MPP1 Select 0x0 - GPP[17] 0x1 - DMAAck[0]* 0x2 - MREQ* 0x3 - EOT[7] 0x4 - TCTcnt[3] 0x5 - InitAct 0x6 - Int[1]* 0x7 - REQ0[0]* 0x8 - REQ1[0]* 0x9 - WDE* 0xa to 0xe - Reserved 0xf - Debug[17]	0x0
11:8	MPPSel18	MPP18 Select 0x0 - GPP[18] 0x1 - DMAReq[1]* 0x2 - PME0* 0x3 - EOT[6] 0x4 - TCEn[2] 0x5 - InitAct 0x6 - Int[2]* 0x7 - GNT0[1]* 0x8 - GNT1[1]* 0x9 - WDNMI* 0xa to 0xe - Reserved 0xf - Debug[18]	0x0

Table 652: MPP Control2, Offset: 0xf008 (Continued)

Bits	Field Name	Function	Initial Value
15:12	MPPSel19	MPP19 Select 0x0 - GPP[19] 0x1 - DMAAck[1]* 0x2 - PME1* 0x3 - EOT[6] 0x4 - TCTcnt[2] 0x5 - DBurst* 0x6 - Int[3]* 0x7 - REQ0[1]* 0x8 - REQ1[1]* 0x9 - WDE* 0xa to 0xe - Reserved 0xf - Debug[19]	0x0
19:16	MPPSel20	MPP20 Select 0x0 - GPP[20] 0x1 - DMAReq[2]* 0x2 - PME1* 0x3 - EOT[5] 0x4 - TCEn[1] 0x5 - InitAct 0x6 - Int[0]* 0x7 - GNT0[2]* 0x8 - GNT1[2]* 0x9 - BClkIn 0xa to 0xe - Reserved 0xf - Debug[20]	0x0
23:20	MPPSel21	MPP21 Select 0x0 - GPP[21] 0x1 - DMAAck[2]* 0x2 - PME0* 0x3 - EOT[5] 0x4 - TCTcnt[1] 0x5 - DBurst* 0x6 - Int[1]* 0x7 - REQ0[2]* 0x8 - REQ1[2]* 0x9 - BClkIn 0xa to 0xe - Reserved 0xf - Debug[21]	0x0

Table 652: MPP Control2, Offset: 0xf008 (Continued)

Bits	Field Name	Function	Initial Value
27:24	MPPSel22	MPP22 Select 0x0 - GPP[22] 0x1 - DMAReq[3]* 0x2 - MGNT* 0x3 - EOT[4] 0x4 - TCEn[0] 0x5 - DBurst* 0x6 - Int[2]* 0x7 - GNT0[3]* 0x8 - GNT1[3]* 0x9 - BClkOut0 0xa to 0xe - Reserved 0xf - Debug[22]	0x0
31:28	MPPSel23	MPP23 Select 0x0 - GPP[23] 0x1 - DMAAck[3]* 0x2 - MREQ* 0x3 - EOT[4] 0x4 - TCTcnt[0] 0x5 - InitAct 0x6 - Int[3]* 0x7 - REQ0[3]* 0x8 - REQ1[3]* 0x9 - BClkOut0 0xa to 0xe - Reserved 0xf - Debug[23]	0x0

Table 653: MPP Control3, Offset: 0xf00c

Bits	Field Name	Function	Initial Value
3:0	MPPSel24	MPP24 Select 0x0 - GPP[24] 0x1 - DMAReq[4]* 0x2 - MGNT* 0x3 - EOT[3] 0x4 - TCEn[7] 0x5 - DBurst* 0x6 - Int[0]* 0x7 - GNT0[4]* 0x8 - GNT1[4]* 0x9 - WDNMI* 0xa to 0xe - Reserved 0xf - Debug[24]	0x0
7:4	MPPSel25	MPP25 Select 0x0 - GPP[25] 0x1 - DMAAck[4]* 0x2 - MREQ* 0x3 - EOT[3] 0x4 - TCTcnt[7] 0x5 - InitAct 0x6 - Int[1]* 0x7 - REQ0[4]* 0x8 - REQ1[4]* 0x9 - WDE* 0xa to 0xe - Reserved 0xf - Debug[25]	0x0
11:8	MPPSel26	MPP26 Select 0x0 - GPP[26] 0x1 - DMAReq[5]* 0x2 - PME1* 0x3 - EOT[2] 0x4 - TCEn[6] 0x5 - InitAct 0x6 - Int[2]* 0x7 - GNT0[5]* 0x8 - GNT1[5]* 0x9 - WDNMI* 0xa to 0xe - Reserved 0xf - Debug[26]	0x0

Table 653: MPP Control3, Offset: 0xf00c (Continued)

Bits	Field Name	Function	Initial Value
15:12	MPPSel27	MPP27 Select 0x0 - GPP[27] 0x1 - DMAAck[5]* 0x2 - PME0* 0x3 - EOT[2] 0x4 - TCTcnt[6] 0x5 - DBurst* 0x6 - Int[3]* 0x7 - REQ0[5]* 0x8 - REQ1[5]* 0x9 - WDE* 0xa to 0xe - Reserved 0xf - Debug[27]	0x0
19:16	MPPSel28	MPP28 Select 0x0 - GPP[28] 0x1 - DMAReq[6]* 0x2 - PME0* 0x3 - EOT[1] 0x4 - TCEn[5] 0x5 - InitAct 0x6 - Int[0]* 0x7 - GNT0[4]* 0x8 - GNT1[4]* 0x9 - BClkOut0 0xa to 0xe - Reserved 0xf - Debug[28]	0x0
23:20	MPPSel29	MPP29 Select 0x0 - GPP[29] 0x1 - DMAAck[6]* 0x2 - PME1* 0x3 - EOT[1] 0x4 - TCTcnt[5] 0x5 - DBurst* 0x6 - Int[1]* 0x7 - REQ0[4]* 0x8 - REQ1[4]* 0x9 - BClkOut0 0xa to 0xe - Reserved 0xf - Debug[29]	0x0

Table 653: MPP Control3, Offset: 0xf00c (Continued)

Bits	Field Name	Function	Initial Value
27:24	MPPSel30	MPP30 Select 0x0 - GPP[30] 0x1 - DMAReq[7]* 0x2 - MGNT* 0x3 - EOT[0] 0x4 - TCEn[4] 0x5 - DBurst* 0x6 - Int[2]* 0x7 - GNT0[3]* 0x8 - GNT1[3]* 0x9 - BClkIn 0xa to 0xe - Reserved 0xf - Debug[30]	0x0
31:28	MPPSel31	MPP31 Select 0x0 - GPP[31] 0x1 - DMAAck[7]* 0x2 - MREQ* 0x3 - EOT[0] 0x4 - TCTcnt[4] 0x5 - InitAct 0x6 - Int[3]* 0x7 - REQ0[3]* 0x8 - REQ1[3]* 0x9 - BClkIn 0xa to 0xe - Reserved 0xf - Debug[31]	0x0

Table 654: Serial Ports Multiplex, Offset: 0xf010

Bits	Field Name	Function	Initial Value
3:0	E0Mux	Ethernet Port0 Multiplex Control 0x0 - High-z. 0x1 - Connected to Ethernet controller 0 MII interface. 0x2 - Connected to Ethernet controllers 0 and 2 RMII interfaces. 0x3 - 0xf - Reserved.	0x0
7:4	Reserved	Reserved	0x0

Table 654: Serial Ports Multiplex, Offset: 0xf010 (Continued)

Bits	Field Name	Function	Initial Value
11:8	S0Mux	Serial Port 0 Multiplex Control 0x0 - High-z 0x1 - Connected to MPSC 0 0x2 - Reserved 0x3 - 0xf - Reserved	0x0
15:12	S1Mux	Serial Port 1 Multiplex Control 0x0 - High-z 0x1 - 0xf - Connected to MPSC 1	0x0
31:16	Reserved	Reserved.	0x0

20. I²C Interface

The GT–64240A has full I²C support. It can act as master generating read/write requests and as a slave responding to read/write requests. It fully supports multiple I²C masters environment (clock synchronization, bus arbitration).

The I²C interface can be used for various applications. It can be used to control other I²C on board devices, to read DIMM SPD ROM and is also used for serial ROM initialization. For more details, see Section 24. "Reset Configuration" on page 522.

20.1 I²C Bus Operation

The I²C port consists of two open drain signals:

- SCL (Serial Clock)
- SDA (Serial address/data)

The I²C master starts a transaction by driving a start condition followed by a 7- or 10-bit slave address and a read/write bit indication. The target I²C slave responds with acknowledge.

In case of write access (R/$\overline{W}$ bit is '0'), following the acknowledge, the master drives 8-bit data and the slave responds with acknowledge. This write access (8-bit data followed by acknowledge) continues until the I²C master ends the transaction with stop condition.

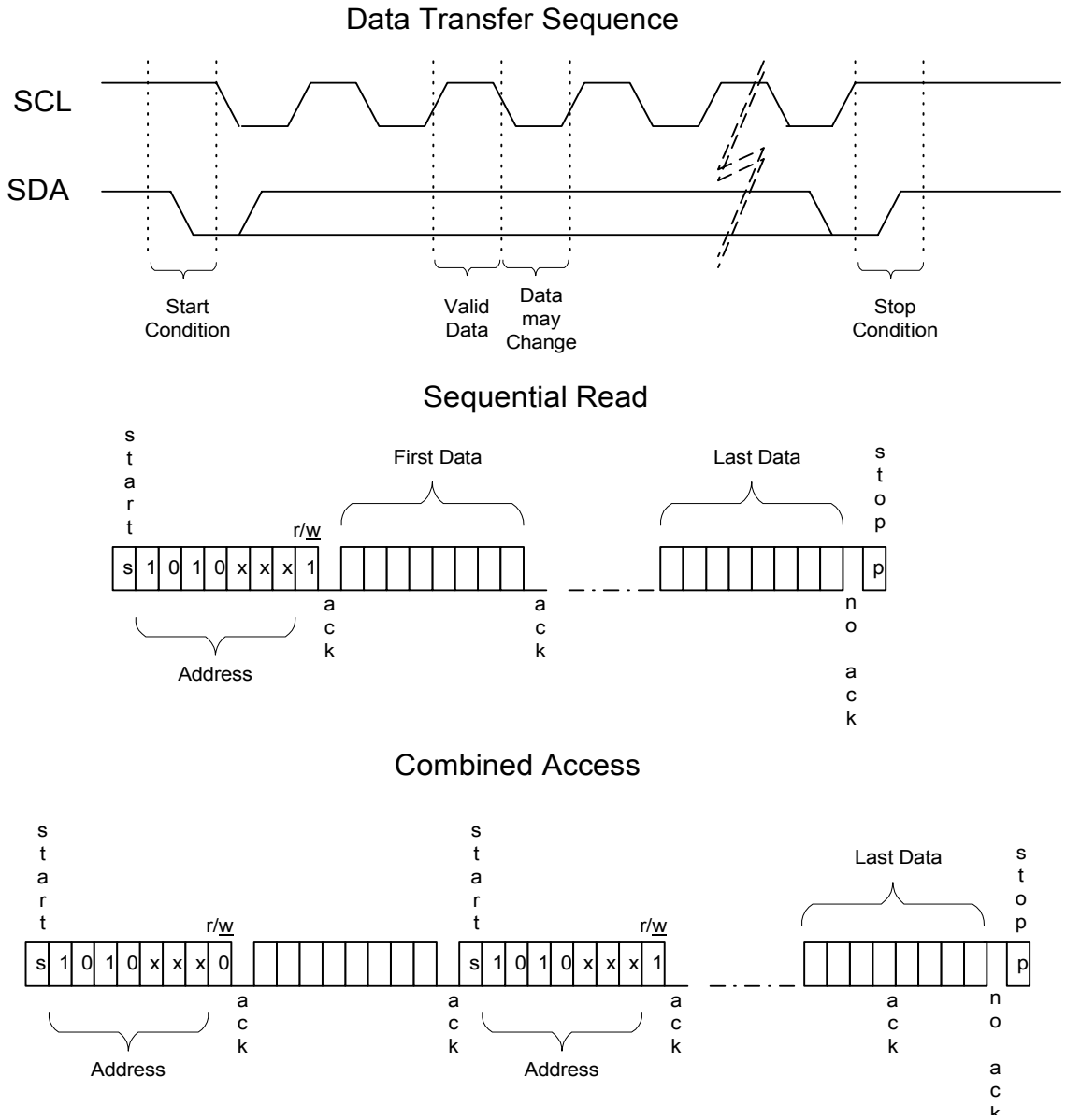
In case of read access, following the slave address acknowledge, the I²C slave drives 8-bit data and the master responds with acknowledge. This read access (8-bit data followed by acknowledge) continues until the I²C master ends the transaction by responding with no acknowledge to the last 8-bit data, followed by a stop condition.

A target slave that cannot drive valid read data right after it received the address, can insert "wait states" by forcing SCL low until it has valid data to drive on the SDA line.

A master is allowed to combine two transactions. After the last data transfer, it can drive a new start condition followed by new slave address, rather than drive stop condition. Combining transactions guarantees that the master does not loose arbitration to some other I²C master.

I²C examples are shown in Figure 78. For full I²C protocol description look in Philips Semiconductor I²C spec.

Figure 78: I^2C Examples



20.2 I²C Registers

The I²C interface master and slave activities are handled by simple CPU (or PCI) access to internal registers, plus interrupt interface. The following sections describe each of these registers.

20.2.1 I²C Slave Address registers

The I²C slave interface supports both 7-bit and 10-bit addressing. The slave address is programmed by the Slave Address register and Extended Slave Address register (see Table 658 and Table 659 on page 506).

When the I²C receives a 7-bit address after a start condition, it compares it against the value programed in the Slave Address register, and if it matches, it responds with acknowledge.

If the received 7 address bits are '11110xx', meaning that it is an 10-bit slave address, the I²C compares the received 10-bit address with the 10-bit value programed in the Slave Address and Extended Slave Address registers, and if it matches, it responds with acknowledge.

The I²C interface also support slave response to general call transactions. If GCE bit in the Slave Address register is set to '1', the I²C also responds to general call address (0x0).

20.2.2 I²C Data Register

The 8-bit Data register is used both in master and slave modes.

In master mode, the CPU must place the slave address or write data to be transmitted. In case of read access, it contains received data (need to be read by CPU).

In slave mode, the Data register contains data received from master on write access, or data to be transmitted (written by CPU) on read access.

NOTE: Data register MSB contains the first bit to be transmitted or being received.

20.2.3 I²C Control Register

This 8-bit register contains the following bits:

Table 655: I²C Control Register Bits

Bit	Function	Description
1:0	Reserved	Read only '0'.
2	Acknowledge Bit	When set to '1', the I²C drives an acknowledge bit on the bus in response to a received address (slave mode), or in response to a data received (read data in master mod, write data in slave mode). For a master to signal an I²C target a read of last data, the CPU must clear this bit (generating no acknowledge bit on the bus). For the slave to respond, this bit must always be set back to 1.

Table 655: I²C Control Register Bits (Continued)

Bit	Function	Description
3	Interrupt Flag	If any of the interrupt events occur, set to '1' by I²C hardware, see Section 20.2.4 "I2C Status Register" on page 501. If set to '1' and I²C interrupts are enabled through bit[7], an interrupt is asserted.
4	Stop Bit	When set to '1', the I²C master initiates a stop condition on the bus. The bit is set only. It is cleared by I²C hardware after a stop condition is driven on the bus.
5	Start Bit	When set to '1', the I²C master initiates a start condition on the bus, when the bus is free, or a repeated start condition, if the master already drives the bus. The bit is set only. It is cleared by I²C hardware after a start condition is driven on the bus.
6	I²C Enable	If set to '1', the I²C slave responds to calls to its slave address, and to general calls if enabled. If set to '0', SDA and SCL inputs are ignored. The I²C slave does not respond to any address on the bus.
7	Interrupt Enable	If set to '1', I²C interrupts are enabled. **NOTE:** It is highly recommended to use I²C interrupt to interface the I²C module, rather than using register polling method.

20.2.4 I²C Status Register

This 8-bit register contains the current status of the I²C interface. Bits[7:3] are the status code, bits[2:0] are Reserved (read only 0). Table 656 summarizes all possible status codes.

Table 656: I²C Status Codes

Code	Status
0x00	Bus error.
0x08	Start condition transmitted.
0x10	Repeated start condition transmitted.
0x18	Address + write bit transmitted, acknowledge received.
0x20	Address + write bit transmitted, acknowledge not received.
0x28	Master transmitted data byte, acknowledge received.
0x30	Master transmitted data byte, acknowledge not received.
0x38	Master lost arbitration during address or data transfer.
0x40	Address + read bit transmitted, acknowledge received.
0x48	Address + read bit transmitted, acknowledge not received.

Table 656: I^2C Status Codes (Continued)

Code	Status
0x50	Master received read data, acknowledge transmitted.
0x58	Master received read data, acknowledge not transmitted.
0x60	Slave received slave address, acknowledge transmitted.
0x68	Master lost arbitration during address transmit, address is targeted to the slave (write access), acknowledge transmitted.
0x70	General call received, acknowledge transmitted.
0x78	Master lost arbitration during address transmit, general call address received, acknowledge transmitted.
0x80	Slave received write data after receiving slave address, acknowledge transmitted.
0x88	Slave received write data after receiving slave address, acknowledge not transmitted.
0x90	Slave received write data after receiving general call, acknowledge transmitted.
0x98	Slave received write data after receiving general call, acknowledge not transmitted.
0xA0	Slave received stop or repeated start condition.
0xA8	Slave received address + read bit, acknowledge transmitted.
0xB0	Master lost arbitration during address transmit, address is targeted to the slave (read access), acknowledge transmitted.
0xB8	Slave transmitted read data, acknowledge received.
0xC0	Slave transmitted read data, acknowledge not received.
0xC8	Slave transmitted last read byte, acknowledge received.
0xD0	Second address + write bit transmitted, acknowledge received.
0xD8	Second address + write bit transmitted, acknowledge not received.
0xE0	Second address + read bit transmitted, acknowledge received.
0xE8	Second address + read bit transmitted, acknowledge not received.
0xF8	No relevant status. Interrupt flag is kept 0.

20.2.5 Baude Rate Register

I²C spec defines SCL frequency of 100KHz (400KHz in fast mode). The I²C module contains a clock divider that separates TClk to generate the SCL clock. Setting bits[6:0] of Baude Rate register defines SCL frequency as follows:

$$F_{SCL} = \frac{TClk}{10 \cdot (M+1) \cdot 2^{(N+1)}}$$

NOTE: Where M is the value represented by bits[6:3] and N the value represented by bits[2:0]. If for example *M=N=4* (which are the default values), running *TClk* at 10MHz results in *SCL* frequency of 62.5KHz.

20.3 I²C Master Operation

The CPU can initiate I²C master read and write transactions via I²C registers, as described in the following sections.

20.3.1 Master Write Access

Master write access consists of the following steps:

1. The CPU sets the I²C Control register's Start bit [5] to '1', see Table 661 on page 506. The I²C master then generates a start condition as soon as the bus is free, then sets an Interrupt flag, and then sets the Status register to 0x8.
2. The CPU writes 7-bit address plus write bit to the Data register, and then clears Interrupt flag for the I²C master interface to drive slave address on the bus. The target slave responds with acknowledge, causing Interrupt flag to be set, and status code of 0x18 be registered in the Status register. If the target I²C device has an 10-bit address, the CPU needs to write the remainder 8-bit address bits to the Data register, and then clears Interrupt flag for the master to drive this address on the bus. The target device responds with acknowledge, causing an Interrupt flag to be set, and status code of 0xD0 be registered in the Status register.
3. The CPU writes data byte to the Data register, and then clears Interrupt flag for the I²C master interface to drive the data on the bus. The target slave responds with acknowledge, causing Interrupt flag to be set, and status code of 0x28 be registered in the Status register. The CPU continues this loop of writing new data to the Data register and clear Interrupt flag, as long as it needs to transmit write data to the target.
4. After last data transmit, the CPU may terminate the transaction or restart a new transaction. To terminate the transaction, the CPU sets the Control register's Stop bit and then clears the Interrupt flag, causing I²C master to generate a stop condition on the bus, and go back to idle state. To restart a new transaction, the CPU sets the I²C Control register's Start bit and clears the Interrupt flag, causing I²C master to generate a new start condition.

NOTE: This sequence describes a normal operation. There are also abnormal cases, such as a slave not responding with acknowledge, or arbitration loss. Each of these cases is reported in the Status register and needs to be handled by CPU.

20.3.2 Master Read Access

1. Generating start condition, exactly the same as in the case of write access, see Section 20.3.1 “Master Write Access” on page 503).
2. Drive 7- or 10-bit slave address, exactly the same as in the case of write access, with the exception that the status code after 1st address byte transmit is 0x40, and after 2nd address byte transmit (in case of 10-bit address) is 0xE0.
3. Read data being received from target device is placed in the data register and acknowledge is driven on the bus. Also interrupt flag is set, and status code of 0x50 is registered in the Status register. The CPU reads data from Data register and clears the Interrupt flag to continue receiving next read data byte. This look is continued as long as the CPU wishes to read data from the target device.
4. To terminate, the read access needs to respond with no acknowledge to the last data. It then generates a stop condition or generates a new start condition to restart a new transaction. With last data, the CPU clears the I^2C Control register’s Acknowledge bit (when clearing the Interrupt bit), causing the I^2C master interface to respond with no acknowledge to last received read data. In this case, the Interrupt flag is set with status code of 0x58. Now, the CPU can issue a stop condition or a new start condition.

NOTE: The above sequence describes a normal operation. There are also abnormal cases, such as the slave not responding with acknowledge, or arbitration loss. Each of these cases is reported in the Status register and needs to be handled by CPU.

20.4 I^2C Slave Operation

The I^2C slave interface can respond to a read access, driving read data back to the master that initiated the transaction, or respond to write access, receiving write data from the master.

The two cases are described in the following sections.

20.4.1 Slave Read Access

Upon detecting a new address driven on the bus with read bit indication, the I^2C slave interface compares the address against the address programmed in the Slave Address register. If it matches, the slave responds with acknowledge. It also sets the Interrupt flag, and sets status code to 0xA8.

NOTE: If the I^2C slave address is 10-bit, the Interrupt flag is set and status code changes only after receiving and identify address match also on the 2nd address byte).

The CPU now needs to write new read data to the Data register and clears the Interrupt flag, causing I^2C slave interface to drive the data on the bus. The master responds with acknowledge causing an Interrupt flag to be set, and status code of 0xB8 to be registered in the Status register.

If the master does not respond with acknowledge, the Interrupt flag is set, status code 0f 0xC0 is registered, and I^2C slave interface returns back to idle state.

If the master generates a stop condition after driving an acknowledge bit, the I^2C slave interface returns back to idle state.

20.4.2 Slave Write Access

Upon detecting a new address driven on the bus with read bit indication, the I^{2}C slave interface compares the address against the address programed in the Slave Address register and, if it matches, responds with acknowledge. It also sets an Interrupt flag, and sets status code to 0x60 (0x70 in case of general call address, if general call is enabled).

Following each write byte received, the I^{2}C slave interface responds with acknowledge, sets an Interrupt flag, and sets status code to 0x80 (0x90 in case of general call access). The CPU then reads the received data from Data register and clears Interrupt flag, to allow transfer to continue.

If a stop condition or a start condition of a new access is detected after driving the acknowledge bit, an Interrupt flag is set and a status code of 0xA0 is registered.

20.5 I^{2}C Interface Registers

Table 657: I^{2}C Interface Register Map

Register	Offset	Page
I^{2}C Slave Address	0xc000	page 505
I^{2}C Extended Slave address	0xc010	page 506
I^{2}C Data	0xc004	page 506
I^{2}C Control	0xc008	page 506
I^{2}C Status/Baude Rate	0xc00c	page 507
I^{2}C Soft Reset	0xc01c	page 507

Table 658: I^{2}C Slave Address, Offset: 0xc000

Bits	Field Name	Function	Initial Value
0	GCE	General Call Enable If set to '1', the I^{2}C slave interface responds to general call accesses.	0x0
7:1	SAddr	Slave address For a 7-bit slave address, bits[7:1] are the slave address. For a 10-bit address, SAddr[7:3] must be set to '11110' and SAddr[2:1] stands for the two MSB (bits[9:8]) of the 10-bit address.	0x0
31:8	Reserved	Reserved.	0x0

Table 659: I²C Extended Slave Address, Offset: 0xc010

Bits	Field Name	Function	Initial Value
7:0	SAddr	Bits[7:0] of the 10-bit slave address.	0x0
31:8	Reserved	Reserved.	0x0

Table 660: I²C Data, Offset: 0xc004

Bits	Field Name	Function	Initial Value
7:0	Data	Data/Address byte to be transmitted by the I²C master or slave, or data byte received.	0x0
31:8	Reserved	Reserved.	0x0

Table 661: I²C Control, Offset: 0xc008

Bits	Field Name	Function	Initial Value
1:0	Reserved	Read only.	0x0
2	ACK	Acknowledge When set to '1', the I²C master drives the acknowledge bit in response to received read data and to the I²C slave in response to received address or write data.	0x0
3	IFlg	Interrupt Flag If any of the status codes other than 0xf8 are set, the I²C hardware sets the bit to '1'. The bit is cleared by a CPU write of '0'.	0x0
4	Stop	Stop When set to '1', the GT–64240A drives a stop condition on the bus. It is cleared by the I²C hardware.	0x0
5	Start	Start When set to '1', the GT–64240A drives a start condition as soon as the bus is free. It is cleared by the I²C hardware.	0x0
6	I²CEn	If set to '0', the SDA and SCL inputs are not sampled and the I²C slave interface does not respond to any address on the bus.	0x0
7	IntEn	Interrupt Enable When set to '1', the interrupt is generated each time the interrupt flag is set.	0x0
31:8	Reserved	Reserved.	0x0

Table 662: I²C Status, Offset: 0xc00c[1]

Bits	Field Name	Function	Initial Value
2:0	Reserved	Read only	0x0
7:3	Stat	I²C Status See exact status code in the I²C section. Read only.	0x1f
31:8	Reserved	Reserved.	0x0

1. Status and Baude Rate registers share the same offset. When being read, this register functions as Status register. When written, it acts as Baude Rate register.

Table 663: I²C Baude Rate, Offset: 0xc00c[1]

Bits	Field Name	Function	Initial Value
2:0	N	See exact frequency calculation in the I²C section. Write only.	0x4
6:3	M	See exact frequency calculation in the I²C section. Write only.	0x4
31:7	Reserved	Reserved.	0x0

1. Status and Baude Rate registers share the same offset. When being read, this register functions as Status register. When written, it acts as Baude Rate register.

Table 664: I²C Soft Reset, Offset: 0xc01c

Bits	Field Name	Function	Initial Value
31:0	Rst	Write Only Write to this register resets the I²C logic and sets all I²C registers to their reset values.	0x0

21. INTERRUPT CONTROLLER

The GT–64240A includes an interrupt controller that routes internal interrupt requests (and optionally external interrupt requests) to both the CPU and the PCI bus.

The GT–64240A can drive up to seven interrupt pins. There are two open-drain interrupt pins dedicated for the two PCI interfaces, one dedicated CPU interrupt, and up to four additional CPU interrupts multiplexed on MPP pins.

NOTE: The four CPU interrupts multiplexed can be used to drive R5000/R7000 multiple interrupt inputs.

All seven interrupts driven by the GT–64240A are level sensitive. The interrupt is kept active as long there is at least one non-masked cause bit set in the Interrupt Cause register.

21.1 Interrupt Cause and Mask Registers

The GT–64240A handles interrupts in two stages. It includes a main cause register that summarizes the interrupts generated by each unit, and specific unit cause registers, that distinguish between each specific interrupt event.

21.1.1 Interrupts Cause Registers

The GT–64240A units cause registers are:

Table 665: Interrupts Cause Registers

• CPU Error Cause register	• PCI_1 Outbound Cause register	• Ethernet2 Cause register
• SDRAM Error Address register	• IDMAs 0-3 Cause register	• MPSC0 Cause register
• Device Interface Cause register	• IDMAs 4-7 Cause register	• MPSC1 Cause register
• PCI_1 Cause register	• Timers 0-3 Cause register	• BRG Cause register
• PCI_0 Cause register	• Timers 4-7 Cause register	• GPP Cause register
• PCI_0 Inbound Cause register	• Ethernet0 Cause register	• I^2C Cause register
• PCI_0 Outbound Cause register	• Ethernet1 Cause register	• SDMA Cause register
• PCI_1 Inbound Cause register		

Each unit has its own cause and mask registers. Once an interrupt event occurs, its corresponding bit in the cause register is set to ‘1’. If the interrupt is not masked, it is also marked in the main interrupt cause register.

NOTE: The unit local mask register has no effect on the setting of interrupt bits in the Local Cause register. It only effects the setting of the interrupt bit in the Main Interrupt Cause register.

For example, if the CPU attempts to write to a write protected region, the WrProt bit in the CPU Cause register is set to '1'. If the interrupt is not masked by CPU Mask register, the CPU bit in the Main Interrupt Cause register is also set. The interrupt handler first reads the Main Cause register and identifies that some CPU error event occurred. Then, it reads the CPU Cause register and identifies the exact cause for the interrupt.

NOTE: The Main Interrupt Cause register bits are Read Only. To clear an interrupt cause, the software needs to clear (write 0) the active bit(s) in the local cause register.

21.1.2 Interrupts Mask Registers

There are seven mask registers corresponding to the seven interrupt pins. Setting these registers allows reporting different interrupt events on different interrupt pins. If a bit in the mask register is set to '1', the corresponding interrupt event is enabled. The setting of the mask bits has no affect on the value registered in the Interrupt Cause register, it only affects the assertion of the interrupt pin.

The Main Interrupt Cause register is built of two 32-bit registers - Low and High. The main three interrupts - PCI_0 interrupt, PCI_1 interrupt and CPU interrupt - also have two 32-bit mask registers, each. However, the additional four optional interrupt pins have a single 32-bit mask register, each. The user can select whether the interrupt is triggered by Low or High Interrupt Cause register bits, depending on the setting of bit[31] of the mask register.

NOTE: The Main Cause and Mask registers are physically placed in different units than the Local Cause and Mask registers. This means that one cannot guarantee write ordering between Main Mask registers and Local Cause registers. If such ordering is required (for example, clear cause bit in the local cause register, and then cancel mask in the main mask register), the first write must be followed with a read (that guarantees that the register programing is done) and only then programs the second register.

21.1.3 Selected Cause Registers

If any of the three main interrupt pins are asserted, for the interrupt handler to identify the exact interrupt, it must read both the Low and High Interrupt Cause registers. To minimize this procedure to a single read, the GT–64240A contains three Selected Cause registers. The interrupt handler can read these registers rather than the cause registers.

A Select Cause register is a shadow register of the Low or High Cause register, depending whether the active interrupt bit is in the Low or High Cause register. Bit[30] of the Select Cause register, indicates which of Low or High Cause registers are currently represented by the Select Cause register.

21.1.4 Error Report Registers

The GT–64240A also implements on each of its interfaces, Error Report registers that latch the address (and sometimes data, command, byte enables) upon interrupt assertion caused by an error condition (such as parity error or address miss match). These registers can be helpful for the interrupt handler to locate the exact failure.

NOTE: For full details, see the registers section of each interface.

21.2 Interrupt Controller Registers

Table 666: Interrupt Controller Register Map

Register	Offset	Page
Main Interrupt Cause (Low)	0xc18	page 510
Main Interrupt Cause (High)	0xc68	page 512
CPU Interrupt Mask (Low)	0xc1c	page 513
CPU Interrupt Mask (High)	0xc6c	page 515
CPU Select Cause	0xc70	page 515
PCI_0 Interrupt Mask (Low)	0xc24	page 515
PCI_0 Interrupt Mask (High)	0xc64	page 515
PCI_0 Select Cause	0xc74	page 516
PCI_1 Interrupt Mask (Low)	0xca4	page 516
PCI_1 Interrupt Mask (High)	0xce4	page 516
PCI_1 Select Cause	0xcf4	page 516
CPU Int[0]* Mask	0xe60	page 516
CPU Int[1]* Mask	0xe64	page 516
CPU Int[2]* Mask	0xe68	page 516
CPU Int[3]* Mask	0xe6c	page 517

Table 667: Main Interrupt Cause (Low), Offset: 0xc18[1]

Bits	Field Name	Function	Initial Value
0	Sum	Logical OR of Low and High Cause registers bits	0x0
1	Dev	Device Interface Interrupt	0x0
2	DMA[2]	DMA Interrupt (error condition)	0x0
3	CPU	CPU Interface Interrupt	0x0
4	IDMA0_1	DMA completion of IDMA Channels 0-1 Interrupt.	0x0
5	IDMA2_3	DMA completion of IDMA Channels 2-3 Interrupt.	0x0
6	IDMA4_5	DMA completion of IDMA Channels 4-5 Interrupt.	0x0
7	IDMA6_7	DMA completion of IDMA Channels 6-7 Interrupt.	0x0
8	Timer0_1	Timers 0-1 Interrupt	0x0
9	Timer2_3	Timers 2-3 Interrupt	0x0

Table 667: Main Interrupt Cause (Low), Offset: 0xc18[1] (Continued)

Bits	Field Name	Function	Initial Value
10	Timer4_5	Timers 4-5 Interrupt.	0x0
11	Timer6_7	Timers 6-7 Interrupt.	0x0
12	PCI0_0	PCI_0 Interrupt **NOTE:** Summary of the PCI_0 Cause register's bits[7:0].	0x0
13	PCI0_1	PCI_0 Interrupt **NOTE:** Summary of the PCI_0 Cause register's bits[15:8].	0x0
14	PCI0_2	PCI_0 Interrupt **NOTE:** Summary of the PCI_0 Cause register's bits[23:16].	0x0
15	PCI0_3	PCI_0 Interrupt **NOTE:** Summary of the PCI_0 Cause register's bits[31:24].	0x0
16	PCI1_0	PCI_1 Interrupt **NOTE:** Summary of the PCI_1 Cause register's bits[7:0].	0x0
17	ECC	ECC Error Interrupt	0x0
18	PCI1_1	PCI_1 Interrupt **NOTE:** Summary of the PCI_1 Cause register's bits[15:8].	0x0
19	PCI1_2	PCI_1 Interrupt **NOTE:** Summary of the PCI_1 Cause register's bits[23:16].	0x0
20	PCI1_3	PCI_1 Interrupt **NOTE:** Summary of the PCI_1 Cause register's bits[31:24].	0x0
21	PCI0OutL	PCI_0 Outbound Interrupt Summary **NOTE:** Summary of the PCI_0 Outbound Cause register's bits[15:0].	0x0
22	PCI0OutH	PCI_0 Outbound Interrupt Summary **NOTE:** Summary of the PCI_0 Outbound Cause register's bits[31:16].	0x0
23	PCI1OutL	PCI_1 Outbound Interrupt Summary **NOTE:** Summary of the PCI_1 Outbound Cause register's bits[15:0].	0x0

Table 667: Main Interrupt Cause (Low), Offset: 0xc18[1] (Continued)

Bits	Field Name	Function	Initial Value
24	PCI1OutH	PCI_1 Outbound Interrupt Summary **NOTE:** Summary of the PCI_1 Outbound Cause register's bits[31:16].	0x0
25	Reserved	Reserved.	0x0
26	PCI0InL	PCI_0 Inbound Interrupt **NOTE:** Summary of the PCI_0 Inbound Cause register's bits[15:0].	0x0
27	PCI0InH	PCI_0 Inbound Interrupt **NOTE:** Summary of the PCI_0 Inbound Cause register's bits[31:16].	0x0
28	PCI1InL	PCI_1 Inbound Interrupt **NOTE:** Summary of the PCI_1 Inbound Cause register's bits[15:0].	0x0
29	PCI1InH	PCI_1 Inbound Interrupt **NOTE:** Summary of the PCI_1 Inbound Cause register's bits[31:16].	0x0
31:30	Reserved	Reserved.	0x0

1. All bits are read only. To clear an interrupt, the software must access the Local Interrupt Cause registers.

2. Set upon any DMA channel address decoding failure, access protection violation, or descriptor ownership violation.

Table 668: Main Interrupt Cause (High), Offset: 0xc68

Bits	Field Name	Function	Initial Value
0	Eth0	Ethernet Controller 0 Interrupt	0x0
1	Eth1	Ethernet Controller 1 Interrupt	0x0
2	Eth2	Ethernet Controller 2 Interrupt	0x0
3	Reserved	Reserved.	0x0
4	SDMA	MPSC SDMA Interrupt	0x0
5	I^2C	I^2C Interrupt	0x0
6	Reserved	Reserved.	0x0
7	BRG	Baude Rate Generator Interrupt	0x0
8	MPSC0	MPSC 0 Interrupt	0x0
9	Reserved	Reserved.	0x0

Table 668: Main Interrupt Cause (High), Offset: 0xc68 (Continued)

Bits	Field Name	Function	Initial Value
10	MPSC1	MPSC 1 Interrupt	0x0
11	Comm	Comm Unit Interrupt	0x0
23:12	Reserved	Reserved.	0x0
24	GPP7_0	GPP[7:0] Interrupt	0x0
25	GPP15_8	GPP[15:8] Interrupt	0x0
26	GPP23_16	GPP[23:16] Interrupt	0x0
27	GPP31_24	GPP[31:24] Interrupt	0x0
31:28	Reserved	Reserved.	0x0

Table 669: CPU Interrupt Mask (Low), Offset: 0xc1c

Bits	Field Name	Function	Initial Value
0	Reserved	Reserved.	0x0
1	Dev	If set to ‘1’, Dev interrupt is enabled.	0x0
2	DMA	If set to ‘1’, DMA interrupt is enabled.	0x0
3	CPU	If set to ‘1’, CPI interrupt is enabled.	0x0
4	IDMA0_1	If set to ‘1’, IDMA0_1 interrupt is enabled.	0x0
5	IDMA2_3	If set to ‘1’, IDMA2_3 interrupt is enabled.	0x0
6	IDMA4_5	If set to ‘1’, IDMA4_5 interrupt is enabled.	0x0
7	IDMA6_7	If set to ‘1’, IDMA6_7 interrupt is enabled.	0x0
8	Timer0_1	If set to ‘1’, Timer0_1 interrupt is enabled.	0x0
9	Timer2_3	If set to ‘1’, Timer2_3 interrupt is enabled.	0x0
10	Timer4_5	If set to ‘1’, Timer4_5 interrupt is enabled.	0x0
11	Timer6_7	If set to ‘1’, Timer6_7 interrupt is enabled.	0x0
12	PCI0_0	If set to ‘1’, PCI0_0 interrupt is enabled.	0x0
13	PCI0_1	If set to ‘1’, PCI0_1 interrupt is enabled.	0x0
14	PCI0_2	If set to ‘1’, PCI0_2 interrupt is enabled.	0x0
15	PCI0_3	If set to ‘1’, PCI0_3 interrupt is enabled.	0x0
16	PCI1_0	If set to ‘1’, PCI1_0 interrupt is enabled.	0x0
17	ECC	If set to ‘1’, ECC interrupt is enabled.	0x0
18	PCI1_1	If set to ‘1’, PCI1_1 interrupt is enabled.	0x0

Table 669: CPU Interrupt Mask (Low), Offset: 0xc1c (Continued)

Bits	Field Name	Function	Initial Value
19	PCI1_2	If set to '1', PCI1_2 interrupt is enabled.	0x0
20	PCI1_3	If set to '1', PCI1_3 interrupt is enabled.	0x0
21	PCI0OutL	If set to '1', PCI0OutL interrupt is enabled.	0x0
22	PCI0OutH	If set to '1', PCI0OutH interrupt is enabled.	0x0
23	PCI1OutL	If set to '1', PCI1OutL interrupt is enabled.	0x0
24	PCI1OutH	If set to '1', PCI1OutH interrupt is enabled.	0x0
25	Reserved	Reserved.	0x0
26	PCI0InL	If set to '1', PCI0InL interrupt is enabled.	0x0
27	PCI0InH	If set to '1', PCI0InH interrupt is enabled.	0x0
28	PCI1InL	If set to '1', PCI1InL interrupt is enabled.	0x0
29	PCI1InH	If set to '1', PCI1InH interrupt is enabled.	0x0
31:30	Reserved	Reserved.	0x0

Table 670: CPU Interrupt Mask (High), Offset: 0xc6c

Bits	Field Name	Function	Initial Value
0	Eth0	If set to '1', Eth0 interrupt is enabled.	0x0
1	Eth1	If set to '1', Eth1 interrupt is enabled.	0x0
2	Eth2	If set to '1', Eth2 interrupt is enabled.	0x0
3	Reserved	Reserved.	0x0
4	SDMA	If set to '1', SDMA interrupt is enabled.	0x0
5	I^2C	If set to '1', I^2C interrupt is enabled.	0x0
6	Reserved		0x0
7	BRG	If set to '1', BRG interrupt is enabled.	0x0
8	MPSC0	If set to '1', MPSC0 interrupt is enabled.	0x0
9	Reserved	Reserved.	0x0
10	MPSC1	If set to '1', MPSC1 interrupt is enabled.	0x0
11	Comm	If set to '1', Comm interrupt is enabled.	0x0
23:12	Reserved	Reserved.	0x0
24	GPP7_0	If set to '1', GPP7_0 interrupt is enabled.	0x0
25	GPP15_8	If set to '1', GPP15_8 interrupt is enabled.	0x0

Table 670: CPU Interrupt Mask (High), Offset: 0xc6c (Continued)

Bits	Field Name	Function	Initial Value
26	GPP23_16	If set to '1', GPP23_16 interrupt is enabled.	0x0
27	GPP31_24	If set to '1', GPP31_24 interrupt is enabled.	0x0
31:28	Reserved	Reserved.	0x0

Table 671: CPU Select Cause, Offset: 0xc70[1]

Bits	Field Name	Function	Initial Value
29:0	Cause	A shadow register of the Low or High Interrupt Cause registers. If any of the High Interrupt Cause register non-masked interrupts are set, and no non-masked interrupt bit of the Low Interrupt Cause register is set, this register contains a copy of the High Interrupt Cause register. In any other case, it contains a copy of the Low Interrupt Cause register.	0x0
30	Sel	Select 0 - Bits[29:0] are a copy of the Low Interrupt Cause register 1 - Bits[29:0] are a copy of the High Interrupt Cause register	0x0
31	Stat	Status 0 - There are no active non-masked interrupts in both Low and High Interrupt Cause registers. 1 - There are active non-masked interrupts in both Low and High Interrupt Cause registers.	0x0

1. Read Only register.

Table 672: PCI_0 Interrupt Mask (Low), Offset: 0xc24

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Interrupt Mask (Low).	0x0

Table 673: PCI_0 Interrupt Mask (High), Offset: 0xc64

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Interrupt Mask (High).	0x0

Table 674: PCI_0 Select Cause, Offset: 0xc74

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Select Cause.	0x0

Table 675: PCI_1 Interrupt Mask (Low), Offset: 0xca4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Interrupt Mask.	0x0

Table 676: PCI_1 Interrupt Mask (High), Offset: 0xce4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Interrupt Mask (High).	0x0

Table 677: PCI_1 Select Cause, Offset: 0xcf4

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Select Cause.	0x0

Table 678: CPU Int[0]* Mask, Offset: 0xe60

Bits	Field Name	Function	Initial Value
30:0	Various	Same as Low or High CPU Interrupt Mask.	0x0
31	Sel	Mask Select 0 - Mask Low Interrupt Cause register bits. 1 - Mask high Interrupt Cause register bits.	0x0

Table 679: CPU Int[1]* Mask, Offset: 0xe64

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Int[0]* Mask.	0x0

Table 680: CPU Int[2]* Mask, Offset: 0xe68

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Int[1]* Mask.	0x0

Table 681: CPU Int[3]* Mask, Offset: 0xe6c

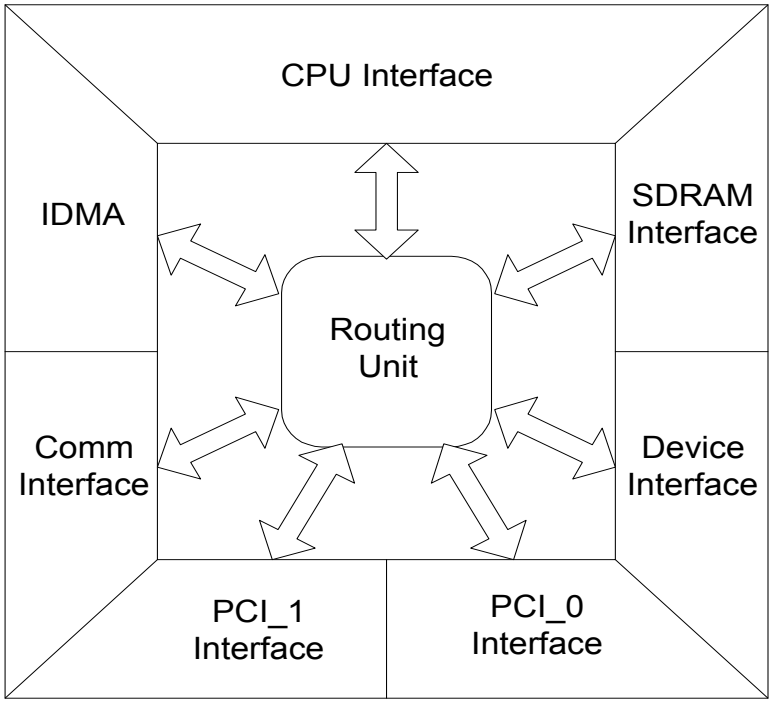
Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Int[2]* Mask.	0x0

22. INTERNAL ARBITRATION CONTROL

The GT–64240A internal architecture is based on a 64-bit data path connecting between the different interfaces. This internal architecture allows concurrent data transfers between different interfaces (for example, CPU read from SDRAM, PCI_0 read from device and IDMA write to PCI_1 at the same time), as well as transaction pipelining (issue multiple transactions in parallel between the same source and destination).

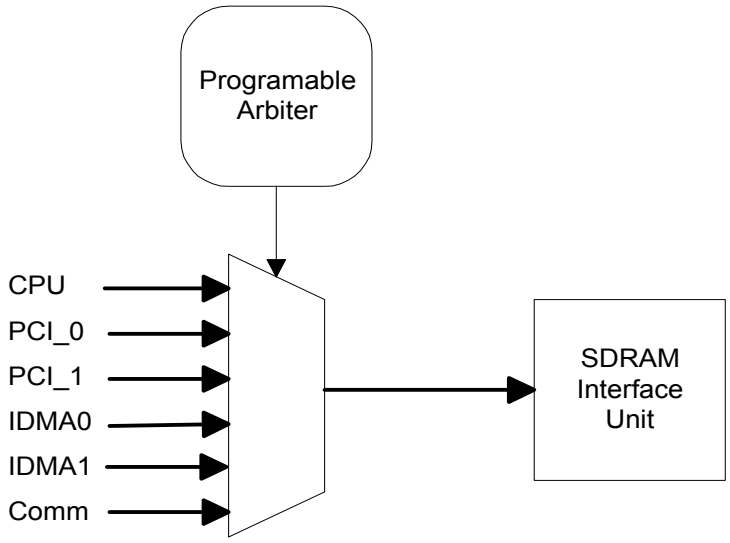
Figure 79 shows how the data path routing is controlled via a central routing unit (also called Crossbar).

Figure 79: GT–64240A Inter Units Connect



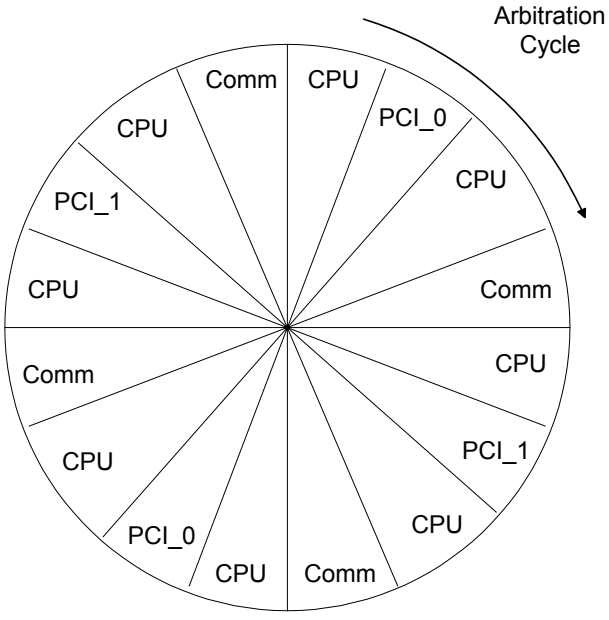
Sometimes conflicts may occur over resources. For example, if the CPU, PCI_0, PCI_1, and IDMA request access to SDRAM simultaneously, these requests cannot be served at the same time. The central routing unit contains programmable arbitration mechanisms to optimize device performance, according to the system requirements, as shown in Figure 80.

Figure 80: SDRAM Interface Arbitration



Each arbiter is a user defined round-robin arbiter (called a “pizza arbiter”). Figure 81 shows an example of the Device interface arbiter.

Figure 81: Configurable Weights Arbiter



The user can define each of the 16 slices of this “pizza arbiter”. The arbiter is working on a transaction basis. Each transaction can vary from one up to 16 64-bit word transfers. In the above example, the transactions targeted to the specific unit are split up as follows:

- 50% for CPU transactions.
- 25% for communication unit transactions.
- 12.5% for the PCI_0 and PCI_1 transactions.

This “pizza” configuration also allows the user to guarantee minimum latency. Even if the CPU does not require 50% allocation, the above configuration guarantees that in the worst case, the CPU request needs to wait for one transaction of another unit before being served.

At each clock cycle, the Crossbar arbiter samples all requests and gives the bus to the next agent according to the “pizza”. It is parked on the last access.

The exact registers settings can be found in the CPU, PCI, SDRAM, Device, IDMA and Comm units registers sections.

An arbiter slice can also be marked as NULL. If marked as NULL, the arbiter works as if the NULL slice does not exist. For example, if only two requests are used, and they need to get the same bandwidth, the user can specify first slice per one request, second slice per the other request, and all the rest slices as NULL. This is equivalent to specifying half of the slices for one request and the other half for the other request.

NOTE: Once a unit is removed from an interface’s “pizza” Arbiter Control register, this unit has no access to this interface. If for example, the comm unit is removed from the DRAM interface “pizza” arbiter, the comm unit no longer accesses the DRAM. If it attempts to access the DRAM, the unit will get stuck.

23. Reset Pins

The GT–64240A supports three reset pins:

- SysRst* which is the main reset pin.
- RST0* and RST1* pins which are the PCI interfaces reset pins.

Separating SysRst* from the PCI reset pins is typically required in Hot Swap configurations, where you want the CPU to boot and start to initialize the board before the PCI slot reset signal is deasserted. Separating the two PCI interface resets is required for PCI compliance reasons (since the two PCI busses are independent, each one must have its own reset).

SysRst* is the main GT–64240A reset pin. When asserted, all GT–64240A logics are in reset state and all outputs are floated, except for DRAM address and control outputs (see Section 5.10 “SDRAM Initialization” on page 116).

NOTE: All resets pins are asynchronous inputs and synchronized internally. The internal synchronized reset is delayed by three clock cycles in respect to the external reset pin, causing the GT–64240A output pins to remain floated for three cycles after reset deassertion.

The PCI reset pins are independent. The PCI interface is kept in its reset state as long as its corresponding reset pin is asserted. On reset deassertion, all PCI configuration registers are set to their initial values as specified in the PCI spec.

NOTE: The PCI reset pins must never be deasserted prior to SysRst* deassertion.

Since the GT–64240A supports SysRst* deassertion prior to the PCI reset pins deassertion, the CPU software might need a hook to recognize when the PCI bus is alive. Use the PCI Mode register’s PRst bit [31] of each PCI interface for this purpose, see Table 234 on page 209. Upon PCI reset deassertion, the bit is set to ‘1’.

24. RESET CONFIGURATION

The GT–64240A must acquire some knowledge about the system before it is configured by the software. Special modes of operation are sampled on RESET to enable the GT–64240A to function as required.

The GT–64240A supports two methods of reset configuration:

- Pins sampled on SysRst* deassertion (requires pins pulled up/down to Vcc/GND).
- Serial ROM initialization.

24.1 Pins Sample Configuration

If not using serial ROM initialization, the following configuration pins are sampled during Rst* assertion. These signals must be kept pulled up or down until Rst* deassertion (zero Hold time in respect to Rst* deassertion).

Table 682: Reset Configuration

Pin	Configuration Function
AD[0]	Serial ROM initialization
0- 1-	Not supported Supported **NOTE:** If Serial ROM initialization is enabled, the additional required strapping options are AD[1] Serial ROM Byte Offset Width, AD[3:2] Serial ROM Address, AD[4] CPU endianess, AD[28:30] PLL Settings, and AD[31] CPU Interface Voltage. See Section 24.2 “Serial ROM Initialization” on page 524.
AD[1]	Serial ROM Byte Offset Width
0- 1-	Up to 8-bit address Address wider than 8-bit
AD[3:2]	Serial ROM Address[1:0]
00- 01- 10- 11-	Rom address is 1010000 Rom address is 1010001 Rom address is 1010010 Rom address is 1010011
AD[4]	CPU Data Endianess
0- 1-	Big endian Little endian
AD[5]	CPU Interface Clock
0- 1-	CPU interface is running with SysClk, asynchronously to TClk CPU interface is running with TClk
AD[7:6]	CPU Bus Configuration
	Must be strapped to 10.

Table 682: Reset Configuration (Continued)

Pin	Configuration Function
AD[8]	Reserved
	NOTE: Must pull down.
AD[9]	Multiple GT–64240A Support
0- 1-	Not supported Supported
AD[11:10]	Multi-GT–64240A Address ID
00- 01- 10- 11-	GT responds to CPU address bits[26,25]='00' GT responds to CPU address bits[26,25]='01' GT responds to CPU address bits[26,25]='10' GT responds to CPU address bits[26,25]='11'
AD[12]	SDRAM UMA
0- 1-	Not supported Supported
AD[13]	UMA Device Type
0- 1-	UMA Master UMA Slave
AD[15:14]	BootCS* Device Width
00- 01- 10- 11-	8 bits 16 bits 32 bits Reserved
AD[16]	PCI Retry
0- 1-	Disable Enable **NOTE:** If PCI Retry is enabled and the CPU software configures the PCI interface, all PCI strapping options (expansion ROM, Power Management, VPD, MSI, Hot Swap, BIST) are not required. The CPU enables/disables each of these features, prior to a PCI access to the device.
AD[17]	PCI_0 Expansion ROM Support
0- 1-	Not supported Supported
AD[18]	PCI_1 Expansion ROM Support
0- 1-	Not supported Supported
AD [22:19]	Must pull pull down.

Table 682: Reset Configuration (Continued)

Pin	Configuration Function
AD [23]	SDClkOut/SDClkIn Select
0 1	SDClkIn SDClkOut
24	Internal Space Default Address
0- 1-	default address 0x1400.0000 default address 0xf100.0000
27:25	Must pull low.
AD[28]	PLL Tune
	Pull down **NOTE:** The board design should support the option for pull up.
AD[29]	PLL Divide
	Pull down **NOTE:** The board design should support the option for pull up.
AD[30]	Bypass PLL
0- 1-	PLL Enabled (pull down) PLL Disabled (pull up)
AD[31]	CPU Interface Voltage
0- 1-	2.5V 3.3V

In addition to the above strapping, the GT–64240A samples PCI_0 64EN pin and PCI_1 REQ64* pin during PCI reset deassertion, to recognize whether the PCI interface is connected to a 64-bit backplane. The PCI spec requires a device to sample the REQ64* pin. However, CompactPCI HotSwap ready devices must sample 64EN, instead. Since the GT–64240A PCI_0 interface is HotSwap ready compliant, it samples the 64EN rather than the REQ64* pin.

NOTE: If used in non-HotSwap board, the 64EN pin must be shorted to REQ64* pin.

24.2 Serial ROM Initialization

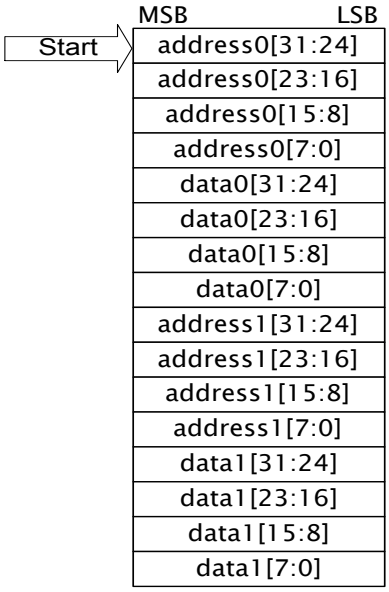
The GT–64240A supports initialization of ALL it's internal and configuration registers and other system components through the I^2C master interface. If serial ROM initialization is enabled (AD[0] pin sampled High on SysRst* deassertion), the GT–64240A I^2C master starts reading initialization data from serial ROM and writes it to the appropriate registers (or to any of GT–64240A interfaces, according to address decoding).

NOTE: If Serial ROM initialization is enabled, the additional required strapping options are AD[1] Serial ROM Byte Offset Width, AD[3:2] Serial ROM Address, AD[4] CPU endianess, AD[28:30] PLL Settings, and AD[31] CPU Interface Voltage.

24.2.1 Serial ROM Data Structure

Serial ROM data structure consists of a sequence of 32-bit address and 32-bit data pairs, as shown in Figure 82.

Figure 82: Serial ROM Data Structure



The GT–64240A reads eight bytes at a time. It compares the first four bytes to the CPU interface address decoding registers and, based on address decoding result, writes the next four bytes to the required target. This scheme enables not only to program the GT–64240A internal registers, but also to initialize other system components. The only limitation is that it supports only single 32-bit writes (no byte enables nor bursts are supported). For example, it is possible to:

- Program the GT–64240A internal registers by setting addresses that match the CPU internal space (default address is 0x14000XXX).
- Program the GT–64240A PCI configuration registers using the PCI_0 Configuration Address and PCI Configuration Data registers (offsets 0xcf8 and 0xcfc).
- Initialize other devices residing on the PCI bus by initiating PCI write transactions.

To support access to the PCI devices that are mapped beyond the 4Gbyte address space, there is also Serial Init PCI High Address register. If initialized to a value other than ‘0’, serial ROM initialization to PCI devices results in DAC cycle on the PCI bus.

The Serial Init Last Data register contains the expected value of last serial data item (default value is 0xffffffff). When the GT–64240A reaches last data, it stops the initialization sequence.

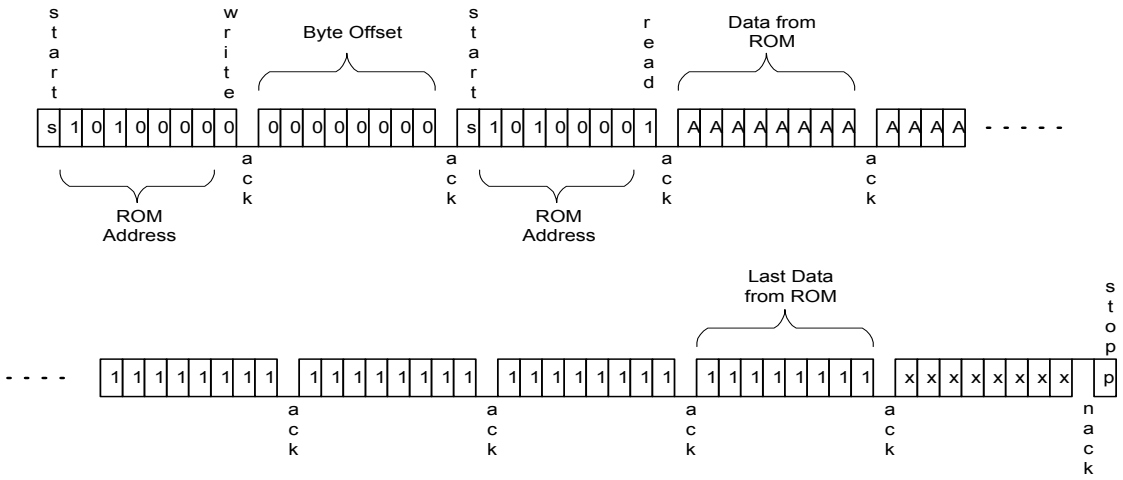
NOTE: The 32-bit address must always be in Little Endian convention.

When using the ROM for initializing the GT–64240A's internal registers, the ROM's data must be in Little Endian convention. This also applies when interfacing with a CPU set to Big Endian.

24.2.2 Serial ROM Initialization Operation

On SysRst* deassertion, the GT–64240A starts the initialization process. It first performs a dummy write access to the serial ROM, with data byte(s) of 0x0, in order to set the ROM byte offset to 0x0. Then, it performs the sequence of reads, until reaches last data item, as shown in Figure 83.

Figure 83: Serial ROM Read Example



For a detailed description of I^2C implementation, see Section 20. “I2C Interface” on page 498.

NOTE: Initialization data must be programmed in the serial ROM starting at offset 0x0

The GT–64240A assumes 7-bit serial ROM address of ‘b10100XX. The value of XX is sampled at reset (see section 24.2.3).

To set the ROM byte offset to ‘0’, the GT–64240A performs a dummy write of one or two bytes, depending on Serial ROM Byte Address strapping.

After receiving the last data identifier (default value is 0xffff.ffff), the GT–64240A receives an additional byte of dummy data. It responds with no-ack and then asserts the stop bit.

24.2.3 Serial ROM Initialization in Multi-GT Configuration

In multi-GT configuration, each GT–64240A device must have its own serial ROM initialization code.

The Serial ROM address bits[1:0] are sampled at reset. Each GT–64240A device must be strapped to a different value, thus having different serial ROM slave addresses.

Each serial ROMs treats slave address bits[1:0] differently. Some serial ROMs use these bits as device chip select. In this case, each slave address corresponds to a different serial ROM. This means that every GT–64240A device has its own ROM on the I^2C bus. Other serial ROMs use these bits as an internal page select. In this case, one serial ROM is shared between all GT–64240A devices.

On SysRst* deassertion, all devices attempt to read from the serial ROM(s). However, since each one of them has a different initialization start address (address bits[1:0] differ), only one master device gains bus ownership. The rest looses arbitration and needs to wait until the first one finish its initialization. This way, each device eventually gains bus mastership and is able to read its ROM and perform initialization.

24.2.4 Restarting Initialization

Initialization can be restarted, either by CPU or even by the serial ROM code itself.

When serial initialization starts, Initialization Control register's InitEn bit is cleared. If when reaching last data, the bit is set to '1' (indicating it was set back to 1 by the initialization code), the initialization process starts again, with ROM address and byte offset taken from the Initialization Control register. This feature effectively allows locating the initialization code in a different location within the ROM or even in several ROMs.

In a similar way, the CPU can later reactivate the initialization sequence. This might be useful, if serial ROM initialization code is changed during system operation.

24.2.5 Other Interfaces During Initialization

During initialization, any PCI attempt to access the GT–64240A results in retry termination. This allows the initialization sequence to program all PCI related registers, prior to an OS access to the GT–64240A.

Also, the DRAM initialization sequence is postponed until serial initialization completes, see Section 5.10 "SDRAM Initialization" on page 116. This guarantees that the SDRAM Timing Parameters register is updated to the right CAS latency prior to DRAM initialization.

NOTE: Do not use serial ROM initialization to initialize the SDRAM.

The CPU access might also need to be postponed until initialization is done. This is achieved by using external hardware to keep the CPU under reset for the entire initialization period. To identify when initialization is done, one of the MPP pins can be configured via the initialization code to act as initialization active output (see Section 19.1 "MPP Multiplexing" on page 480).

24.2.6 Serial ROM Initialization Registers

Table 683: Serial Init PCI High Address, Offset: 0xf320

Bits	Field Name	Function	Initial Value
31:0	PCIHAddr	Bits[63:32] of the PCI address.	0x0

Table 684: Serial Init Last Data, Offset: 0xf324

Bits	Field Name	Function	Initial Value
31:0	DLast	Last Serial Data The GT–64240A finishes with serial ROM initialization when it reaches data that equals this register.	0xffffffff

Table 685: Serial Init Control, Offset: 0xf328

Bits	Field Name	Function	Initial Value
0	Reserved	Reserved.	0x0
7:1	ROMAddr	Serial ROM Address	Bits [1:0]: AD[3:2] sampled at reset. Bits [3:2]: 0x0
15:8	OffsetL	Bits[7:0] of the first byte offset.	0x0
23:16	OffsetH	Bits[15:8] of the first byte offset.	0x0
24	OffsetHEn	Enable 16-bit Byte Offset 0 - 8-bit offset 1 - 16-bit offset	AD[1] sampled at reset.
25	InitEn	Serial Initialization Enable When initialization begins, cleared by the serial ROM initialization logic. Setting this field to '1' restarts initialization.	AD[0] sampled at reset.
27:26	Reserved	Reserved.	0x0
31:28	HAddr	Bits[35:32] of target address (concatenated to address bits[31:0] received from serial ROM).	0x0

Table 686: Serial Init Status, Offset: 0xf32c

Bits	Field Name	Function	Initial Value
4:0	Stat	Serial Initialization Status If the initialization ends successfully, stat uses offset 0x1f. Any other status implies an initialization failure. Stat bits decoding is the same as I^2C Status register bits[7:3]. Read only.	0x1f
31:5	Reserved	Reserved.	0x0

25. GT–64240A CLOCKING

The GT–64240A supports up to four clock domains:

- TClk (core and DRAM clock)
- SysClk (CPU bus clock)
- PClk0
- PClk1

NOTE: In addition, each serial port has a different clock.

TClk is the fastest clock domain. It can run up to 133MHz and drives an internal PLL, that generates the GT–64240A core clock.

TClk is also used as the DRAM interface clock. The same clock source must drive the GT–64240A TClk input and the SDRAM clock (up to 0.5ns clocks skew). The GT–64240A also drives SDClkOut clock. This clock can be used as the SDRAM clock source (after buffered with zero delay clock buffer) instead of TClk, see Section 5.13.1 “SDRAM Clock Output” on page 119.

The CPU interface can run with a dedicated SysClk asynchronous to TClk, or with the core clock (TClk). The CPU interface clock source is determined via AD[5] sampled at reset. SysClk can run up to TClk frequency. When running the CPU interface with the core clock, the SysClk input is not used.

The PCI interfaces clocks can run up to 66MHz, asynchronous to TClk. The two PCI interfaces can run at different asynchronous clocks. There are no limitations on the two interfaces clocks ratio. However, PCI clock frequency must not exceed TClk frequency.

26. DC Characteristics

NOTE: See *AN-67: Powering Up and Powering Down Galileo Technology Integrated Circuits* for information on the power up and power down requirements for a system's power supply.

26.1 Absolute and Recommended Operating Conditions

NOTE: The CPU interface I/O voltage is configured to be 2.5V or 3.3V through reset sample, see Table 682 on page 522.

Table 687: Absolute Maximum Ratings

Symbol	Parameter	Min.	Max.	Unit
V_{CC} core	Core Supply Voltage	-0.3	2.1	V
V_{CC} 2.5V	I/O Supply Voltage	-0.3	4	V
V_{CC} 3.3V	I/O Supply Voltage	-0.3	4	V
V_i	Input Voltage (for 3.3 Volt Tolerant, 2.5)	-0.3	4	V
	Input Voltage (for 5 Volt Tolerant)	-0.3	6.0	V
I_{ik}	Input Protect Diode Current		+-20	mA
I_{ok}	Output Protect Diode Current		+-20	mA
T_c	Operating Case Temperature	0	120	C
T_{stg}	Storage Temperature	- 40	125	C

NOTE: Operation at or beyond the maximum ratings is not recommended or guaranteed. Extended exposure at the maximum rating for extended periods of time may adversely affect device reliability.

Table 688: Recommended Operating Conditions

Symbol	Parameter	Min.	Typ.	Max.	Unit
V_{cc} core	Core Supply Voltage	1.7	1.8	1.9	V
V_{cc} 2.5	I/O Supply Voltage (@ 2.5V CPU)	2.375	2.5	2.625	V
	I/O Supply Voltage (@ 3.3V CPU)	3.15	3.3	3.45	V
V_{cc} 3.3	I/O Supply Voltage	3.15	3.3	3.45	V
V_i	Input Voltage (@ 3.3 V CPU)	0		3.45	V
	Input Voltage (@ 2.5 V CPU)	0		2.625	V
	Input Voltage (for 5VT)	0		5.5	V
V_o	Output Voltage	0		3.45	V
T_c	Operating Case Temperature	0		110	C

NOTE: It is strongly recommended that before designing a system, read *AN-63: Thermal Management for Galileo Technology Products*. This application note describes basic understanding of thermal management of integrated circuits (ICs) and guidelines to ensure optimal operating conditions for GalileoTechnologys products.

Table 689: Pin Capacitance

Symbol	Parameter	Min.	Typ.	Max.	Unit
C	Pin Capacitance	5.2	8.7	9	pF

26.2 DC Electrical Characteristics Over Operating Range

Table 690: DC Electrical Characteristics Over Operating Range

Symbol	Parameter	Test Condition	I/F	Min.	Max.	Unit
V_{ih}	Input HIGH level	Guaranteed logic HIGH level	2.5V 3.3V	1.5 2		V
V_{il}	Input LOW level	Guaranteed logic LOW level	2.5V 3.3V		0.8 0.8	V
V_{oh}	Output HIGH Voltage JTDO, I2CSCK, I2CSDA	IoH = 4 mA	3.3V	2.4		V
V_{oh}	Output HIGH Voltage: E0[14:0], E1[14:0], MDC, MDIO, MPP[31:0], S0[6:0], S1[6:0],	IoH = 8 mA	3.3V	2.4		V

Table 690: DC Electrical Characteristics Over Operating Range (Continued)

Symbol	Parameter	Test Condition	I/F	Min.	Max.	Unit
V_{oh}	Output HIGH Voltage: RspSwap*, SysAD[63:0], SysADC[7:0], SysCmd[8:0], SysRdyOut*, TcDOE*, TcWord[1:0], ValidIn*, SDClkOut	IoH = 8 mA	2.5V 3.3V	1.9 2.4		
V_{oh}	Output HIGH Voltage: AD[31:0], ECC[7:0], SData[31:0], SDQM[7:0]*	IoH = 12 mA	3.3V	2.4		V
V_{oh}	Output HIGH Voltage: BankSel[0], BankSel[1], DAdr[12:0], DWr* , SCAS*, SCS*[3:0], SRAS*	IoH = 24 mA	3.3V	2.4		V
V_{ol}	Output LOW Voltage: JTDO, I2CSCK, I2CSDA	IoL = 4 mA	3.3V		0.4	V
V_{ol}	Output LOW Voltage: E0[14:0], E1[14:0], MDC, MDIO, MPP[31:0], S0[6:0], S1[6:0],	IoL = 8 mA	3.3V		0.4	V
V_{ol}	Output LOW Voltage: RspSwap*, SysAD[63:0], SysADC[7:0], SysCmd[8:0], SysRdyOut*, TcDOE*, TcWord[1:0], ValidIn*, SDClkOut	IoL = 8 mA	2.5V 3.3V		0.4 0.4	
V_{ol}	Output LOW Voltage: AD[31:0], ECC[7:0], SDATA[31:0], SDQM[7:0]*	IoL = 12 mA	3.3V		0.4	V
V_{ol}	Output LOW Voltage: BankSel[0], BankSel[1], DAdr[12:0], DWr*, SCAS*, SCS*[3:0], SRAS*	IoL = 24 mA	3.3V		0.4	V
I_{ih}	Input HIGH Current				10	uA
I_{il}	Input LOW Current				10	uA
I_{ozh}	High Impedance Output Current				10	uA
I_{ozl}	High Impedance Output Current				10	uA
NOTE: Does not apply to SysAD[63:0] and SysADC[7:0].						
I_{ozl}	High Impedance Output Current for SysAD[63:0] and SysADC[7:0].		2.5V 3.3V		50 70	uA

Table 690: DC Electrical Characteristics Over Operating Range (Continued)

Symbol	Parameter	Test Condition	I/F	Min.	Max.	Unit
I_{cc}	Operating Current	I/O VCC3.3 = 3.45 V VCC2.5 =3.45 V f = 133 MHz TClk/66 Mhz PClk			500	mA
		Core VCC1.8 = 1.9 V f = 133 MHz TClk/66 Mhz PClk			950	mA
	ACK640/1*, CBE0/1*[7:0], CLK0/1, DEVSEL0/1*, ENUM0*, FRAME0/1*, GNT0/1*, HS0, IDSEL0/1, INT0/1, IRDY0/1*, LED, P64EN0*, PAD0/1[63:0], PAR0/1, PAR640/1, PERR0/1*, REQ0/1*, REQ640/1*, Rst0/1*, SERR0/1*, STOP0/1*, TRDY0/1*, VREF0/1	See PCI Specification Rev. 2.2				

NOTE: The PCI VREF0/1 pins must be connected directly to the 3.3V or the 5V power plane depending on which voltage level PCI_0/1 supports. VREF0 and VREF1 can be completely independent voltage levels.

26.3 Thermal Data

Table 691 shows the package thermal data for the GT–64240A.

NOTE: For further information, see *AN-63: Thermal Management for Galileo Technology Products*. This application note describes basic understanding of thermal management of integrated circuits (ICs) and guidelines to ensure optimal operating conditions for GalileoTechnologys products.

Table 691: Thermal Data for The GT–64240A in BGA 665

Airflow	Definition	Value		
		0 m/s	1 m/s	2 m/s
Θja	Thermal resistance: junction to ambient.	13.3 C/W	12.1 C/W	10.8 C/W
Ψjt	Thermal characterization parameter: junction to case center.	0.28 C/W	0.31 C/W	0.38 C/W
Θjc	Thermal resistance: junction to case (not air-flow dependent)	4.7 C/W		

26.4 PLL Power Filter Circuit

The GT–64240A has an on-chip PLL to improve its AC timing. To garauntee the stability of the PLL operation, it is critical to insulate the PLL power supply from external signal noise.

26.4.1 PLL Power Supply

The GT–64240A uses two dedicated power supply pins for the PLL:

- H25 - AVCC - Supplies the 1.8V DC for the Analog part of the PLL.
- G25 - AGND - Supplies the GND for the Analog part of the PLL.

The GT–64240A DC specification requires that the PLL GND and the PLL VCC must be supplied with a nominal value of 1.8V DC, with a tolerance of up to 5%. The recommended filtering circuit ensures that the PLL DC specifications are met.

The following sections outline two circuits depending on if the1.8V supply source is available or un-available on board.

26.4.2 PLL Power Filter With a 1.8V Power Supply Available On Board

Figure 84 shows a recommended circuit for the GT–64240A PLL filter.

The circuit's purpose is to prevent the interference of the differential and common modes, usually present in PCBs containing several devices, reaching the PLL power supply traces and, subsequently, disturbing its normal operation.

It is assumed that the 1.8V DC source, necessary to bias the PLL, is available on board.

The user must:

- Use dedicated traces to supply the AGND and the 1.8V AVCC directly from the systems power supply to the filtering circuit.
- Confirm that the PLL supply balls (H25, G25) are isolated from other VCC and GND pins of the GT–64240A.

Figure 84: PLL Power Filter Circuit With Common On-board 1.8V Supply

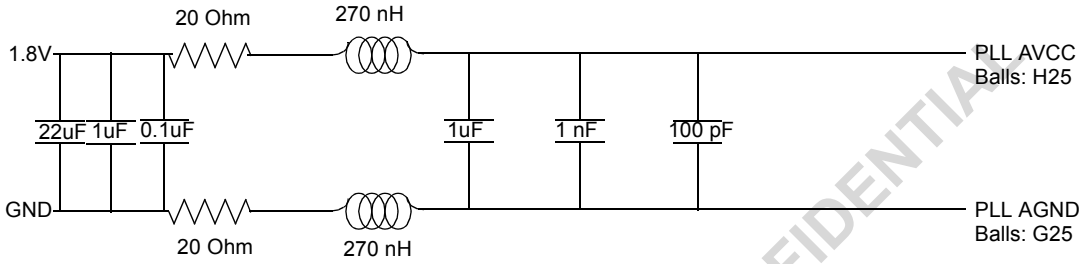
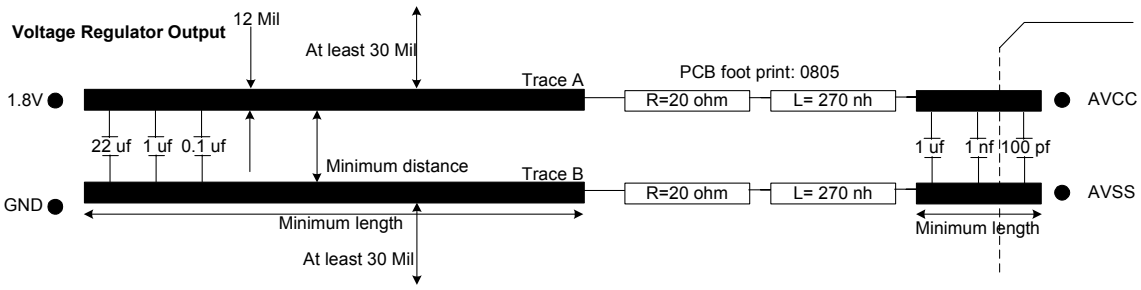


Figure 85: PLL Layout Guideline for a PCI Add-on Card



NOTE: In Figure 85, Traces A and B must be parallel and the same length. Also, Galileo Technology recommends to route the traces on the component side, or print side, and, if possible, leave the area clean in layers.

26.4.3 PLL Power Filter With No 1.8V Power Supply Available On-board (Backplane Layout)

Figure 86 shows a recommended circuit for the GT–64240A PLL filter when a 1.8V power supply is not readily available on board.

For example, for a 5V DC board supply, the industry standard LM317/LP2951 in an SMT packaging can be used to produce the 1.8V DC for the PLL, with the 240 Ohm resistors connected to the output pin and an adjust pin as indicated.

The user must:

- Use dedicated traces to supply the AGND and the 1.8V AVCC directly from the systems power supply to the filtering circuit.

- Confirm that the PLL supply balls (H25, G25) are isolated from other VCC and GND pins of the GT–64240A.

Figure 86: PLL Power Filter Circuit With Dedicated 1.8V Supply

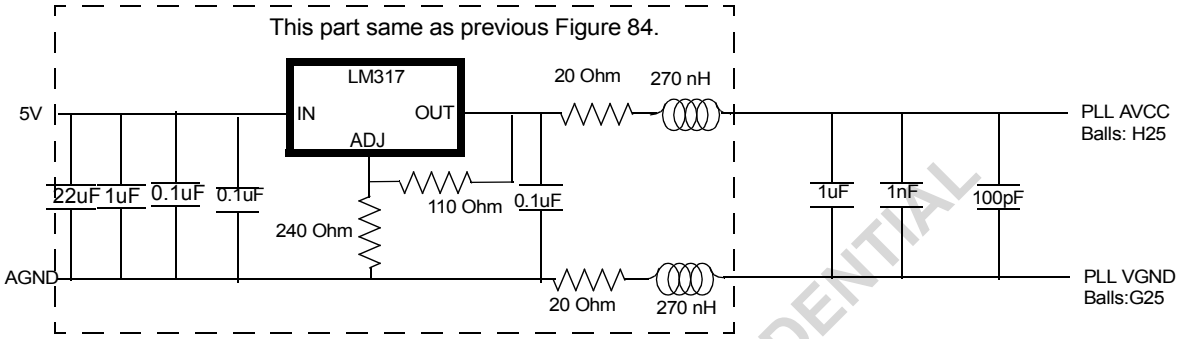
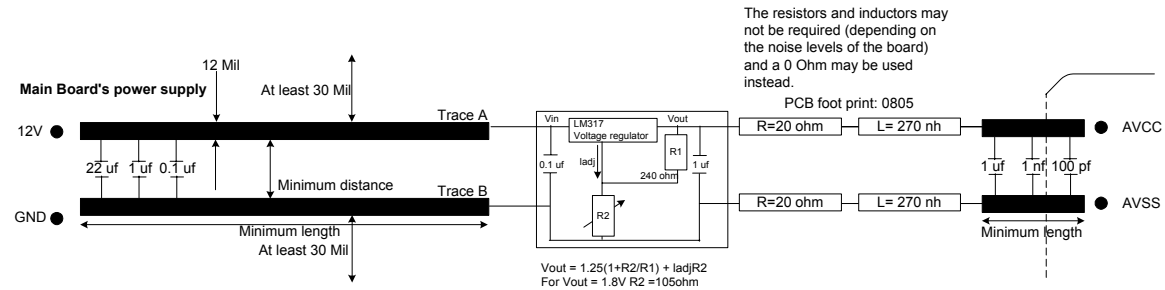


Figure 87: PLL Layout Guideline for Backplane Layout



26.4.4 PLL Power Filter Layout Considerations

For the two dedicated traces going from the supply source to the filtering circuit, the following must be garaunteed:

- Provide each trace with a minimum width of 20 mil.
- Route the traces in parallel, with minimal spacing.
- Give each trace an equal and minimal length.
- Route the traces in noise-free areas and as far as possible from high current traces.
- Make the filtering components SMT, 0603 size.
- Place the 0.1nF capacitor as close as possible to the PLL DC supply pins.
- Place the capacitors in the shown order, with the smallest capacitor closest to the PLL's DC Supply Pins.

27. AC TIMING

NOTE: The following targets are subject to change.

Table 692: 100 MHz AC Timing

Signals	Description	100MHz Min.	100MHz Max.	Units	Loading
Clock					
TClk/SysClk	Frequency	66	100	MHz	
TClk/SysClk	Cycle Time	10	15	ns	
TClk/SysClk	Clock High	4.5	5.5	ns	
TClk/SysClk	Clock Low	4.5	5.5	ns	
TClk/SysClk	Rise Time		2	ns	
TClk/SysClk	Fall Time		2	ns	
CPU Interface **NOTE:** All CPU interface Output Delays, Setup, and Hold times are referred to TClk rising edge. If using a CPU interface with SysClk, all setup, hold, and output delay parameters increase by 3 ns. The setup decreases by 1.5 ns .					
SysADC[7:0], SysAD[63:0], Release*, PReq*, TcTce*	Setup	3		ns	
SysCmd[8:0], ValidOut*, SysRdyIn[2:0]	Setup	3.5		ns	
TcMatch	Setup	4.5		ns	
SysADC[7:0], SysAD[63:0], SysCmd[8:0], ValidOut*, Release*, PReq*, SysRdyIn[2:0], TcMatch, TcTCE*	Hold	0.5		ns	
TcDOE*, TcWord[1:0]	Output Delay	1	5	ns	
SysADC[7:0], SysCmd[8:0], ValidIn*, SysRdyOut*, RspSwap*, RdWrRdy*	Output Delay	1	5.5	ns	20pF
SysAD[63:0], PAck*	Output Delay	1	6	ns	
SysRst*	Active	1		ms	
PCI Interface **NOTE:** All PCI interface Output Delays, Setup, and Hold times are referred to PClk rising edge.					
PClk0,PClk1	Frequency		66	MHz	
PClk0,PClk1	Clock Period	15	∞	ns	

Table 692: 100 MHz AC Timing (Continued)

Signals	Description	100MHz Min.	100MHz Max.	Units	Loading
Rst0*, Rst1*	Active	1		ms	
FRAME0/1*, IRDY0/1*, TRDY0/1*, STOP0/1*, IDSEL0/1, DEVSEL0/1* REQ640/1*, ACK640/1*, PAR640/1, PERR0/1*, PAD0/1[63:0], CBE0/1[7:0]*, PAR0/1, GNT0/1*	Setup	3		ns	
FRAME0/1*, IRDY0/1*, PAD0/1[63:0], TRDY0/1*, STOP0/1*, IDSEL0/1, PAR640/1, DEVSEL0/1* GNT0/1*, REQ640/1*, ACK640/1*,PAR0/1, PERR0/1*, CBE0/1[7:0]*	Hold	0		ns	
FRAME0/1*, TRDY0/1*, IRDY0/1* DEVSEL0/1*, PAD0/1[63:0], STOP0/1*, CBE0/1[7:0]*, REQ640/1*, ACK640/1*, REQ0/1*, PAR0/1 PERR0/1*, SERR0/1*, PAR640/1 **NOTE:** Output delays are measured as specified in PCI spec rev. 2.2 section 7.6.4.3	Output Delay	2	6	ns	10pF
SDRAM Interface (TClk) **NOTE:** All SDRAM interface Output Delays, Setup, and Hold times are referred to the **TClk's** rising edge.					
ECC[7:0], SData[63:0]	Setup	1.8		ns	
SData[63:0], ECC[7:0]	Hold	0.5		ns	
SData[63:0], BankSel[1:0], DAdr[12:0], SRAS*, SCAS*, SCS[3:0]*, SDQM*[7:0], DWr*, SData[63:0]	Output Delay	1	4.5	ns	30pF
ECC[7:0]	Output Delay	1	4.8	ns	30pF
sdclckout	Output Delay	1.5	3.5	ns	50pF

Table 692: 100 MHz AC Timing (Continued)

Signals	Description	100MHz Min.	100MHz Max.	Units	Loading
Device Interface **NOTE:** All Device interface Output Delays, Setup, and Hold times are referred to TClk rising edge.					
AD[31:0], Ready*	Setup	2.5		ns	
AD[31:0], Ready*	Hold	0		ns	
NOTE: The Ready* setup and hold parameters are determined with the Device Interface Control register's ReadyS bit [18] set to '1'. For further details, see Section 7.4 "Ready* Support" on page 144.					
Wr[3:0]*, ALE, AD[31:0]	Output Delay	1	4	ns	25pF
CSTiming*, BAdr[2:0]	Output Delay	1	4.5	ns	30pF
MPP Interface **NOTE:** All MPP pins Output Delays, Setup, and Hold times are referred to TClk rising edge, unless stated otherwise. The following MPP maximum output delay parameters vary according to the multipurpose pin being used.					
MREQ*, MGNT*, EOT[7:0], DMAReq[7:0]*, TCEn[7:0], GPP[31:0]	Setup	3		ns	20pF
PCI0Req[5:0]*, PCI1Req[5:0]*	Setup	5		ns	20pF
MREQ*, MGNT*, EOT[7:0], DMAReq[7:0]*, TCEn[7:0],	Hold	0		ns	20pF
PCI0Req[5:0]*, PCI1Req[5:0]*	Hold	0		ns	20pF
GPP[31:0]	Hold	TBD		ns	20pF
DBurst*/Dlast*	Output Delay	1.5	6	ns	20pf
MREQ*,MGNT*, TCTCnt[7:0], GPP[31:0], InitAct	Output Delay	1.5	5.5	ns	20pf
PCI0Gnt[5:0]*, PCI1Gnt[5:0]*	Output Delay	3	8	ns	20pf
DMAAck[7:0]*,	Output Delay	1.5	6	ns	20pF
PME0/1, INT[4:0], WDE*, WDNMI* are asyncronus signals.					
Ethernet MII Interface **NOTES:** All receive pins Setup, and Hold times are referred to RxClk rising edge. All transmit pins Output Delays, Setup, and Hold times are referred to TxClk rising edge. The multiplier in the following timing numbers is 1 for 100Mb/s operation, and 10 for 10Mb/s operation					
MTxCLK0/1	Cycle		25	Mhz	
MTxCLK0/1	Clock Period	40		ns	

Table 692: 100 MHz AC Timing (Continued)

Signals	Description	100MHz Min.	100MHz Max.	Units	Loading
MTxD0/1[3:0],MTxEN0/1	Output Delay	3.3	8	ns	20pf
MRXclk Clock	Frequency		25	Mhz	
MRXclk Clock	Clock Period	40		ns	
MRxD0/1[3:0], MRxDV0/1, MRxER0/1, MCOL0/1, MCRS0/1	Setup	10		ns	
MRxD0/1[3:0], MRxDV0/1, MRxER0/1, MCOL0/1, MCRS0/1	Hold	5		ns	
Ethernet RMII Interface **NOTES:** All receive pins Setup, and Hold times are referred to RxClk rising edge. All transmit pins Output Delays, Setup, and Hold times are referred to TxClk rising edge. The multiplier in the following timing numbers is 1 for 100Mb/s operation, and 10 for 10Mb/s operation					
RMII Clock	Frequency		50	Mhz	
RMII Clock	Clock Period	20		ns	
DvCRS, RxD[1:0]	Setup	2		ns	
DvCRS, RxD[1:0]	Hold	1		ns	
TxD[1:0],TxEN	Output Delay	3	14	ns	20pf
MPSC Interface **NOTE:** All receive pins Setup, and Hold times are referred to RClk rising edge. All transmit pins Output Delays, Setup, and Hold times are referred to TClk rising edge.					
TClk (MPSC Transmit Timing)	Frequency		50	Mhz	
TClk (MPSC Transmit Timing)	Clock Period	20		ns	
CTS*	Setup	0		ns	
CTS*	Hold	2.5		ns	
TXD, RTS*	Output Delay	5	14.5	ns	
RClk (MPSC Receive Timing)	Frequency		50	Mhz	
RClk (MPSC Receive Timing)	Clock Period	20		ns	
RXD, CD*	Setup	1			
RXD, CD*	Hold	2.5			
I_2C Interface					
I_2C Clock	Frequency		4	Mhz	
I_2C Data	Clock Period	250		ns	

Table 692: 100 MHz AC Timing (Continued)

Signals	Description	100MHz Min.	100MHz Max.	Units	Loading
I_2C Data	Setup	10		ns	
I_2C Data	Hold	3		ns	
I_2C Data	Output Delay	1	15	ns	20pF

Table 693: 133 MHz AC Timing

Signals	Description	133MHz Min.	133MHz Max.	Units	Loading
Clock					
TClk/SysClk	Frequency	66	133	MHz	
TClk/SysClk	Cycle Time	7.5	15	ns	
TClk/SysClk	Clock High	3.3	4.2	ns	
TClk/SysClk	Clock Low	3.3	4.2	ns	
TClk/SysClk	Rise Time		2	ns	
TClk/SysClk	Fall Time		2	ns	
CPU Interface **NOTE:** All CPU interface Output Delays, Setup, and Hold times are referred to TClk rising edge. If using a CPU interface with SysClk, all setup, hold, and output delay parameters increase by 3 ns. The setup decreases by 1.5 ns .					
SysAD[63:0	Setup	2.6		ns	
SysCmd[8:0], ValidOut*	Setup	2.4		ns	
SysADC[7:0], TcMatch	Setup	2.3		ns	
SysRdyIn[2:0], PReq*	Setup	2.0		ns	
Release*, TcTce*	Setup	1.9		ns	
SysADC[7:0], SysAD[63:0], SysCmd[8:0], ValidOut*, Release*, PReq*, SysRdyIn[2:0], TcMatch, TcTce*	Hold	0		ns	
SysADC[7:0], SysAD[63:0]	Output Delay	1	4.2	ns	25pf
ValidIn*, TcDOE*	Output Delay	1.4	3.5	ns	25pf
RspSwap*, SysRdyOut*, PAck*,	Output Delay	1.4	3.7	ns	25pf

Table 693: 133 MHz AC Timing (Continued)

Signals	Description	133MHz Min.	133MHz Max.	Units	Loading
SysCmd[8:0], TcWord[1:0]	Output Delay	1.4	4.0	ns	25pf
SysRst*	Active	1		ms	
PCI Interface **NOTE:** All PCI interface Output Delays, Setup, and Hold times are referred to PClk rising edge.					
PClk0,PClk1	Frequency		66	MHz	
PClk0,PClk1	Clock Period	15	∞	ns	
Rst0*, Rst1*	Active	1		ms	
FRAME0/1*, IRDY0/1*, TRDY0/1*, STOP0/1*, IDSEL0/1, DEVSEL0/1* REQ640/1*, ACK640/1*, PAR640/1, PERR0/1*, PAD0/1[63:0], CBE0/1[7:0]*, PAR0/1, GNT0/1*	Setup	3		ns	
FRAME0/1*, IRDY0/1*, PAD0/1[63:0], TRDY0/1*, STOP0/1*, IDSEL0/1, PAR640/1, DEVSEL0/1* GNT0/1*, REQ640/1*, ACK640/1*,PAR0/1, PERR0/1*, CBE0/1[7:0]*	Hold	0		ns	
FRAME0/1*, TRDY0/1*, IRDY0/1* DEVSEL0/1*, PAD0/1[63:0], STOP0/1*, CBE0/1[7:0]*, REQ640/1*, ACK640/1*, REQ0/1*, PAR0/1 PERR0/1*, SERR0/1*, PAR640/1 **NOTE:** Output delays are measured as specified in PCI spec rev. 2.2 section 7.6.4.3	Output Delay	2	6	ns	10pF
SDRAM Interface (TClk) **NOTE:** All SDRAM interface Output Delays, Setup, and Hold times are referred to the **TClk's** rising edge.					
ECC[7:0]	Setup	0.8		ns	
SData[63:0]	Setup	1.3		ns	
SData[63:0], ECC[7:0]	Hold	0.4		ns	
SCAS*, BankSel[1:0]	Output Delay	1.1	3.1	ns	50pF
DAdr[12:0]	Output Delay	1.1	3.6	ns	50pF

Table 693: 133 MHz AC Timing (Continued)

Signals	Description	133MHz Min.	133MHz Max.	Units	Loading
SDQM*[7:0]	Output Delay	1.1	3.7	ns	30pF
SDQM*[7:0], DWr*, SRAS*	Output Delay	1.1	3.7	ns	50pF
SCS[3:0]*	Output Delay	1.2	3.8	ns	50pF
SData[63:0], SCS[3:0]*	Output Delay	1.2	3.8	ns	30pF
ECC[7:0]	Output Delay	1.3	4.1	ns	30pF
SDRAM Interface (SDClkOut) **NOTE:** All SDRAM interface Output Delays are referred to the **SDClkOut** rising edge.					
BankSel[1:0], DAdr[12:0], SRAS*, SCAS*, DWr*	Output Delay	0.5	1.4	ns	50pF
SDQM*[7:0]	Output Delay	0.5	1.4	ns	30pF
SDClkOut (rising edge relative to TClk)	Output Delay	0.6	1.7	ns	10pF
SData[63:0]	Output Delay	0.6	2.1	ns	30pF
ECC[7:0]	Output Delay	0.8	2.4	ns	30pF
SDRAM Interface (SDClkIn) **NOTE:** The SData[63:0] and ECC[7:0] setup and hold parameters are determined with the SDRAM Timing Parameter register's RdDelay bit [12] set to '1'. For further details, see Section 5.13 "SDRAM Clocking" on page 118. All SDRAM interface Setup, and Hold times are referred to the SDClkInrising edge. These parameters also apply when using the SDClkOut as the DRAM clock. In this mode, SDClkIn is not generated externally. SDClkIn is directly connected within the chip from the SDClkOut pad.					
SDClkIn (rising edge relative to TClk)	Setup		3.8	ns	
SDClkIn (rising edge relative to TClk)	Hold	0		ns	
ECC[7:0], SData[63:0]	Setup	0.15		ns	
ECC[7:0], SData[63:0]	Hold	0.7		ns	
Device Interface **NOTE:** All Device interface Output Delays, Setup, and Hold times are referred to TClk rising edge.					
AD[31:0]	Setup	1.6		ns	

Table 693: 133 MHz AC Timing (Continued)

Signals	Description	133MHz Min.	133MHz Max.	Units	Loading
Ready*	Setup	1.8		ns	
AD[31:0], Ready*	Hold	0		ns	
NOTE: The Ready* setup and hold parameters are determined with the Device Interface Control register's ReadyS bit [18] set to '1'. For further details, see Section 7.4 "Ready* Support" on page 144.					
AD[31:0]	Output Delay	1	3.2	ns	25pF
Wr[3:0]*, ALE	Output Delay	1	2.8	ns	25pF
CSTiming*	Output Delay	1	3.8	ns	25pF
BAdr[2:0]	Output Delay	1	3.5	ns	25pF
MPP Interface **NOTE:** All MPP pins Output Delays, Setup, and Hold times are referred to TClk rising edge, unless stated otherwise. The following MPP maximum output delay parameters vary according to the multipurpose pin being used.					
MREQ*, MGNT*, EOT[7:0], DMAReq[7:0]*, TCEn[7:0], GPP[31:0]	Setup	2.3		ns	
PCI0Req[5:0]*, PCI1Req[5:0]* **NOTE:** Referred to the PClk0/1 rising edge.	Setup	4		ns	
MREQ*, EOT[7:0], DMAReq[7:0]*, TCEn[7:0]	Hold	0		ns	
PCI0Req[5:0]*, PCI1Req[5:0]* **NOTE:** Referred to the PClk0/1 rising edge.	Hold	0		ns	
GPP[31:0]	Hold	TBD		ns	
DBurst*/DLast*	Output Delay	2.0	5.1-5.6	ns	20pf
MREQ*, GPP[31:0], MGNT*, TCTCnt[7:0], InitAct	Output Delay	1.6	4.5-5.0	ns	20pf
PCI0Gnt[5:0]*, PCI0Gnt[5:0]* **NOTE:** Referred to the PClk0/1 rising edge.	Output Delay	3	6.6-7.5	ns	20pf
DMAAck[7:0]*,	Output Delay	1.7	4.6-5.4	ns	20pF
PME0/1, INT[4:0], WDE*, WDNMI* are asyncronus signals.					

Table 693: 133 MHz AC Timing (Continued)

Signals	Description	133MHz Min.	133MHz Max.	Units	Loading
Ethernet MII Interface **NOTES:** All receive pins Setup, and Hold times are referred to RxClk rising edge. All transmit pins Output Delays, Setup, and Hold times are referred to TxClk rising edge. The multiplier in the following timing numbers is 1 for 100Mb/s operation, and 10 for 10Mb/s operation					
MTxCLK0/1	Cycle		25	Mhz	
MTxCLK0/1	Clock Period	40		ns	
MTxD0/1[3:0],MTxEN0/1	Output Delay	3.3	8	ns	20pf
MRXclk Clock	Frequency		25	Mhz	
MRXclk Clock	Clock Period	40		ns	
MRxD0/1[3:0], MRxDV0/1, MRxER0/1, MCOL0/1, MCRS0/1	Setup	10		ns	
MRxD0/1[3:0], MRxDV0/1, MRxER0/1, MCOL0/1, MCRS0/1	Hold	5		ns	
Ethernet RMII Interface **NOTES:** All receive pins Setup, and Hold times are referred to RxClk rising edge. All transmit pins Output Delays, Setup, and Hold times are referred to TxClk rising edge. The multiplier in the following timing numbers is 1 for 100Mb/s operation, and 10 for 10Mb/s operation					
RMII Clock	Frequency		50	Mhz	
RMII Clock	Clock Period	20		ns	
DvCRS, RxD[1:0]	Setup	2		ns	
DvCRS, RxD[1:0]	Hold	1		ns	
TxD[1:0],TxEN	Output Delay	3	14	ns	20pf
MPSC Interface **NOTE:** All receive pins Setup, and Hold times are referred to RClk rising edge. All transmit pins Output Delays, Setup, and Hold times are referred to TClk rising edge.					
TClk (MPSC Transmit Timing)	Frequency		50	Mhz	
TClk (MPSC Transmit Timing)	Clock Period	20		ns	
CTS*	Setup	0		ns	
CTS*	Hold	2.5		ns	
TXD, RTS*	Output Delay	5	14.5	ns	
RClk (MPSC Receive Timing)	Frequency		50	Mhz	
RClk (MPSC Receive Timing)	Clock Period	20		ns	

Table 693: 133 MHz AC Timing (Continued)

Signals	Description	133MHz Min.	133MHz Max.	Units	Loading
RXD, CD*	Setup	1			
RXD, CD*	Hold	2.5			
I_2C Interface					
I_2C Clock	Frequency		4	Mhz	
I_2C Data	Clock Period	250		ns	
I_2C Data	Setup	10		ns	
I_2C Data	Hold	3		ns	
I_2C Data	Output Delay	1	15	ns	20pF

28. PINOUT TABLE, 665 PIN BGA

NOTE: The following table is sorted by ball number.

Table 694: GT–64240A Pinout Table

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
A03–A25		***A26–A29***		***B23–B30***	
A03	BAdr[0]	A26	PReq*	B23	SData[62]
A04	Wr[0]	A27	NC	B24	TDI
A05	SData[1]	A28	NC	B25	NC
A06	SData[4]	A29	SysRdyOut*	B26	NC
A07	SData[7]	***B02–B22***		B27	NC
A08	SData[10]	B02	VCC 3.3	B28	NC
A09	SData[13]	B03	BAdr[1]	B29	NC
A10	ECC[0]	B04	Wr[1]	B30	VCC 2.5
A11	SCAS*	B05	SData[32]	***C01–C17***	
A12	SCS[1]*	B06	SData[35]	C01	AD[0]
A13	DAdr[3]	B07	SData[38]	C02	ALE
A14	DAdr[8]	B08	SData[41]	C03	BAdr[2]
A15	SDClkOut	B09	SData[44]	C04	Wr[2]
A16	SDQM[6]*	B10	SData[47]	C05	SData[0]
A17	ECC[7]	B11	DWr*	C06	SData[3]
A18	SData[18]	B12	SCS[0]*	C07	SData[6]
A19	SData[52]	B13	DAdr[2]	C08	SData[9]
A20	SData[23]	B14	DAdr[7]	C09	SData[12]
A21	SData[57]	B15	DAdr[11]	C10	SData[15]
A22	SData[28]	B16	SDQM[2]*	C11	ECC[5]
A23	SData[31]	B17	ECC[3]	C12	SDQM[5]*
A24	SData[63]	B18	SData[49]	C13	DAdr[1]
A25	NC	B19	SData[20]	C14	DAdr[6]
		B20	SData[54]	C15	BankSel[1]
		B21	SData[25]	C16	SCS[3]*
		B22	SData[59]	C17	ECC[6]

Table 694: GT–64240A Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
C18–C31		***D14–D31***		***E10–E31***	
C18	SData[17]	D14	DAdr[5]	E10	SData[14]
C19	SData[51]	D15	BankSel[0]	E11	ECC[4]
C20	SData[22]	D16	SCS[2]*	E12	SDQM[4]*
C21	SData[56]	D17	ECC[2]	E13	SRAS*
C22	SData[27]	D18	SData[48]	E14	DAdr[4]
C23	SData[30]	D19	SData[19]	E15	DAdr[10]
C24	TMS	D20	SData[53]	E16	DAdr[12]
C25	PAck*	D21	SData[24]	E17	SDQM[7]*
C26	NC	D22	SData[58]	E18	SData[16]
C27	CPUInt*	D23	SData[61]	E19	SData[50]
C28	NC	D24	TRST	E20	SData[21]
C29	TcMatch	D25	SysRst*	E21	SData[55]
C30	NC	D26	NC	E22	SData[26]
C31	ValidIn*	D27	NC	E23	SData[29]
D01–D13		D28	NC	E24	TCK
D01	AD[3]	D29	NC	E25	SysClk
D02	AD[2]	D30	NC	E26	RspSwap*
D03	AD[1]	D31	ValidOut*	E27	Release*
D04	Wr[3]	***E01–E09***		E28	TcWord[1]
D05	Ready*	E01	AD[7]	E29	TcWord[0]
D06	SData[34]	E02	AD[6]	E30	NC
D07	SData[37]	E03	AD[5]	E31	SysCmd[0]
D08	SData[40]	E04	AD[4]		
D09	SData[43]	E05	CSTiming*		
D10	SData[46]	E06	SData[2]		
D11	ECC[1]	E07	SData[5]		
D12	SDQM[1]*	E08	SData[8]		
D13	DAdr[0]	E09	SData[11]		

Table 694: GT–64240A Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
F01–F29		***F30–F31***		***H06–H07, H25–H31***	
F01	AD[12]	F30	SysCmd[3]	H06	AD[19]
F02	AD[11]	F31	SysCmd[2]	H07	VCC 3.3
F03	AD[10]	***G01–G10, G22–G31***		H25	AVCC
F04	AD[9]	G01	AD[18]	H26	TcDOE*
F05	AD[8]	G02	AD[17]	H27	SysRdyIn[2]*
F06	SData[33]	G03	Ad[16]	H28	NC
F07	SData[36]	G04	AD[15]	H29	NC
F08	SData[39]	G05	AD[14]	H30	SysRdyIn[1]*
F09	SData[42]	G06	AD[13]	H31	SysRdyIn[0]*
F10	SData[45]	G07	GND	***J01–J07, J25–J31***	
F11	VCC 3.3	G08	VCC 3.3	J01	AD[30]
F12	SDQM[0]*	G09	GND	J02	AD[29]
F13	VCC 3.3	G10	VCC 3.3	J03	AD[28]
F14	VCC Core	G22	VCC 3.3	J04	AD[27]
F15	DAdr[9]	G23	GND	J05	AD[26]
F16	VCC Core	G24	VCC 3.3	J06	AD[25]
F17	SDQM[3]*	G25	AGND	J07	VCC 3.3
F18	GND	G26	SysCmd[8]	J25	VCC 2.5
F19	VCC 3.3	G27	SysCmd[7]	J26	NC
F20	VCC 3.3	G28	NC	J27	NC
F21	VCC 3.3	G29	SysCmd[6]	J28	NC
F22	VCC 3.3	G30	TcTCE*	J29	NC
F23	SData[60]	G31	NC	J30	NC
F24	TClk	***H01–H05***		J31	NC
F25	JTDO	H01	AD[24]		
F26	SysCmd[1]	H02	AD[23]		
F27	NC	H03	AD[22]		
F28	SysCmd[5]	H04	AD[21]		
F29	SysCmd[4]	H05	AD[20]		

Table 694: GT–64240A Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
K01–K07, K25–K31		***M01–M06, M26–M31***		***N26–N31***	
K01	E0[3]	M01	E0[13]	N26	VCC Core
K02	E0[2]	M02	E0[12]	N27	SysAD[28]
K03	E0[1]	M03	E0[11]	N28	SysAD[29]
K04	E0[0]	M04	E0[10]	N29	NC
K05	AD[31]	M05	E0[9]	N30	SysADC[2]
K06	VCC 3.3	M06	VCC 3.3	N31	SysAD[4]
K07	VCC 3.3	M26	SysAD[31]	***P01–P06, P13–19, P26–P31***	
K25	VCC 2.5	M27	NC	P01	E1[7]
K26	NC	M28	SysADC[5]	P02	E1[6]
K27	NC	M29	SysADC[3]	P03	E1[5]
K28	NC	M30	SysAD[22]	P04	E1[4]
K29	NC	M31	SysAD[24]	P05	E1[3]
K30	NC	***N01–N06, N13–N19***		P06	E1[2]
K31	NC	N01	E1[1]	P13	GND
L01–L06, L26–L31		N02	E1[0]	P14	GND
L01	E0[8]	N03	MDIO	P15	GND
L02	E0[7]	N04	MDC	P16	GND
L03	E0[6]	N05	E0[14]	P17	GND
L04	E0[5]	N06	VCC 3.3	P18	GND
L05	E0[4]	N13	GND	P19	GND
L06	VCC 3.3	N14	GND	P26	VCC Core
L26	VCC 2.5	N15	GND	P27	SysAD[18]
L27	NC	N16	GND	P28	SysAD[21]
L28	SysADC[7]	N17	GND	P29	SysAD[52]
L29	SysAD[56]	N18	GND	P30	SysAD[54]
L30	SysAD[55]	N19	GND	P31	SysAD[37]
L31	SysAD[26]				

Table 694: GT–64240A Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
R01–R06, R13–R19, R26–R31		***T13–T31***		***U26–U31***	
R01	E1[12]	T13	GND	U26	VCC 2.5
R02	E1[11]	T14	GND	U27	SysAD[60]
R03	E1[10]	T15	GND	U28	SysAD[44]
R04	E1[9]	T16	GND	U29	SysADC[1]
R05	E1[8]	T17	GND	U30	SysAD[25]
R06	GND	T18	GND	U31	SysAD[8]
R13	GND	T19	GND	***V01–V06, V13–19, V26–V31***	
R14	GND	T26	SysAD[32]	V01	I2CSCK
R15	GND	T27	SysAD[27]	V02	S1[6]
R16	GND	T28	SysAD[3]	V03	S1[5]
R17	GND	T29	SysADC[0]	V04	S1[4]
R18	GND	T30	SYSAD[17]	V05	S1[3]
R19	GND	T31	SysAD[12]	V06	S1[2]
R26	VCC 2.5	***U01–U06, U13–U19***		V13	GND
R27	SysAD[30]	U01	S1[1]	V14	GND
R28	SysADC[6]	U02	S1[0]	V15	GND
R29	SysAD[7]	U03	S0[6]	V16	GND
R30	SysAD[50]	U04	S0[5]	V17	GND
R31	SysAD[42]	U05	S0[4]	V18	GND
T01–T06		U06	VCC Core	V19	GND
T01	S0[3]	U13	GND	V26	SysAD[15]
T02	S0[2]	U14	GND	V27	SysAD[19]
T03	S0[1]	U15	GND	V28	SysAD[11]
T04	S0[0]	U16	GND	V29	SysAD[57]
T05	E1[14]	U17	GND	V30	SysAD[58]
T06	E1[13]	U18	GND	V31	SysAD[46]
		U19	GND		

Table 694: GT–64240A Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
W01–W06, W13–W19, W26–W31		***Y26–Y31***		***AB25–AB31***	
W01	MPP[3]	Y26	VCC Core	AB25	VCC 2.5
W02	MPP[2]	Y27	SysAD[23]	AB26	VCC 2.5
W03	MPP[1]	Y28	SysAD[5]	AB27	SysAD[36]
W04	MPP[0]	Y29	SysAD[20]	AB28	SysAD[35]
W05	I2CSDA	Y30	SysAD[10]	AB29	SysAD[2]
W06	VCC Core	Y31	SysAD[61]	AB30	SysADC[4]
W13	GND	***AA01–AA06, AA26–AA31***		AB31	SysAD[6]
W14	GND	AA01	MPP[13]	***AC01–AC07, AC25–AC31***	
W15	GND	AA02	MPP[12]	AC01	MPP[25]
W16	GND	AA03	MPP[11]	AC02	MPP[24]
W17	GND	AA04	MPP[10]	AC03	MPP[23]
W18	GND	AA05	MPP[9]	AC04	MPP[22]
W19	GND	AA06	VCC 3.3	AC05	MPP[21]
W26	GND	AA26	VCC 2.5	AC06	MPP[20]
W27	SysAD[43]	AA27	SysAD[14]	AC07	VCC 3.3
W28	SysAD[41]	AA28	SysAD[45]	AC25	VCC 2.5
W29	SysAD[39]	AA29	SysAD[9]	AC26	SysAD[16]
W30	SysAD[33]	AA30	SysAD[48]	AC27	SysAD[63]
W31	SysAD[1]	AA31	SysAD[47]	AC28	SysAD[62]
Y01–Y06		***AB01–AB07***		AC29	SysAD[59]
Y01	MPP[8]	AB01	MPP[19]	AC30	SysAD[51]
Y02	MPP[7]	AB02	MPP[18]	AC31	SysAD[13]
Y03	MPP[6]	AB03	MPP[17]	***AD01–AD04***	
Y04	MPP[5]	AB04	MPP[16]	AD01	MPP[31]
Y05	MPP[4]	AB05	MPP[15]	AD02	MPP[30]
Y06	VCC 3.3	AB06	MPP[14]	AD03	MPP[29]
		AB07	VCC 3.3	AD04	MPP[28]

Table 694: GT–64240A Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
AD05–AD07, AD25–AD31		***AE30–AE31***		***AF27–AF31***	
AD05	MPP[27]	AE30	PAD0[36]	AF27	PAD0[38]
AD06	MPP[26]	AE31	PAD0[37]	AF28	PAD0[39]
AD07	VCC 3.3	***AF01–AF26***		AF29	PAD0[40]
AD25	VCC Core	AF01	PAD1[25]	AF30	PAD0[41]
AD26	SysAD[53]	AF02	PAD1[26]	AF31	PAD0[42]
AD27	SysAD[49]	AF03	PAD1[27]	***AG01–AG23***	
AD28	SysAD[40]	AF04	PAD1[28]	AG01	PAD1[22]
AD29	SysAD[38]	AF05	PAD1[29]	AG02	PAD1[23]
AD30	SysAD[34]	AF06	PAD1[30]	AG03	IDSEL1
AD31	SysAD[0]	AF07	PAD1[6]	AG04	CBE1[3]*
AE01–AE10, AE22–AE29		AF08	PAD1[0]	AG05	PAD1[24]
AE01	PAD1[31]	AF09	CBE1[4]*	AG06	PAD1[11]
AE02	REQ1*	AF10	PAD1[59]	AG07	PAD1[7]
AE03	GNT1*	AF11	VCC 3.3	AG08	PAD1[1]
AE04	CLK1	AF12	GND	AG09	CBE1[5]*
AE05	Rst1*	AF13	VCC 3.3	AG10	PAD1[60]
AE06	INT1*	AF14	PAD1[38]	AG11	PAD1[54]
AE07	GND	AF15	VCC 3.3	AG12	PAD1[49]
AE08	VREF1	AF16	REQ0*	AG13	PAD1[44]
AE09	VCC Core	AF17	VREF0	AG14	PAD1[39]
AE10	VCC Core	AF18	VCC 3.3	AG15	PAD1[33]
AE22	VCC 3.3	AF19	GND	AG16	GNT0*
AE23	VCC 3.3	AF20	IRDY0*	AG17	PAD0[27]
AE24	VCC Core	AF21	VCC 3.3	AG18	IDSEL0
AE25	VCC 3.3	AF22	PAD0[12]	AG19	PAD0[19]
AE26	PAD0[32]	AF23	PAD0[7]	AG20	Frame0*
AE27	PAD0[33]	AF24	PAD0[1]	AG21	SERR0*
AE28	PAD0[34]	AF25	HS	AG22	PAD0[13]
AE29	PAD0[35]	AF26	PAR640	AG23	CBE0[0]*

Table 694: GT–64240A Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
AG24–AG31		***AH21–AH31***		***AJ18–AJ31***	
AG24	PAD0[2]	AH21	PERR0*	AJ18	PAD0[24]
AG25	LED	AH22	PAD0[14]	AJ19	PAD0[21]
AG26	CBE0[4]*	AH23	PAD0[8]	AJ20	PAD0[16]
AG27	PAD0[59]	AH24	PAD0[3]	AJ21	STOP0*
AG28	PAD0[43]	AH25	ENUM*	AJ22	PAD0[15]
AG29	PAD0[44]	AH26	P64EN*	AJ23	PAD0[9]
AG30	PAD0[45]	AH27	PAD0[60]	AJ24	PAD0[4]
AG31	PAD0[46]	AH28	PAD0[55]	AJ25	REQ640*
AH01–AH20		AH29	PAD0[47]	AJ26	CBE0[5]*
AH01	PAD1[18]	AH30	PAD0[48]	AJ27	PAD0[61]
AH02	PAD1[19]	AH31	PAD0[49]	AJ28	PAD0[56]
AH03	PAD1[20]	***AJ01–AJ17***		AJ29	PAD0[52]
AH04	PAD1[21]	AJ01	CBE1[2]*	AJ30	PAD0[51]
AH05	CBE1[1]*	AJ02	PAD1[16]	AJ31	PAD0[50]
AH06	PAD1[12]	AJ03	PAD1[17]	***AK02–AK15***	
AH07	CBE1[0]*	AJ04	STOP1*	AK02	VCC 3.3
AH08	PAD1[2]	AJ05	PAR1	AK03	IRDY1*
AH09	CBE1[6]*	AJ06	PAD1[13]	AK04	DEVSEL1*
AH10	PAD1[61]	AJ07	PAD1[8]	AK05	SERR1*
AH11	PAD1[55]	AJ08	PAD1[3]	AK06	PAD1[14]
AH12	PAD1[50]	AJ09	CBE1[7]*	AK07	PAD1[9]
AH13	PAD1[45]	AJ10	PAD1[62]	AK08	PAD1[4]
AH14	PAD1[40]	AJ11	PAD1[56]	AK09	REQ641*
AH15	PAD1[34]	AJ12	PAD1[51]	AK10	PAD1[63]
AH16	CLK0	AJ13	PAD1[46]	AK11	PAD1[57]
AH17	PAD0[28]	AJ14	PAD1[41]	AK12	PAD1[52]
AH18	CBE0[3]*	AJ15	PAD1[35]	AK13	PAD1[47]
AH19	PAD0[20]	AJ16	Rst0*	AK14	PAD1[42]
AH20	CBE0[2]*	AJ17	PAD0[29]	AK15	PAD1[36]

Table 694: GT–64240A Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
AK16–AK30		***AL03–AL17***		***AL18–AL29***	
AK16	INT0*	AL03	FRAME1*	AL18	PAD0[26]
AK17	PAD0[30]	AL04	TRDY1*	AL19	PAD0[23]
AK18	PAD0[25]	AL05	PERR1*	AL20	PAD0[18]
AK19	PAD0[22]	AL06	PAD1[15]	AL21	TRDY0*
AK20	PAD0[17]	AL07	PAD1[10]	AL22	PAR0
AK21	DEVSEL0*	AL08	PAD1[5]	AL23	PAD0[11]
AK22	CBE0[1]*	AL09	ACK641*	AL24	PAD0[6]
AK23	PAD0[10]	AL10	PAR641	AL25	PAD0[0]
AK24	PAD0[5]	AL11	PAD1[58]	AL26	CBE0[7]*
AK25	ACK640*	AL12	PAD1[53]	AL27	PAD0[63]
AK26	CBE0[6]*	AL13	PAD1[48]	AL28	PAD0[58]
AK27	PAD0[62]	AL14	PAD1[43]	AL29	PAD0[54]
AK28	PAD0[57]	AL15	PAD1[37]		
AK29	PAD0[53]	AL16	PAD1[32]		
AK30	VCC 3.3	AL17	PAD0[31]		

Figure 88: GT–64240A Pinout Map (top view, left section)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
A			BAdr[0]	Wr[0]	SData[1]	SData[4]	SData[7]	SData[10]	SData[13]	ECC[0]	SCAS*	SCS[1]*	DAdr[3]	DAdr[8]	SDClkOut	A
B		VCC 3.3	BAdr[1]	Wr[1]	SData[32]	SData[35]	SData[38]	SData[41]	SData[44]	SData[47]	DWr*	SCS[0]*	DAdr[2]	DAdr[7]	DAdr[11]	B
C	AD[0]	ALE	BAdr[2]	Wr[2]	SData[0]	SData[3]	SData[6]	SData[9]	SData[12]	SData[15]	ECC[5]	SDQM*[5]	DAdr[1]	DAdr[6]	BankSel[1]	C
D	AD[3]	AD[2]	AD[1]	Wr[3]	Ready*	SData[34]	SData[37]	SData[40]	SData[43]	SData[46]	ECC[1]	SDQM[1]*	DAdr[0]	DAdr[5]	BankSel[0]	D
E	AD[7]	AD[6]	AD[5]	AD[4]	CSTiming*	SData[2]	SData[5]	SData[8]	SData[11]	SData[14]	ECC[4]	SDQM*[4]	SRAS*	DAdr[4]	DAdr[10]	E
F	AD[12]	AD[11]	AD[10]	AD[9]	AD[8]	SData[33]	SData[36]	SData[39]	SData[42]	SData[45]	VCC 3.3	SDQM[0]*	VCC 3.3	VCC Core	DAdr[9]	F
G	AD[18]	AD[17]	AD[16]	AD[15]	AD[14]	AD[13]	GND	VCC 3.3	GND	VCC 3.3						G
H	AD[24]	AD[23]	AD[22]	AD[21]	AD[20]	AD[19]	VCC 3.3									H
J	AD[30]	AD[29]	AD[28]	AD[27]	AD[26]	AD[25]	VCC 3.3									J
K	E0[3]	E0[2]	E0[1]	E0[0]	AD[31]	VCC 3.3	VCC 3.3									K
L	E0[8]	E0[7]	E0[6]	E0[5]	E0[4]	VCC 3.3										L
M	E0[13]	E0[12]	E0[11]	E0[10]	E0[9]	VCC 3.3										M
N	E[1]	E[0]	MDIO	MDC	E0[14]	VCC 3.3							GND	GND	GND	N
P	E[7]	E[6]	E[5]	E[4]	E[3]	E[2]							GND	GND	GND	P
R	E[12]	E[11]	E[10]	E[9]	E[8]	GND							GND	GND	GND	R
T	S0[3]	S0[2]	S0[1]	S0[0]	E[14]	E[13]							GND	GND	GND	T
U	S[1]	S[0]	S0[6]	S0[5]	S0[4]	VCC Core							GND	GND	GND	U
V	I2CSCK	S[6]	S[5]	S[4]	S[3]	S[2]							GND	GND	GND	V
W	MPP[3]	MPP[2]	MPP[1]	MPP[0]	I2CSDA	VCC Core							GND	GND	GND	W
Y	MPP[8]	MPP[7]	MPP[6]	MPP[5]	MPP[4]	VCC 3.3										Y
AA	MPP[13]	MPP[12]	MPP[11]	MPP[10]	MPP[9]	VCC 3.3										AA
AB	MPP[19]	MPP[18]	MPP[17]	MPP[16]	MPP[15]	MPP[14]	VCC 3.3									AB
AC	MPP[25]	MPP[24]	MPP[23]	MPP[22]	MPP[21]	MPP[20]	VCC 3.3									AC
AD	MPP[31]	MPP[30]	MPP[29]	MPP[28]	MPP[27]	MPP[26]	VCC 3.3									AD
AE	PAD[31]	REQ*	GNT*	CLK1	Rst*	INT*	GND	VREF1	VCC Core	VCC Core						AE
AF	PAD[25]	PAD[26]	PAD[27]	PAD[28]	PAD[29]	PAD[30]	PAD[6]	PAD[0]	CBE*[4]	PAD[59]	VCC 3.3	GND	VCC 3.3	PAD[38]	VCC 3.3	AF
AG	PAD[22]	PAD[23]	IDSEL1	CBE*[3]	PAD[24]	PAD[11]	PAD[7]	PAD[1]	CBE[5]*	PAD[60]	PAD[54]	PAD[49]	PAD[44]	PAD[39]	PAD[33]	AG
AH	PAD[18]	PAD[19]	PAD[20]	PAD[21]	CBE*[1]*	PAD[12]	CBE[0]*	PAD[2]	CBE[6]*	PAD[61]	PAD[55]	PAD[50]	PAD[45]	PAD[40]	PAD[34]	AH
AJ	CBE*[2]	PAD[16]	PAD[17]	STOP*	PAR1	PAD[13]	PAD[8]	PAD[3]	CBE*[7]	PAD[62]	PAD[56]	PAD[51]	PAD[46]	PAD[41]	PAD[35]	AJ
AK		VCC 3.3	IRDY*	DEVSEL*	SERR*	PAD[14]	PAD[9]	PAD[4]	REQ64*	PAD[63]	PAD[57]	PAD[52]	PAD[47]	PAD[42]	PAD[36]	AK
AL			FRAME*	TRDY*	PERR*	PAD[15]	PAD[10]	PAD[5]	ACK64*	PAR641	PAD[58]	PAD[53]	PAD[48]	PAD[43]	PAD[37]	AL
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

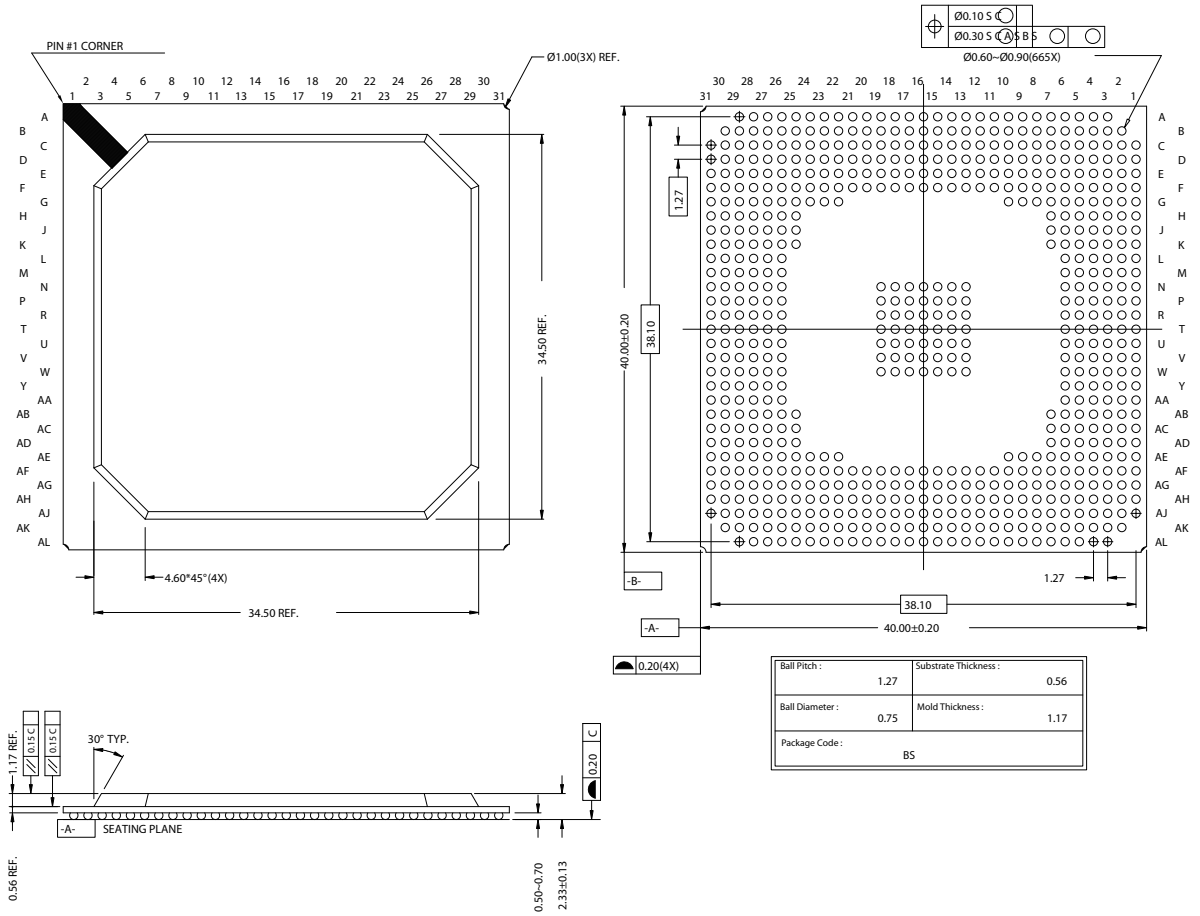
NOTE: VCC=VDD, GND=VSS, NC=Not Connected

Figure 89: GT–64240A Pinout Map (top view, right section)

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
A	SDQM[6]*	ECC[7]	SData[18]	SData[52]	SData[23]	SData[57]	SData[28]	SData[31]	SData[63]	NC	PReq*	NC	NC	SysRdyOut*			A
B	SDQM[2]*	ECC[3]	SData[49]	SData[20]	SData[54]	SData[25]	SData[59]	SData[62]	TDI	NC	NC	NC	NC	NC	VCC 2.5		B
C	SCS*[3]	ECC[6]	SData[17]	SData[51]	SData[22]	SData[56]	SData[27]	SData[30]	TMS	PAck*	NC	CPUInt*	NC	TcMatch	NC	ValidIn*	C
D	SCS[2]*	ECC[2]	SData[48]	SData[19]	SData[53]	SData[24]	SData[58]	SData[61]	TRST	SysRst*	NC	NC	NC	NC	NC	ValidOut*	D
E	DAdr[12]	SDQM*[7]	SData[16]	SData[50]	SData[21]	SData[55]	SData[26]	SData[29]	TCK	SysClk	RspSwap*	Release*	TcWord[1]	TcWord[0]	NC	SysCmd[0]	E
F	VCC Core	SDQM*[3]	GND	VCC 3.3	VCC 3.3	VCC 3.3	VCC 3.3	SData[60]	TClk	JTDO	SysCmd[1]	NC	SysCmd[5]	SysCmd[4]	SysCmd[3]	SysCmd[2]	F
G							VCC 3.3	GND	VCC 3.3	AGND	SysCmd[8]	SysCmd[7]	NC	SysCmd[6]	TcTCE*	NC	G
H										AVCC	TcDOE*	SysRdyIn[2]*	NC	NC	SysRdyIn[1]*	SysRdyIn[0]*	H
J										VCC 2.5	NC	NC	NC	NC	NC	NC	J
K										VCC 2.5	NC	NC	NC	NC	NC	NC	K
L											VCC 2.5	NC	SysADC[7]	SysAD[56]	SysAD[55]	SysAD[26]	L
M											SysAD[31]	NC	SysADC[5]	SysADC[3]	SysAD[22]	SysAD[24]	M
N	GND	GND	GND	GND							VCC Core	SysAD[28]	SysAD[29]	NC	SysADC[2]	SysAD[4]	N
P	GND	GND	GND	GND							VCC Core	SysAD[18]	SysAD[21]	SysAD[52]	SysAD[54]	SysAD[37]	P
R	GND	GND	GND	GND							VCC 2.5	SysAD[30]	SysADC[6]	SysAD[7]	SysAD[50]	SysAD[42]	R
T	GND	GND	GND	GND							SysAD[32]	SysAD[27]	SysAD[3]	SysADC[0]	SysAD[17]	SysAD[12]	T
U	GND	GND	GND	GND							VCC 2.5	SysAD[60]	SysAD[44]	SysADC[1]	SysAD[25]	SysAD[8]	U
V	GND	GND	GND	GND							SysAD[15]	SysAD[19]	SysAD[11]	SysAD[57]	SysAD[58]	SysAD[46]	V
W	GND	GND	GND	GND							GND	SysAD[43]	SysAD[41]	SysAD[39]	SysAD[33]	SysAD[1]	W
Y											VCC Core	SysAD[23]	SysAD[5]	SysAD[20]	SysAD[10]	SysAD[61]	Y
AA											VCC 2.5	SysAD[14]	SysAD[45]	SysAD[9]	SysAD[48]	SysAD[47]	AA
AB										VCC 2.5	VCC 2.5	SysAD[36]	SysAD[35]	SysAD[2]	SysADC[4]	SysAD[6]	AB
AC										VCC 2.5	SysAD[16]	SysAD[63]	SysAD[62]	SysAD[59]	SysAD[51]	SysAD[13]	AC
AD										VCC Core	SysAD[53]	SysAD[49]	SysAD[40]	SysAD[38]	SysAD[34]	SysAD[0]	AD
AE							VCC 3.3	VCC 3.3	VCC Core	VCC 3.3	PAD0[32]	PAD0[33]	PAD0[34]	PAD0[35]	PAD0[36]	PAD0[37]	AE
AF	REQ0*	VREF0	VCC 3.3	GND	IRDY0*	VCC 3.3	PAD0[12]	PAD0[7]	PAD0[1]	HS0	PAR640	PAD0[38]	PAD0[39]	PAD0[40]	PAD0[41]	PAD0[42]	AF
AG	GNT0*	PAD0[27]	IDSEL0	PAD0[19]	Frame0*	SERR0*	PAD0[13]	CBE0[0]*	PAD0[2]	LED	CBE0[4]*	PAD0[59]	PAD0[43]	PAD0[44]	PAD0[45]	PAD0[46]	AG
AH	CLK0	PAD0[28]	CBE0[3]*	PAD0[20]	CBE0[2]*	PERR0*	PAD0[14]	PAD0[8]	PAD0[3]	ENUM*	P64EN	PAD0[60]	PAD0[55]	PAD0[47]	PAD0[48]	PAD0[49]	AH
AJ	Rst0*	PAD0[29]	PAD0[24]	PAD0[21]	PAD0[16]	STOP0*	PAD0[15]	PAD0[9]	PAD0[4]	REQ640*	CBE0[5]*	PAD0[61]	PAD0[56]	PAD0[52]	PAD0[51]	PAD0[50]	AJ
AK	INT0*	PAD0[30]	PAD0[25]	PAD0[22]	PAD0[17]	DEVSEL0*	CBE0[1]*	PAD0[10]	PAD0[5]	ACK640*	CBE0[6]*	PAD0[62]	PAD0[57]	PAD0[53]	VCC 3.3		AK
AL	PAD[32]	PAD0[31]	PAD0[26]	PAD0[23]	PAD0[18]	TRDY0*	PAR0	PAD0[11]	PAD0[6]	PAD0[0]	CBE0[7]*	PAD0[63]	PAD0[58]	PAD0[54]			AL
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	

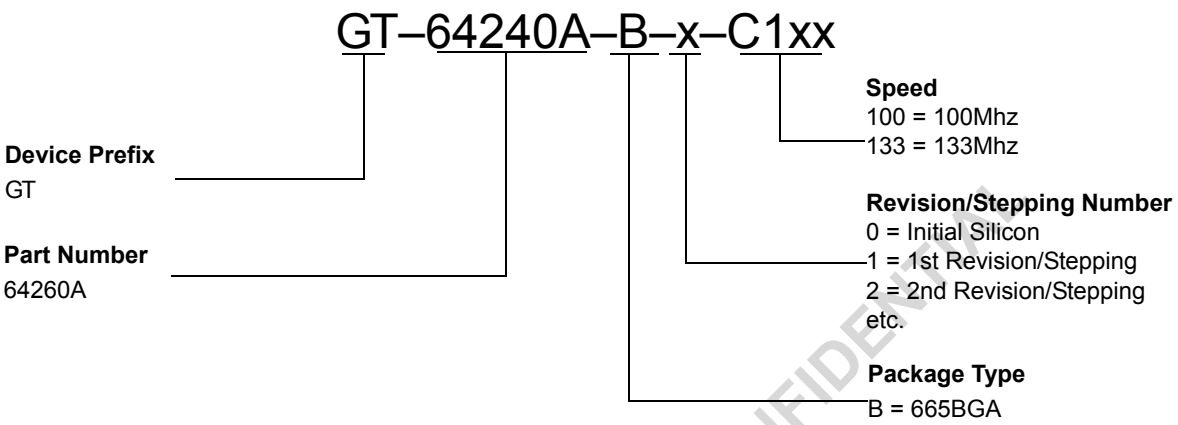
NOTE: VCC=VDD, GND=VSS, NC=Not Connected

29. 665 PBGA Package Mechanical Information



30. GT–64240A PART NUMBERING

Figure 90: Sample Part Number



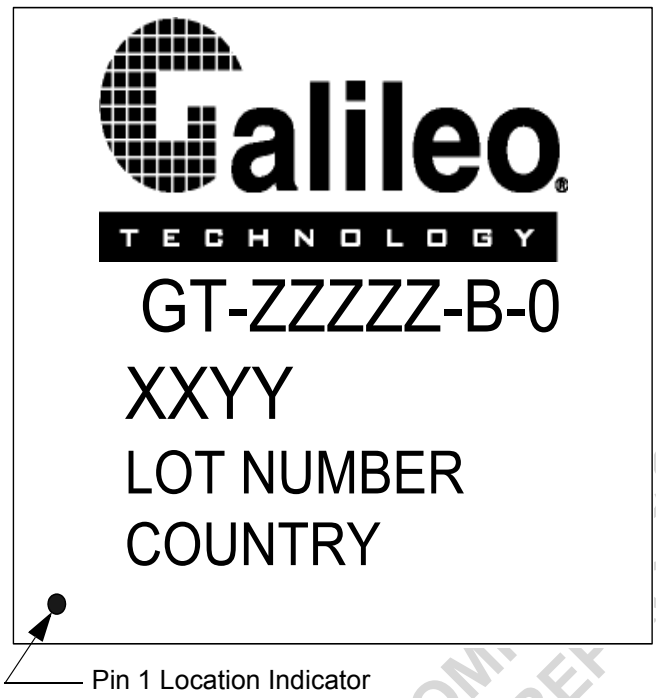
The part numbers for the GT–64240A is GT–64240A-B-x-C100 or GT–64240A-B-x-C133.

These part numbers indicates that this is the commercial temperature grade, 100MHz or 133MHz version.

These are the only valid part numbers that can be used when ordering the GT–64240A.

31. GT–64240A Part Marking and Pin Location

Figure 91: Package Markings and Pin 1 Location



Nomenclature:

Galileo Part Number:
GT-ZZZZZ

Packaging Designator:
P = PQFP
B = BGA
M = MQUAD
L = PLCC

Metal Mask Revision:
0 = First silicon (Mask A)
1 = Mask B
2 = Mask C
3 = Mask D
4 = Mask E
Etc.

Manufacturing Running Date:
XX= Year
YY= Workweek

32. Revision History

Table 695: Revision History

Document Type	Revision	Date
Preliminary Datasheet	-	August 7, 2001
NOTE: Recently, there was a change in the revision numbeing system used by Marvell Technology. This revision, marked "Rev. -", is the latest revision of the datasheet. First revision. 5. Revised DC electrical characteristics in Table 690, "DC Electrical Characteristics Over Operating Range," on page 532. New parameters for I_{ih}, I_{il}, I_{ozh}, and I_{ozl},		
Preliminary Datasheet	A	August 29, 2001
Change to Revision A. No substansive specification changes.		