GT–64242

System Controller for MIPS Processors

FEATURES

Integrated system controller with PCI interface 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 100MHz 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–64242 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–64242 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 100MHz 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–64242 devices share the same SDRAM.

Device controller:

- A dedicated 32-bit multiplexed address/data bus (separated from SDRAM bus).
- Up to 100MHz 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.

Four 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)

One high-performance PCI 2.2 compliant interface.

- PCI bus speed of up to 66MHz with zero wait states.
- 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.

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 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.

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

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

Serial ROM initialization through I^{2}C 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.

Disclaimer

Preliminary or Advanced Information: This document provides preliminary or advanced information about the product described. All specifications described herein are based on design goals only. **Do not use for final design.** Visit Marvell's web site at www.marvell.com or call 1-866-674-7253 for the latest information on Marvell products.

DISCLAIMER

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose, without the express written permission of Marvell. Marvell retains the right to make changes to this document at any time, without notice. Marvell makes no warranty of any kind, expressed or implied, with regard to any information contained in this document, including, but not limited to, the implied warranties of merchantability or fitness for any particular purpose. Further, Marvell does not warrant the accuracy or completeness of the information, text, graphics, or other items contained within this document. Marvell makes no commitment either to update or to keep current the information contained in this document. Marvell products are not designed for use in life-support equipment or applications that would cause a life-threatening situation if any such products failed. Do not use Marvell products in these types of equipment or applications. The user should contact Marvell to obtain the latest specifications before finalizing a product design.

Marvell assumes no responsibility, either for use of this product or for any infringements of patents and trademarks, or other rights of third parties resulting from its use. No license is granted under any patents, patent rights, or trademarks of Marvell. These products may include one or more optional functions. The user has the choice of implementing any particular optional function. Should the user choose to implement any of these optional functions, it is possible that the use could be subject to third party intellectual property rights. Marvell recommends that the user investigate whether third party intellectual property rights are relevant to the intended use of this product and obtain licenses as appropriate under relevant intellectual property rights.

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)

List of Tables

List of Tables (Continued)

List of Tables (Continued)

List of Tables (Continued)

List of Figures

List of Figures (Continued)

List of Figures (Continued)

1. OVERVIEW

The GT–64242 provides a single-chip solution for designers building systems for a MIPS64-bit bus CPU. The GT–64242 architecture supports several system implementations for different applications.

The GT–64242 has a four bus architecture:

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

The four 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 bufferor a PCI device can write into an on-chip FIFO, all simultaneously.

1.1 CPU Bus Interface

The GT–64242 supports MIPS bus protocol. With a maximum frequency of 100MHz, the CPU can transfer in excess of 1 Gbytes/sec.

NOTE: The QED RM7000C CPU is now supported in TTL mode, only.

The GT–64242 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–64242 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–64242 drives the SDRAM read data first and then the PCI read data that arrives later.

The CPU can connect with up to four GT–64242 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–64242 supports CPU address remapping to the PCI interface. It also supports access, write, and caching protection, per user specified address ranges.

The GT–64242 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 44.

1.2 SDRAM Interface

The GT–64242 SDRAM controller supports SDRAM and registered SDRAM. It supports 16/64/128/256/512 Mbit SDRAMs.

The GT–64242 works at frequencies up to 100MHz, and can address up to 4GBytes.

Up to four banks of SDRAM may be connected to The GT–64242.

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

The GT–64242 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–64242 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 84.

1.3 Device Interface

The GT–64242 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–64242 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 122.

1.4 PCI Interface

The GT–64242 interfaces directly with one 64-bit PCI busses, operating at a maximum frequency of 66MHz. The PCI interface can act as a master initiating a PCI bus transaction or as a target responding to a PCI bus transaction.

The GT–64242 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–64242 acts as a target when the PCI device initiates a memory access. 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.

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–64242, 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–64242 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–64242 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–64242 also supports PCI Hot-Plug and CompactPCI Hot-Swap ready.

The GT–64242 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 135.

1.5 DMA Engines

The GT–64242 incorporates four 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 using one FIFO, while channel2 transfers data from external system memory 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 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.

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. Four End of Transfer pins act as inputs to the GT–64242 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 234.

1.6 Data Integrity

The GT–64242 supports full data integrity on its different interfaces.

The GT–64242 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–64242 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–64242 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–64242 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 115.

2. Pin Information

Figure 1 shows the GT–64242 interfaces.

Figure 1: GT–64242 Interfaces

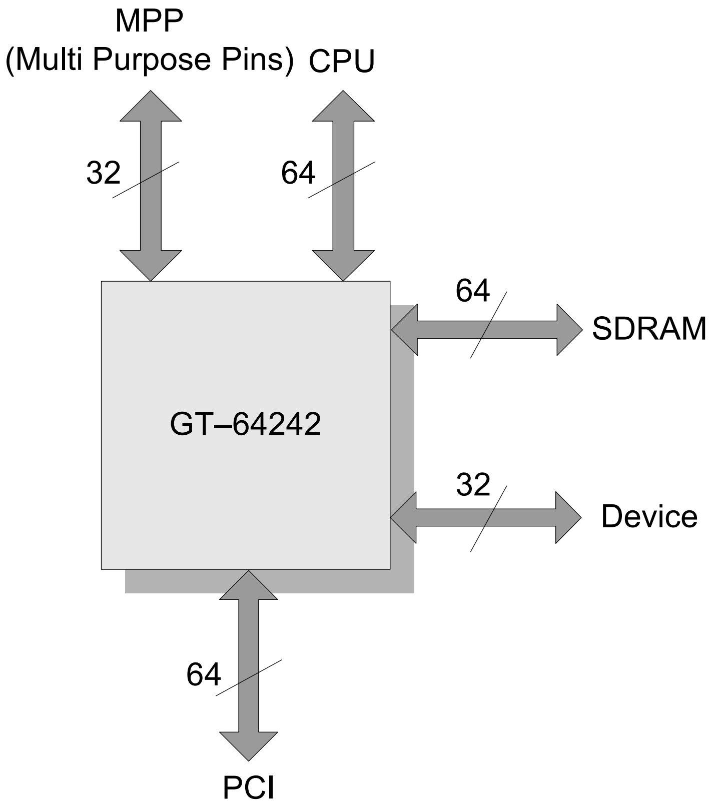


Table 1 lists the conventions that apply to I/O or O type pins described in the Pin Assignment tables:

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–64242 units internal clock (up to 100MHz). 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 22.4 “PLL Power Filter Circuit” on page 324.
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–64242. 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–64242 during read response data phase.
[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–64242 during read response data phase. **NOTE:** – SysADC is valid on data cycles only. – If not using CPU parity, a pull-up is required.
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–64242 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–64242 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–64242 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–64242 configurations, SysRdyOut*, of the boot GT–64242, is connected to the CPU RdRdy* and WrRdy* inputs. The SysRdyOut* outputs, of the other GT–64242 devices, are connected to SysRdyIn[2:0] of the boot GT–64242 device.
SysRdyIn[2:0]* H27, H30, H31	I	Read/Write Ready Input	**NOTE:** Relevant only in multi-GT–64242 configurations. In a single GT configuration, connect to GND. SysRdyIn of the boot GT–64242 device is connected to SysRdyOut of all other GT–64242 devices. • SysRdyIn[0]* of all the GT–64242 devices, except for the boot device, are connected to the boot GT–64242 device's SysRdyOut* signal, which is also the CPU RdRdy* and WrRdy* input. • SysRdyIn[2:1]* of all the GT–64242 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–64242 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–64242 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–64242 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–64242 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–64242 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–64242 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 RAMon 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 Interface Pin Assignments

Pin Name/ Ball #	Type	Full Name	Description
Clk AH16	I	PCI Clock	The PCI clock range is between 0 and 66MHz. PClk0 is completely independent of SysClk and TClk.
Rst* AJ16	I	PCI Reset	Dedicated reset signal for PCI interface. When in the reset state, all PCI output pins are put into tristate and all open drain signals are floated.

Table 4: PCI Bus Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
VREF AF17	I	PCI Voltage Reference	This pin must be connected directly to the 3.3V or the 5V power plane depending on which voltage level PCI supports.
PAD[63:0]	t/s I/O	PCI Address/ Data	-bit PCI 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, PAD[63:32] requires a pull-up. When configured as a 32-bit bus, the GT–64242 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			
CBE[7:0]* AL26, AK26, AJ26, AG26, AH18, AH20, AK22, AG23	t/s I/O	PCI 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, CBE[7:4] requires a pull-up. When configured as a 32-bit bus, the GT–64242 drives these pins; a pull-up is not required.
PAR AL22	t/s I/O	PCI Parity (low)	Even parity calculated for PAD[31:0] and CBE[3:0]. Driven by transaction master for address phase and write data phase. Driven by target for read data phase.
FRAME* AG20	s/t/s I/O	PCI Frame	Asserted by the transaction master to indicate the beginning of a transaction. The master de-asserts FRAME* to indicate that the next data phase is the final data phase transaction.
IRDY* AF20	s/t/s I/O	PCI 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 TRDY* and IRDY* are asserted.

Table 4: PCI Bus Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
DEVSEL* AK21	s/t/s I/O	PCI Device Select	Asserted by the target of the current access. As a master, the GT–64242 expects the target to assert DEVSEL* within five bus cycles. If the target does not assert DEVSEL* within the required bus cycles, the GT–64242 aborts the cycle. As a target, the GT–64242 asserts DEVSEL* in a medium speed; two cycles after the assertion of FRAME*.
TRDY* AL21	s/t/s I/O	PCI 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 TRDY* and IRDY* are asserted.
STOP* AJ21	s/t/s I/O	PCI Stop	Asserted by target to indicate transaction termination. Used by a target device to generate a Retry, Disconnect, or Target Abort termination signal.
IDSEL AG18	I	PCI Initialization Device Select	Asserted to act as a target device chip select during PCI configuration transactions.
REQ64* AJ25	s/t/s I/O	PCI 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 FRAME* timing. **NOTE:** A 64-bit transaction occurs when REQ64* and ACK64* are asserted.
ACK64* AK25	s/t/s I/O	PCI Acknowledge 64-bit Transfer	Asserted by the target in response to REQ64* to indicate it accepts a 64-bit bus width transaction. ACK64* timing is the same as DEVSEL* timing. **NOTE:** A 64-bit transaction occurs when REQ64* and ACK64* are asserted.
PAR64 AF26	t/s I/O	PCI Parity (high)	In cases of a 64-bit PCI transaction, even parity is calculated for PAD[63:32] and CBE[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–64242 drives this pin; a pull-up is not required.
REQ* AF16	t/s O	PCI Bus Request	If using an external PCI arbiter, asserted by the GT–64242 PCI master to indicate it requires PCI bus mastership to initiate a new transaction. If using the internal PCI arbiter, leave unconnected.

Table 4: PCI Bus Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
GNT* AG16	I	PCI Bus Grant	If using an external PCI arbiter, asserted to indicates to the GT–64242 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–64242 internal PCI arbiter, a pull-up is required.
PERR* AH21	s/t/s I/O	PCI 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.
SERR* AG21	o/d O	PCI System Error	Asserted when a serious system error (not necessarily a PCI error) is detected.
INT* AK16	o/d O	PCI Interrupt Request	Asserted by the GT–64242 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–64242 during hot swap insertion or removal.
LED AG25	t/s O	Compact PCI Hot Swap LED	Driven by the GT–64242 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.
P64EN AH26	I	Compact PCI Hot Swap 64-bit PCI Enable	The GT–64242 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: SDRAM Interface Pin Assignments

Pin Name/ Ball #	Type	Full Name	Description
SDClkOut A15	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 100.
SRAS* E13	t/s O	SDRAM Row Address Select	Asserted by the GT–64242 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–64242 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–64242 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–64242 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–64242 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–64242 to select a specific SDRAM physical bank. **NOTE:** If UMA enabled, acts as s/t/s pin.
SDQM[7:0]* E17, A16, C12, E12, F17, B16, D12, F12	t/s O	SDRAM Data Mask	Asserted by the GT–64242 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–64242 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			

Table 5: SDRAM Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
ECC[7:0] A17, C17, C11, E11, B17, D17, D11, A10	t/s I/O	SDRAM ECC byte	Driven by the GT–64242 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 6: 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–64242 during burst read/write transactions to a device. **NOTE:** The GT–64242 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–64242 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.
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.

Table 6: Device Interface Pin Assignments (Continued)

Pin Name/ Ball #	Type	Full Name	Description
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 7: 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 8: I^{2}C Interface Pin Assignments

Pin Name/ Ball #	Type	Full Name	Description
I2CSCK V01	o/d I/O	I^{2}C Clock	I^{2}C serial clock. Serves as output when the GT–64242 acts as an I^{2}C master. Serves as input when the GT–64242 acts as an I^{2}C slave.
I2CSDA W05	o/d I/O	I^{2}C Serial Data	Address or write data driven by the I^{2}C master or read response data driven by the I^{2}C slave.
I^{2}C Interface Pin Count: 2			

Table 9: JTAG Interface Pin Assignments

Pin Name/ Ball #	Type	Full Name	Description
TCK E24	I	JTAG Clock	Clock input for the GT–64242 JTAG controller. **NOTE:** A pull-down is required.
TRST D24	I	JTAG Reset	When asserted, resets the GT–64242 JTAG controller. **NOTE:** A pull-down is required.
TMS C24	I	JTAG Mode Select	Controls the GT–64242 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–64242 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 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 15. “MPP Multiplexing” on page 275 for more information.

Table 10 summarizes the MPP pins functionality.

Table 10: 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–64242 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.
TCTCnt[7:0]	O	Timer/Counter[7:0] Terminal Count	Terminal count output. **NOTE:** One pin per timer/counter..
GPP[31:0]	I/O	General Purpose Port [31:0]	General purpose input/output port, see Section 14. “General Purpose Port” on page 272 for more information.
InitAct	O	Initialization Active	Driven to 1 for the entire serial ROM initialization period.
PME*	o/d O	PCI Power Management Event	If PME is enabled, asserted by the GT–64242 upon CPU request.
MREQ*	I/O	UMA Request	SDRAM bus request asserted by a UMA slave device.

Table 10: MPP Pins Functionality (Continued)

Pin Name	Type	Functionality	Description
MGNT*	I/O	UMA Grant	Asserted by the UMA master in response to MREQ* to indicate bus mastership to the UMA slave device.
PCIReq[5:0]*	I	PCI Request[5:0]	External PCI bus requests when the GT–64242 PCI bus arbiter is enabled.
PCIGnt[5:0]*	O	PCI Grant[5:0]	Bus grant to external PCI masters when the GT–64242 PCI 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.
Int[3:0]*	O	CPI Interrupt[3:0]	Four CPU interrupt pins.
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–64242 has a fully programmable address map.

two address spaces exist:

- The CPU address space.
- The PCI address space.

The GT–64242 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–64242 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–64242 implements one level decoding that maps directly to the target device.

3.1 CPU Address Decoding

The CPU interface address decoding map consists of 15 address windows for the different devices, as shown in Table 11.

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

Table 11: CPU Interface Address Decoder Mappings

CPU Decoder	Associated Target
SCS[3:0]*	SDRAM chip selects.
CS[3:0]*, BootCS*	Devices chip selects.
PCI I/O	PCI I/O space.
PCI Mem 0/1/2/3	PCI Memory space.
Internal	GT–64242 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 11 address windows for the different devices, as shown in Table 12.

Table 12: PCI Interface Address Decoder Mappings

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

In addition, PCI slave supports 12 more address windows for 64-bit addressing (using PCI Dual Access Cycle [DAC] transactions), as shown in Table 13.

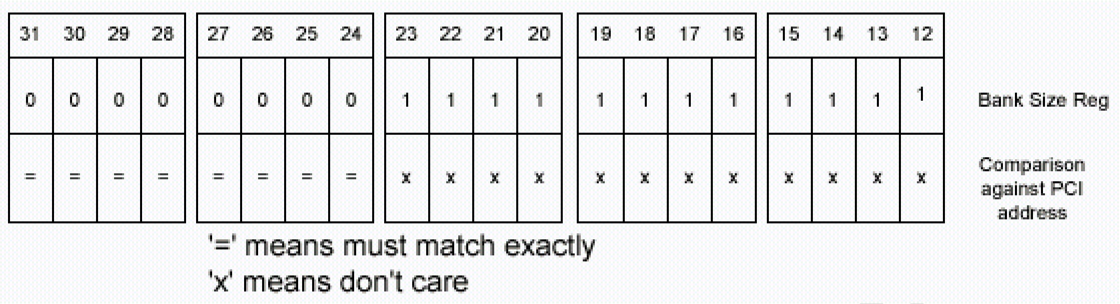
Table 13: PCI Interface 64-bit Addressing Address Decoder Mappings

PCI_0 Slave Decoder	Associated Target
DAC SCS[3:0]*	SDRAM chip selects.
DAC CS[3:0]*, DAC BootCS*	Devices chip selects.

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 8.5.4 “64-bit Addressing BARs” on page 142 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)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	Bank Size Reg
=	=	=	=	=	=	=	=	x	x	x	x	x	x	x	x	x	x	x	x	Comparison against PCI address

'=' means must match exactly
'x' means don't care

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–64242 does not respond (no DEVSEL* asserted) to a PCI transaction that it's address match the BARs address space, see Table 172 on page 170.

3.4 IDMA Unit Address Decoding

The IDMA 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 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 33 for details.

3.5 Address Space Decoding Errors

When the CPU tries to access an unmapped address:

- The GT–64242 latches the address into the CPU Error Address registers, see Section 4.15.5 "CPU Error Report Registers" on page 80.
- The CPU AddrErr bit [0] in the CPU Error Cause register is set, see Table 94 on page 82.
- 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–64242 BARs results in no response at all from the GT–64242, since the address is targeted to some other target device on the PCI bus.

When an IDMA accesses an unmapped address:

- The GT–64242 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).

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

Table 14: 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 14: 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 64 Kbytes
PCI_0 Mem0	0x1200.0000 to 0x13ff.ffff 32 Mbytes
PCI Mem1	0xf200.0000 to 0xf3ff.ffff 32 Megabytes
PCI Mem2	0xf400.0000 to 0xf5ff.ffff 32 Mbyte
PCI Mem3	0xf600.0000 to 0xf7ff.ffff 32 Mbyte
PCI I/O	0x1000.0000 to 0x11ff.ffff 32 Mbytes

Table 15 shows the default PCI memory map that is valid following RESET.

Table 15: PCI 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 15: PCI Default Address Mapping

Decoder	Address Range
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

Table 16 shows the default 64-bit addressing PCI memory map that is valid following RESET.

Table 16: 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

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–64242, programming the address decoding registers might be problematic. Also, The software must confirm that the programing actually happened, before it attempts to access GT–64242 with an address that matches the new programed decoder.

3.7.1 PCI Programming of Address Decoders

PCI accesses to the GT–64242 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–64242 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–64242 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 36 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–64242 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–64242 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

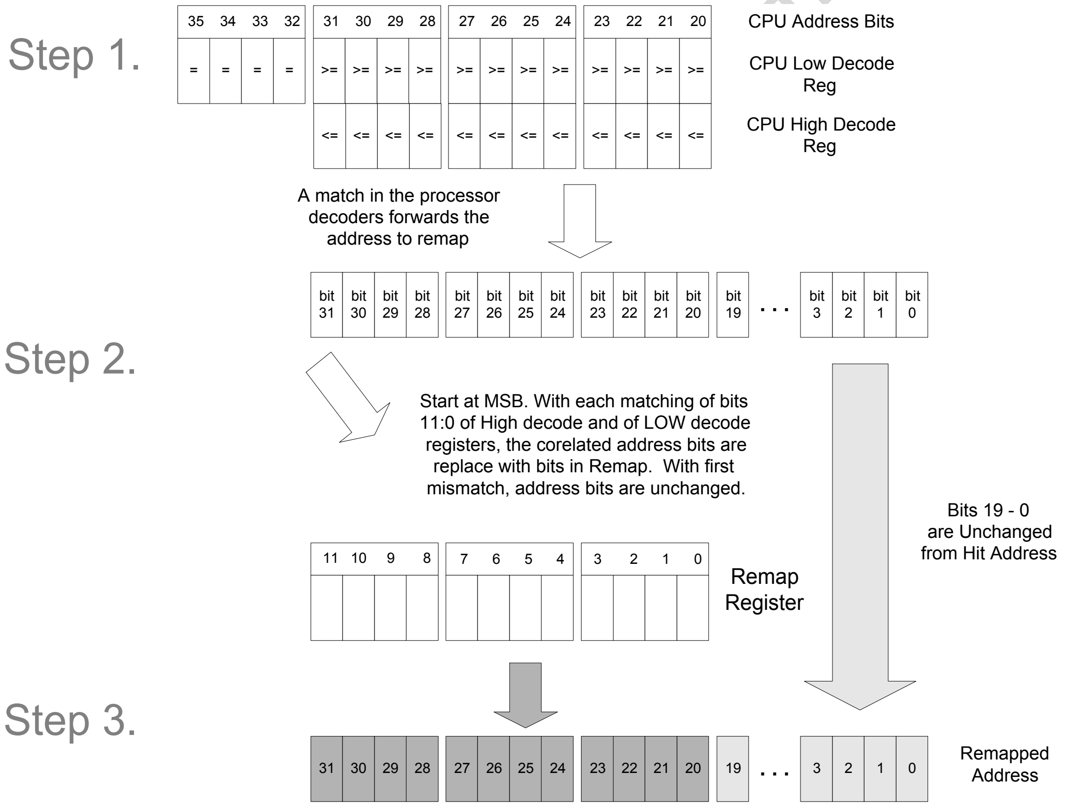
Each of the CPU to PCI address windows has a Remap Register associated with it.

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


Step 1.
35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20
= = = = >= >= >= >= >= >= >= >= >= >= >= >=
<= <= <= <= <= <= <= <= <= <= <= <=
CPU Address Bits
CPU Low Decode Reg
CPU High Decode Reg
A match in the processor decoders forwards the address to remap
bit 31 bit 30 bit 29 bit 28 bit 27 bit 26 bit 25 bit 24 bit 23 bit 22 bit 21 bit 20 bit 19 . . . bit 3 bit 2 bit 1 bit 0
Step 2.
Start at MSB. With each matching of bits 11:0 of High decode and of LOW decode registers, the corelated address bits are replace with bits in Remap. With first mismatch, address bits are unchanged.
Bits 19 - 0 are Unchanged from Hit Address
11 10 9 8 7 6 5 4 3 2 1 0
Remap Register
Step 3.
31 30 29 28 27 26 25 24 23 22 21 20 19 . . . 3 2 1 0
Remapped Address

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 17.

Table 17: 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 17, 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 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 135 for more details.

3.9 IDMA Unit Address Decoding Override

In default, the IDMA unit uses the CPU interface address decoding as in Section 3.4 “IDMA Unit Address Decoding” on page 36. However, the unit 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 interfaces, without going through the CPU interface address decoders.

For more details see, IDMA, Section 10.3 “IDMA Address Decoding” on page 236.

4. CPU INTERFACE

The GT–64242 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 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 33. This section summarizes CPU address decoding and emphasizes few details.

NOTE: For an exact list of CPU Address Decoding registers, seeTable 25 on page 62.

The CPU interface supports 15 address windows.

- Four for SDRAM chip selects.
- Five for device chip selects.
- Five for PCI interface (4 memory + one I/O).
- One for the GT–64242 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–64242 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–64242 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–64242 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–64242 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–64242 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 18: 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–64242 completion of a read transaction.
ValidIn*	The GT–64242 indication for driving valid read data and data identifier on SysAD and SysCmd busses.
SysRdyOut*	The GT–64242 indication that it is capable of accepting a new read or write transaction.
SysRdyIn[2:0]*	SysRdy* input used in a multi-GT–64242 configuration.
PRqst*	The CPU request from the GT–64242 for bus mastership so it can issue a new transaction.
PAck*	The GT–64242 bus acknowledge to CPU. The CPU may issue a new transaction on the next cycle.
RspSwap*	The GT–64242 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–64242 on L3 read hit.
TcTCE*	L3 cache Tag RAM chip enable. Sampled by the GT–64242 to identify L3 access.
TcWord[1:0]	L3 cache word index. Driven by the GT–64242 during L3 read miss.
CPUInt*	Level sensitive CPU interrupt asserted by the GT–64242.

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 19.

Table 19: 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 20: 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 21: 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 22: 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	-	-	-	-	-	-

Table 22: 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
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–64242 for completion of the read.

Two cycles after the Release* assertion, the GT–64242 starts driving the bus.

NOTE: There is a one turn-around cycle between the CPU drive and the GT–64242 drive.

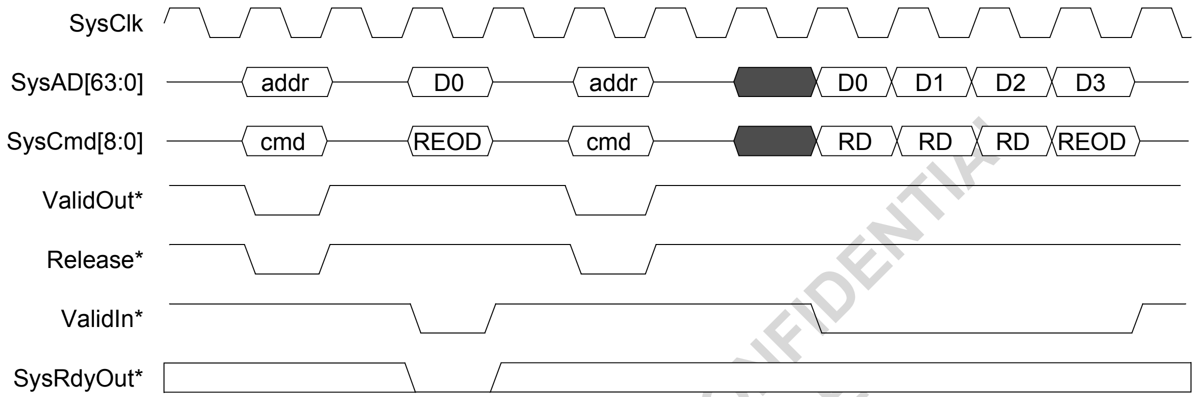
As soon as read data is available, the GT–64242 asserts ValidIn* and drives valid data on SysAD, and valid read response (mnemonic = RD) on SysCmd. On the last data, the GT–64242 drives last data identifier (mnemonic = REOD) on SysCmd.

On the clock cycle following REOD, the GT–64242 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–64242.

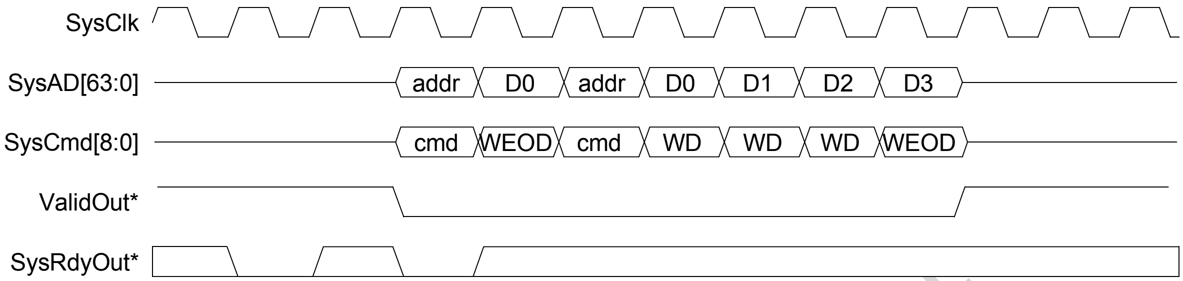
4.4.4 Write Protocol

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

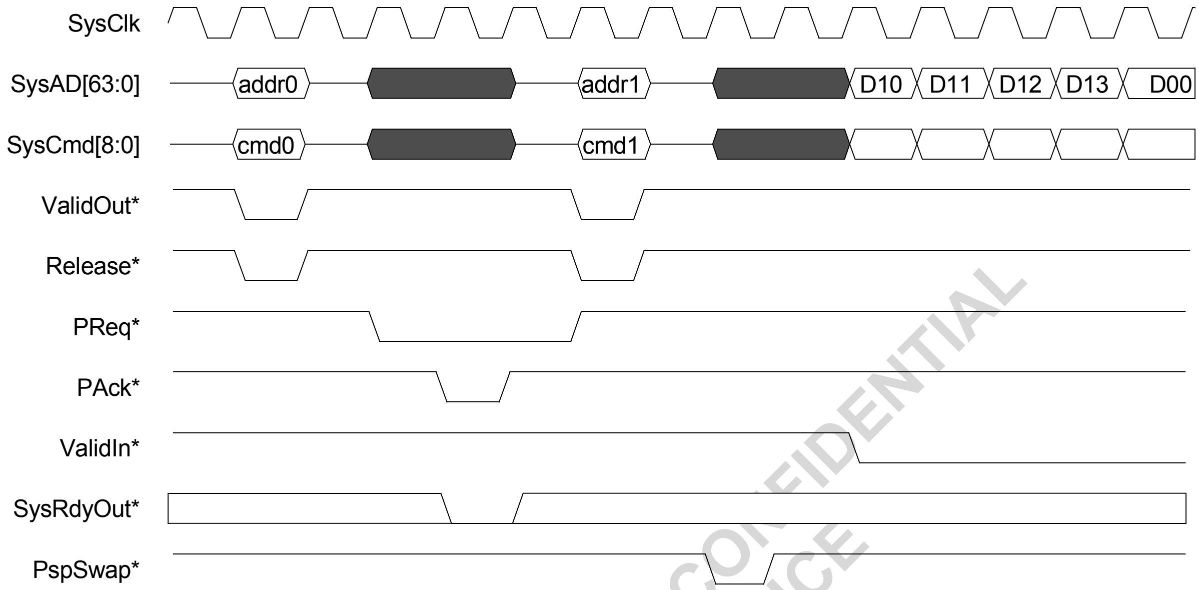
The GT–64242 supports the Non-Pendant mode of the RM7000 processor.

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–64242 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–64242 to complete both reads.

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

If the GT–64242 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–64242.

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 68 on page 73. 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 23 (DW0 is the least significant dword, DW3 is the most).

Table 23: 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–64242 to accept a new CPU write transaction it must have “room” in both the transactions queue and in the write data buffer.

The GT–64242 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–64242’s ability to accept a new transaction, there is a single SysRdyOut* signal driven by the GT–64242 rather than separate RdRdy* and WrRdy*. The GT–64242 SysRdyOut* output must be connected to both RdRdy* and WrRdy* inputs of the CPU.

The GT–64242 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–64242 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–64242 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–64242 supports only DDDD write pattern (64-bit data every clock cycle).

GT–64242 controls read data flow using the ValidIn* signal. If the CPU accesses a slow device, the GT–64242 keeps ValidIn* deasserted until read data arrives from the target device. In case of burst read from a slow device, the GT–64242 can deassert ValidIn* to create wait states between data beats.

4.8 MIPS CPU Cache Support

The GT–64242 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–64242. See the PMC-Sierra application note for more details.

The GT–64242 samples the TcMatch signal. In case of a CPU access that hits the L3 cache line (Tag RAM asserts TcMatch signal), the GT–64242 ignores the transaction. This enables the CPU to complete the transaction against L3 cache.

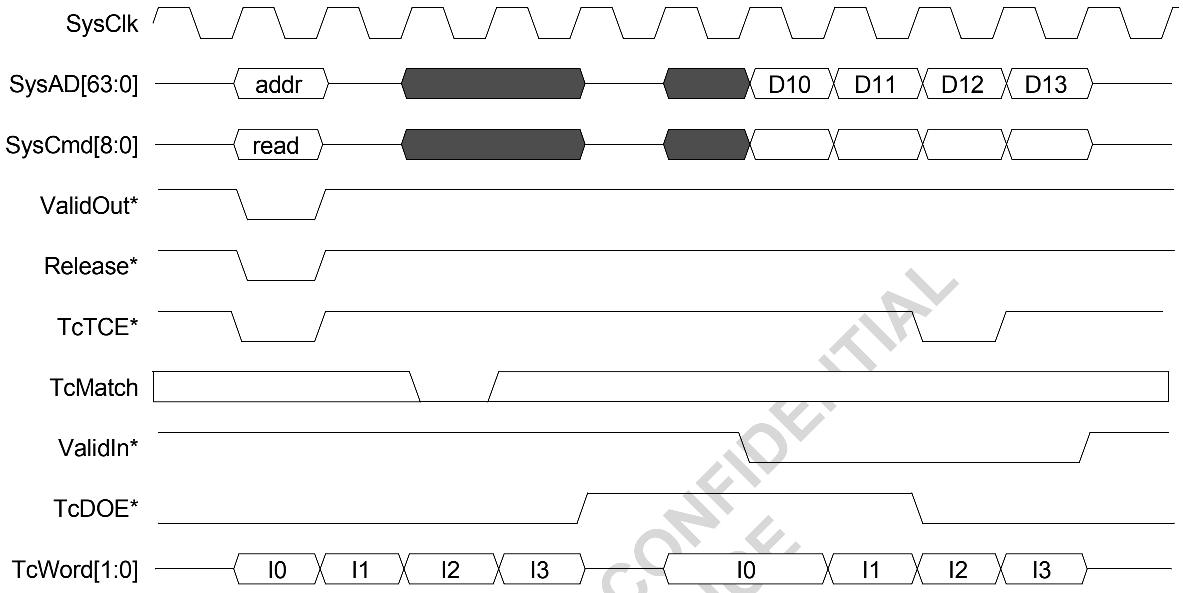
NOTE: Due to PMC-Sierra RM7000 errata, the RdOOO bit in the CPU Configuration register, see Table 68 on page 73, must be set to’0’ when using L3 cache.

The GT–64242 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–64242 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–64242 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–64242 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–64242.

4.9 Multi-GT Support

Up to four GT–64242 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–64242 is enabled through the reset configuration. See Section 20.1 “Pins Sample Configuration” on page 311.

NOTE: A Multi-GT–64242 configuration can also be used for the CPU to interface GT–64242 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–64242 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–64242 devices must be tied together to drive the CPU ValidIn* input.

All PAck* outputs from the GT–64242 devices must be tied together to drive the CPU PAck* input.

All TcDOE* outputs from the GT–64242 devices must be tied together to drive L3 TcDOE* input.

ValidIn* and TcDOE* are only driven by the target GT–64242. After last the ValidIn* cycle, the GT–64242 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.

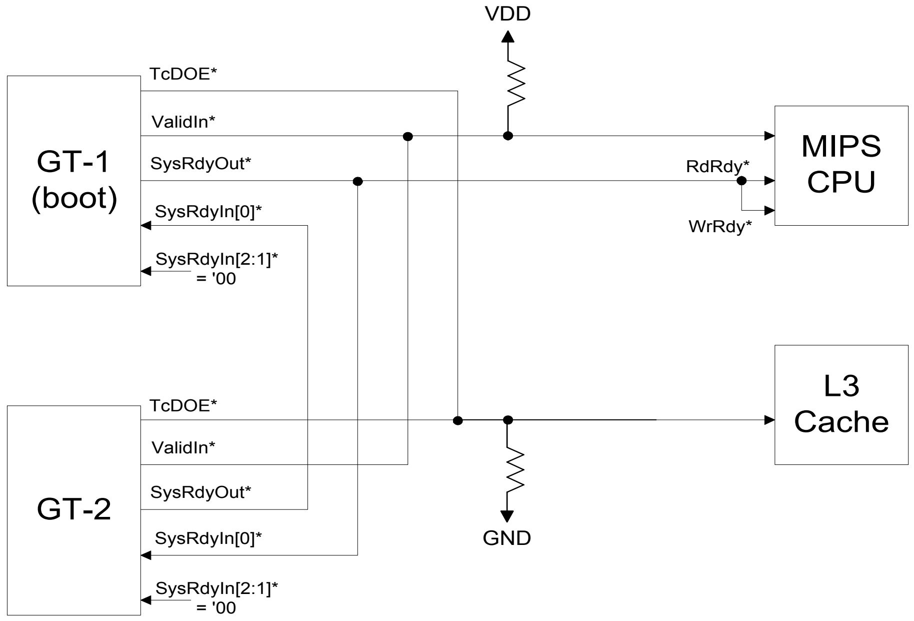
There is a new input signal related to the multi-GT–64242 configuration - SysRdyIn[2:0]*. This signal functions differently in the boot GT–64242 device than the other GT–64242 devices. The CPU RdRdy* and WrRdy* inputs are connected only to SysRdyOut* of the boot GT–64242 device. SysRdyOut* of the other GT–64242 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–64242 configurations. To compensate on the sampling stage this signal passes in the boot GT–64242 device.

SysRdyIn[0]* of all the GT–64242 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–64242 devices is shown in Figure 9.

Figure 9: Multi-GT–64242 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–64242 responds after a timeout period defined in NoMatchCnt field and completes the transaction, see Table 68 on page 73.

NOTES: In a multi-GT–64242 configuration, the GT–64242 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–64242 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–64242, the system might hang in case of address mismatch. To avoid a system hang, the non-GT–64242 slave device must have some address mismatch protection mechanism.

4.9.2 Multi-GT Mode Enabled

In multi-GT mode, each GT–64242 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 24.

Table 24: Multi-GT ID Encoding

Pin	Configuration Function
ID	Multi-GT–64242 Address ID
00	GT–64242 responds to SysAD[26:25]='00'
01	GT–64242 responds to SysAD[26:25]]='01'
10	GT–64242 responds to SysAD[26:25]='10'
11	GT–64242 responds to SysAD[26:25]='11' **NOTE:** The boot GT–64242 ID must be programmed to '11'.

If the GT–64242 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–64242 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–64242 System

The following procedure is recommended to initialize a system with two GT–64242s attached to the same CPU.

NOTE: For this example, the two GT–64242s are called GT-1 and GT-2, GT-1 ID is '11' (boot GT–64242) 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–64242 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–64242s resume NORMAL operation with USUAL address decoding.

NOTE: In the presence of multiple GT–64242 devices, each devices' CPU Configuration register must be programed to the same value.

4.10 Parity Support

The GT–64242 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 115.

NOTE:

4.11 CPU Endian Support

The CPU bus endianess is determined via the CPU Configuration register's Endianess bit, see Table 68 on page 73. The GT–64242 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–64242 (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–64242 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–64242 maintains MByteSwap and MWordSwap bits in the PCI Command register, see Table 193 on page 176. If the PCI Command register's MSwapEn bit is set to '1', the GT–64242 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.

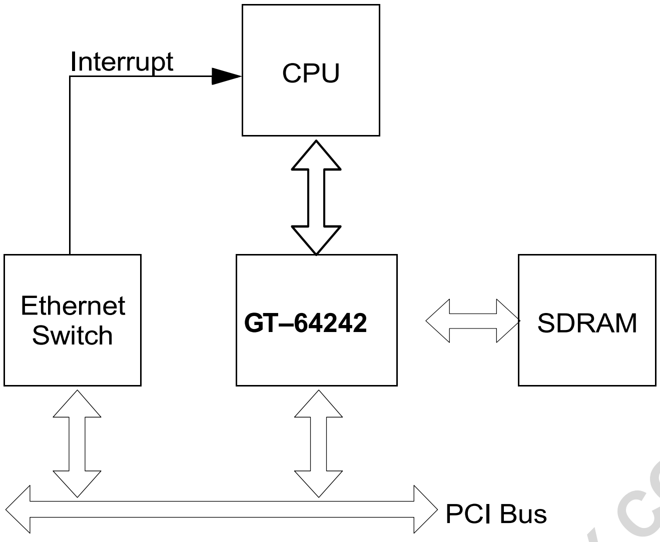
See Section 8.12 "Data Endianess" on page 149 for more information on data swapping.

4.12 CPU Synchronization Barrier

The GT–64242 supports a sync barrier mechanism. This mechanism is a hardware hook to help software synchronize between the CPU and PCI activities. The GT–64242 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–64242 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–64242 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 20. “Reset Configuration” on page 311. 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–64242 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 25: CPU Address Decode Register Map

Register	Offset	Page
SCS[0]* Low Decode Address	0x008	page 65
SCS[0]* High Decode Address	0x010	page 65
SCS[1]* Low Decode Address	0x208	page 65
SCS[1]* High Decode Address	0x210	page 65
SCS[2]* Low Decode Address	0x018	page 66
SCS[2]* High Decode Address	0x020	page 66
SCS[3]* Low Decode Address	0x218	page 66
SCS[3]* High Decode Address	0x220	page 66
CS[0]* Low Decode Address	0x028	page 66
CS[0]* High Decode Address	0x030	page 66
CS[1]* Low Decode Address	0x228	page 67
CS[1]* High Decode Address	0x230	page 67
CS[2]* Low Decode Address	0x248	page 67

Table 25: CPU Address Decode Register Map (Continued)

Register	Offset	Page
CS[2]* High Decode Address	0x250	page 67
CS[3]* Low Decode Address	0x038	page 67
CS[3]* High Decode Address	0x040	page 67
Boot CS* Low Decode Address	0x238	page 68
Boot CS* High Decode Address	0x240	page 68
PCI I/O Low Decode Address	0x048	page 68
PCI I/O High Decode Address	0x050	page 68
PCI Memory 0 Low Decode Address	0x058	page 68
PCI Memory 0 High Decode Address	0x060	page 69
PCI Memory 1 Low Decode Address	0x080	page 69
PCI Memory 1 High Decode Address	0x088	page 69
PCI Memory 2 Low Decode Address	0x258	page 70
PCI Memory 2 High Decode Address	0x260	page 70
PCI Memory 3 Low Decode Address	0x280	page 70
PCI Memory 3 High Decode Address	0x288	page 70
Internal Space Decode Address	0x068	page 70
PCI I/O Address Remap	0x0f0	page 71
PCI Memory 0 Remap (Low)	0x0f8	page 72
PCI Memory 0 Remap (High)	0x320	page 72
PCI Memory 1 Remap (Low)	0x100	page 72
PCI Memory 1 Remap (High)	0x328	page 72
PCI Memory 2 Remap (Low)	0x2f8	page 72
PCI Memory 2 Remap (High)	0x330	page 72
PCI Memory 3 Remap (Low)	0x300	page 73
PCI Memory 3 Remap (High)	0x338	page 73

Table 26: CPU Control Register Map

Register	Offset	Page
CPU Configuration	0x000	page 73
CPU Mode	0x120	page 75

Table 26: CPU Control Register Map (Continued)

Register	Offset	Page
CPU Read Response Crossbar Control (Low)	0x170	page 76
CPU Read Response Crossbar Control (High)	0x178	page 76

Table 27: CPU Sync Barrier Register Map

Register	Offset	Page
PCI Sync Barrier Virtual Register	0x0c0	page 77

Table 28: CPU Access Protection Register Map

Register	Offset	Page
Protect Low Address 0	0x180	page 77
Protect High Address 0	0x188	page 77
Protect Low Address 1	0x190	page 78
Protect High Address 1	0x198	page 78
Protect Low Address 2	0x1a0	page 78
Protect High Address 2	0x1a8	page 79
Protect Low Address 3	0x1b0	page 79
Protect High Address 3	0x1b8	page 79
Protect Low Address 4	0x1c0	page 79
Protect High Address 4	0x1c8	page 80
Protect Low Address 5	0x1d0	page 80
Protect High Address 5	0x1d8	page 80
Protect Low Address 6	0x1e0	page 81
Protect High Address 6	0x1e8	page 81
Protect Low Address 7	0x1f0	page 81
Protect High Address 7	0x1f8	page 82

Table 29: CPU Error Report Register Map

Register	Offset	Page
CPU Error Address (Low)	0x070	page 82

Table 29: CPU Error Report Register Map (Continued)

Register	Offset	Page
CPU Error Address (High)	0x078	page 83
CPU Error Data (Low)	0x128	page 83
CPU Error Data (High)	0x130	page 83
CPU Error Parity	0x138	page 83
CPU Error Cause	0x140	page 84
CPU Error Mask	0x148	page 84

4.15.1 CPU Address Decode Registers

Table 30: 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 31: 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 32: 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 33: 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 34: 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 35: 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 36: 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 37: 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 38: 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 39: 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 40: 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 41: 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 42: 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 43: 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 44: CS[3]* Low Decode Address, Offset: 0x038

Bits	Field Name	Function	Initial Value
15:0	LowAddr	CS[3] Base Address	0x0f0
31:16	Reserved	Reserved.	0x0

Table 45: CS[3]* High Decode Address, Offset: 0x040

Bits	Field Name	Function	Initial Value
11:0	HighAddr	CS[3] Top Address	0xf7
31:12	Reserved	Reserved.	0x0

Table 46: BootCS* Low Decode Address, Offset: 0x238

Bits	Field Name	Function	Initial Value
15:0	LowAddr	BootCS Base Address	0x0f8
31:16	Reserved		0x0

Table 47: BootCS* High Decode Address, Offset: 0x240

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

Table 48: PCI 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
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 49: PCI 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 50: PCI Memory 0 Low Decode Address, Offset: 0x058

Bits	Field Name	Function	Initial Value
15:0	LowAddr	PCI Memory 0 Base Address	0x0120
23:16	Reserved	Reserved.	0x0

Table 50: PCI Memory 0 Low Decode Address, Offset: 0x058 (Continued)

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
27	PCIReq64	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

Table 51: PCI Memory 0 High Decode Address, Offset: 0x060

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

Table 52: PCI Memory 1 Low Decode Address, Offset: 0x080

Bits	Field Name	Function	Initial Value
15:0	LowAddr	PCI 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 53: PCI Memory 1 High Decode Address, Offset: 0x088

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

Table 54: PCI Memory 2 Low Decode Address, Offset: 0x258

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

Table 55: PCI Memory 2 High Decode Address, Offset: 0x260

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

Table 56: PCI Memory 3 Low Decode Address, Offset: 0x280

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

Table 57: PCI Memory 3 High Decode Address, Offset: 0x288

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

Table 58: Internal Space Decode, Offset: 0x068

Bits	Field Name	Function	Initial Value
11:0	IntDecode	GT–64242 Internal Space Base Address	0x0140
23:15	Reserved	Reserved.	0x0

Table 58: Internal Space Decode, Offset: 0x068 (Continued)

Bits	Field Name	Function	Initial Value
26:24	PCISwap	Same as PCI 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 59: PCI I/O Address Remap, Offset: 0x0f0

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

Table 60: PCI Memory 0 Address Remap (Low), Offset: 0x0f8

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

Table 61: PCI Memory 0 Address Remap (High), Offset: 0x320

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

Table 62: PCI Memory 1 Address Remap (Low), Offset: 0x100

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

Table 63: PCI Memory 1 Address Remap (High), Offset: 0x328

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

Table 64: PCI Memory 2 Address Remap (Low), Offset: 0x2f8

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

Table 65: PCI Memory 2 Address Remap (High), Offset: 0x330

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

Table 66: PCI Memory 3 Address Remap (Low), Offset: 0x300

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

Table 67: PCI Memory 3 Address Remap (High), Offset: 0x338

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

4.15.2 CPU Control Registers

Table 68: CPU Configuration, Offset: 0x000

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

Table 68: CPU Configuration, Offset: 0x000 (Continued)

Bits	Field Name	Function	Initial Value
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–64242. 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–64242 drives SysCmd[4] to 1 during reads. This indicates to the CPU not to check read parity. 1 - Connected	0x0
21:20	Reserved	Reserved.	0x0
22	PErrProp	Parity Error Propagation 0 - The GT–64242 drives good parity on SysADC during CPU reads 1 - The GT–64242 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
26:23	Reserved	Reserved.	0x0
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	0x0

Table 68: CPU Configuration, Offset: 0x000 (Continued)

Bits	Field Name	Function	Initial Value
29	IOSBDis	I/O Read Sync Barrier Disable 0 - Sync Barrier enabled 1 - Sync Barrier disabled	0x0
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 69: CPU Mode, Offset: 0x120

Bits	Field Name	Function	Initial Value
1:0	MultiGTID	Multi-GT ID Represents the ID to which the GT–64242 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 70: 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 read data 0x4 - Reserved 0x5 - Comm unit internal registers read data 0x6 - IDMA 0/1/2/3 internal registers read data 0x7 - 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 71: 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 72: PCI Sync Barrier Virtual Register, Offset: 0x0c0

Bits	Field Name	Function	Initial Value
31:0	SyncBarrier	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 73: 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 74: 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 75: 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 76: 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 77: 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
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 78: 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 79: 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 80: 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 81: 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

Table 81: CPU Protect Address 4 (Low), Offset: 0x1c0

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 82: 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 83: 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
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 84: 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

Table 84: CPU Protect Address 5 (High), Offset: 0x1d8

Bits	Field Name	Function	Initial Value
31:12	Reserved	Reserved.	0x0

Table 85: 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 86: 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 87: 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

Table 87: CPU Protect Address 7 (Low), Offset: 0x1f0

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

Table 88: 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 89: 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 90: 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 91: 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 92: 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 93: 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 94: 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 95: 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
31:7	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–64242.

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, 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 100MHz, 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–64242 DRAM controller to interface SDRAM or registered SDRAM, according to the setting of DType bits in the SDRAM Configuration register, see Table 102 on page 106.

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 88 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

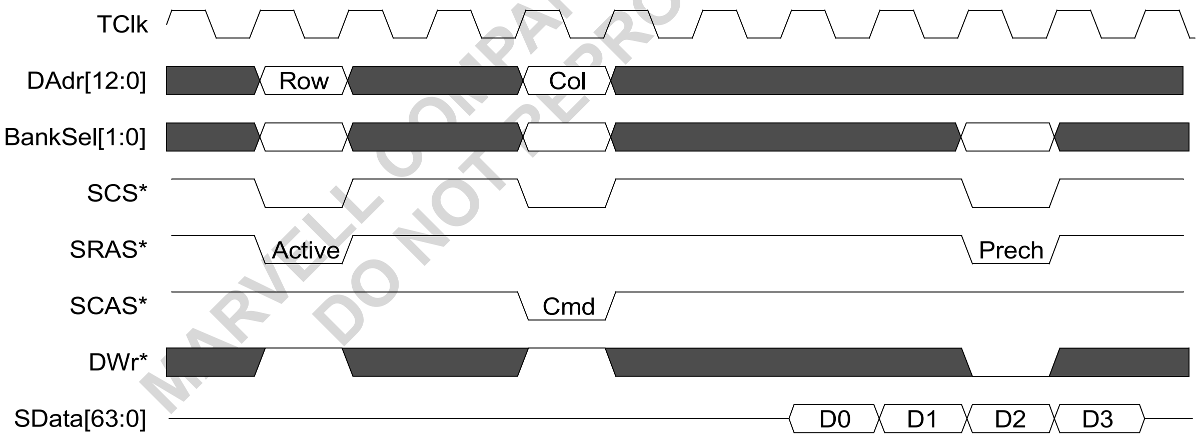
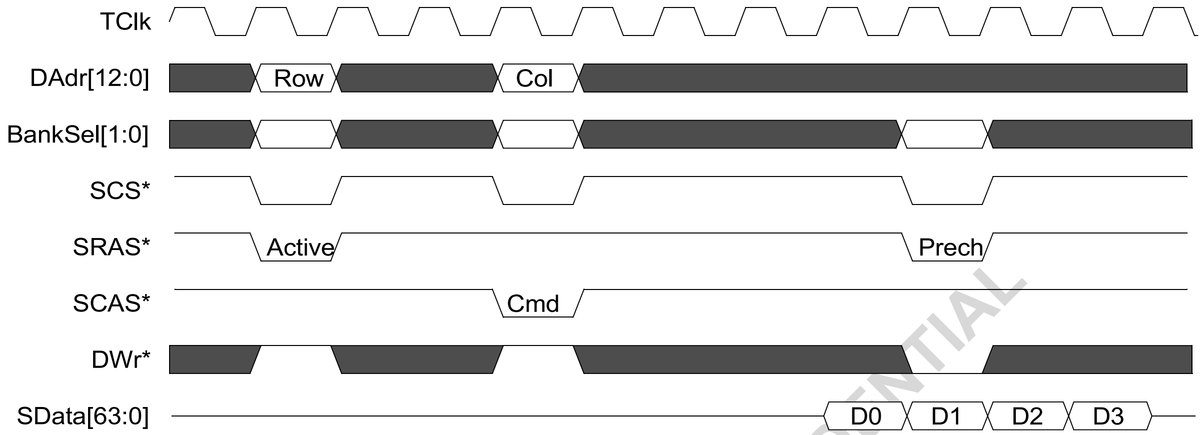


Figure 12 shows a registered SDRAM read. In registered SDRAM, all address and control signals (DAdr[12:0], BankSel[1:0], RAS*, CAS*, DWr*, CS* and DQM*) are registered externally. This means the signals arrive to the SDRAM device one cycle after they are driven by the DRAM controller. It also means that 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–64242 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–64242 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–64242 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–64242 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–64242 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–64242 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 105 on page 108.

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 99 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.

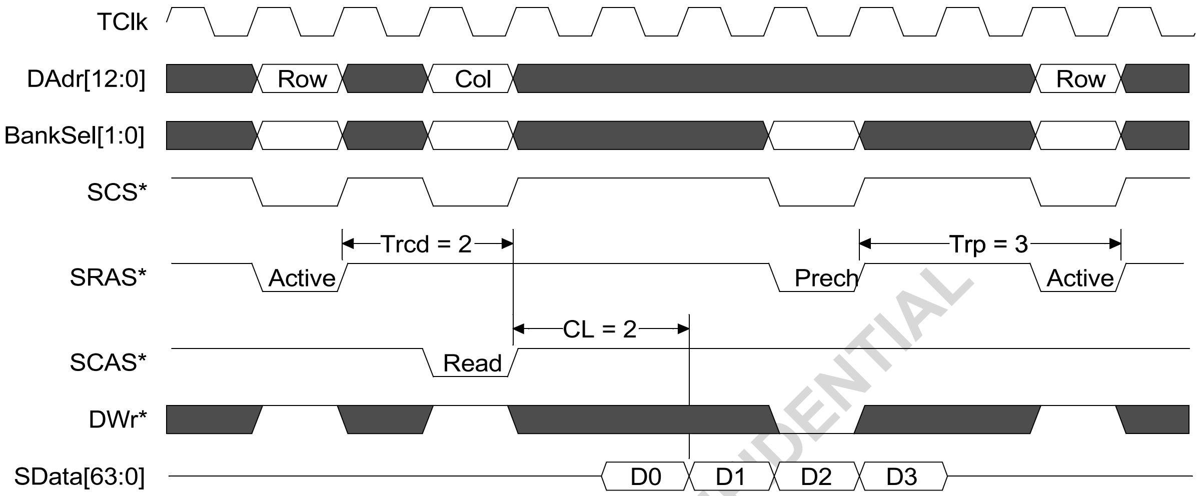
5.4.3 SRAS* to SCAS* (Trcd)

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



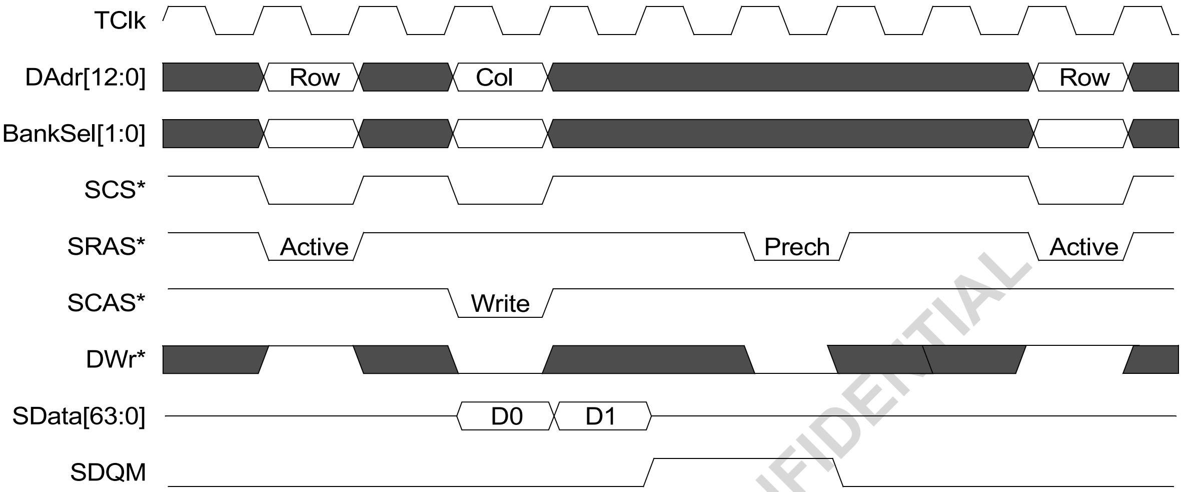
5.5 SDRAM Burst

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

The GT–64242 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.

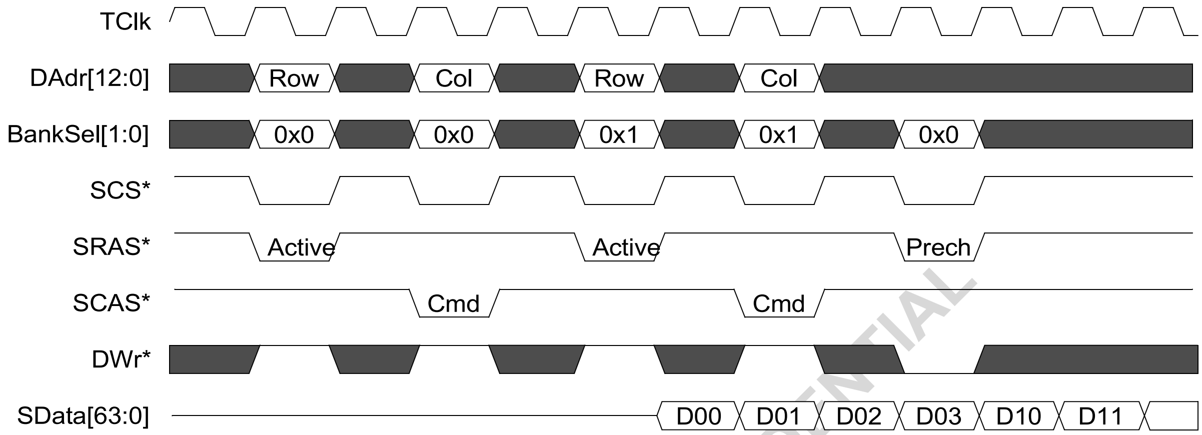
5.6 SDRAM Interleaving

The GT–64242 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 94.

5.6.1 Bank Interleaving Implementation

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

It occurs in the GT–64242 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–64242 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–64242 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 102 on page 106.

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, 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 96 through Table 98.

Table 96: 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 97: 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 97: 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 98: 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 98: 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.

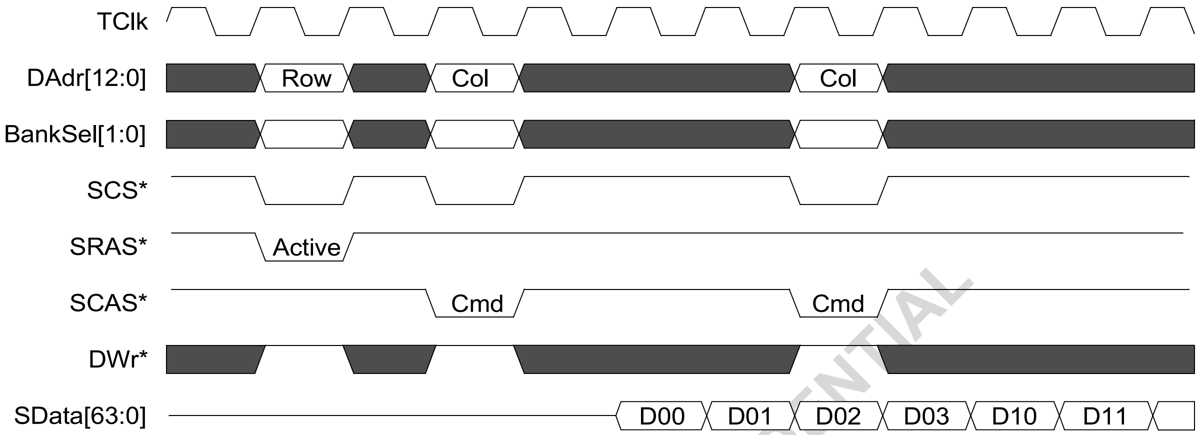
[6] Only for 128Mx4 devices.

5.7 SDRAM Open Pages

It is possible to configure the GT–64242 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

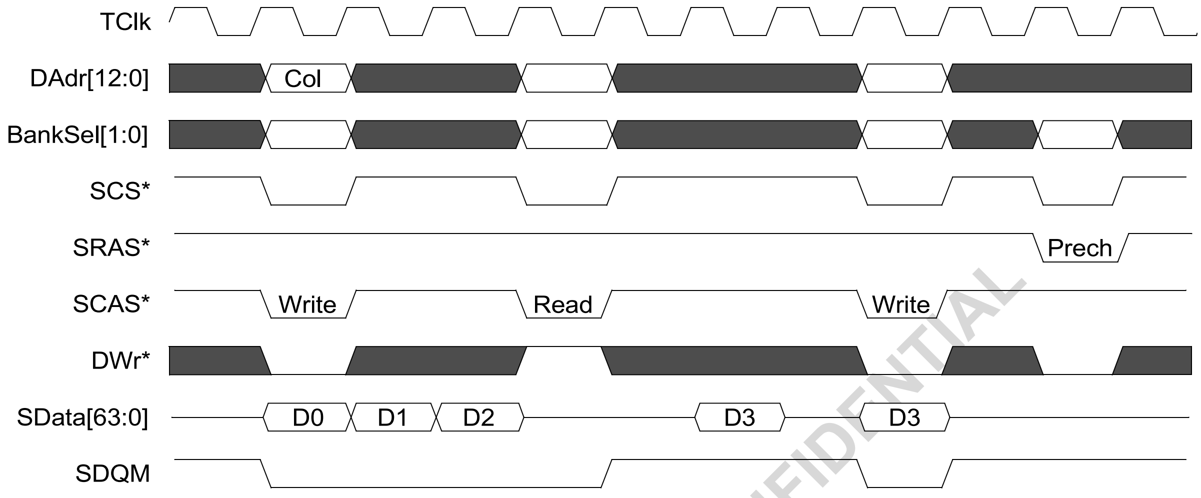
The GT–64242 supports Error Checking and Correction (ECC).

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–64242 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 115.

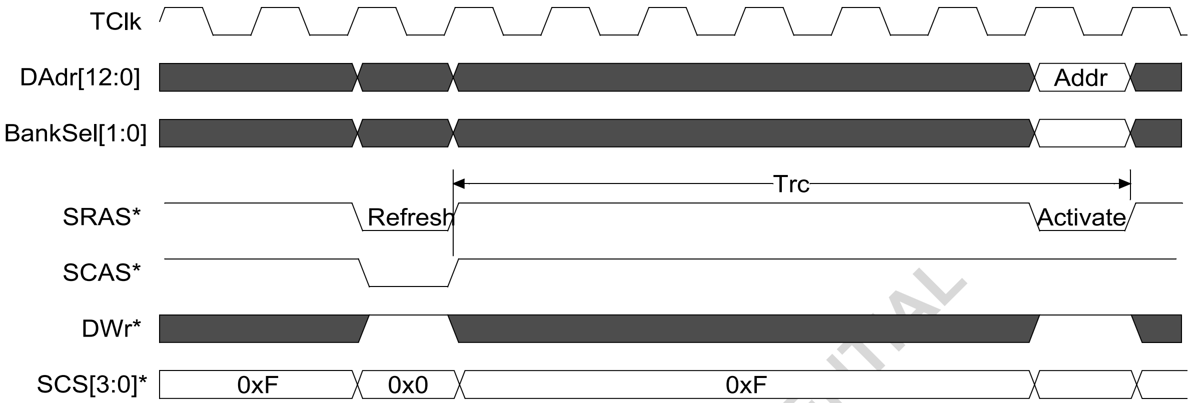
5.9 SDRAM Refresh

The GT–64242 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 100MHz, 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.

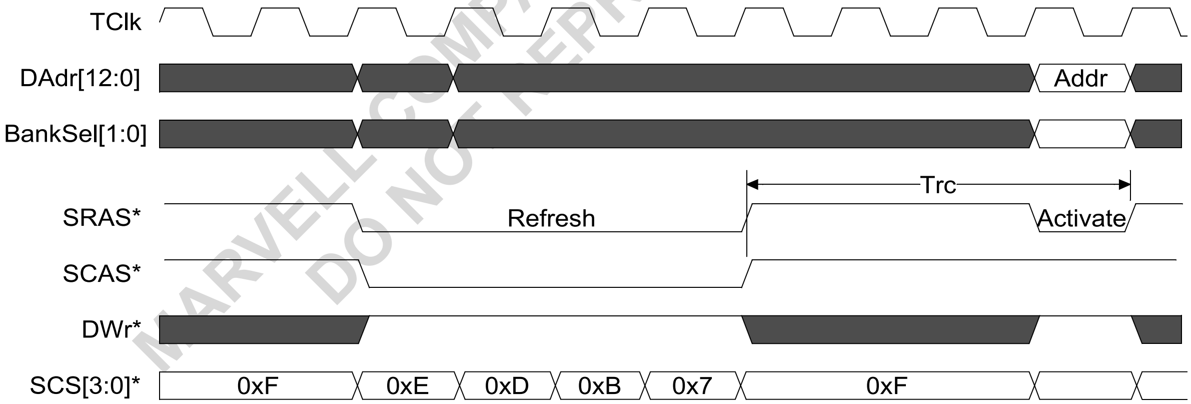
Figure 18: Non-Staggered Refresh Waveform



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–64242 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–64242 I^{2}C 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

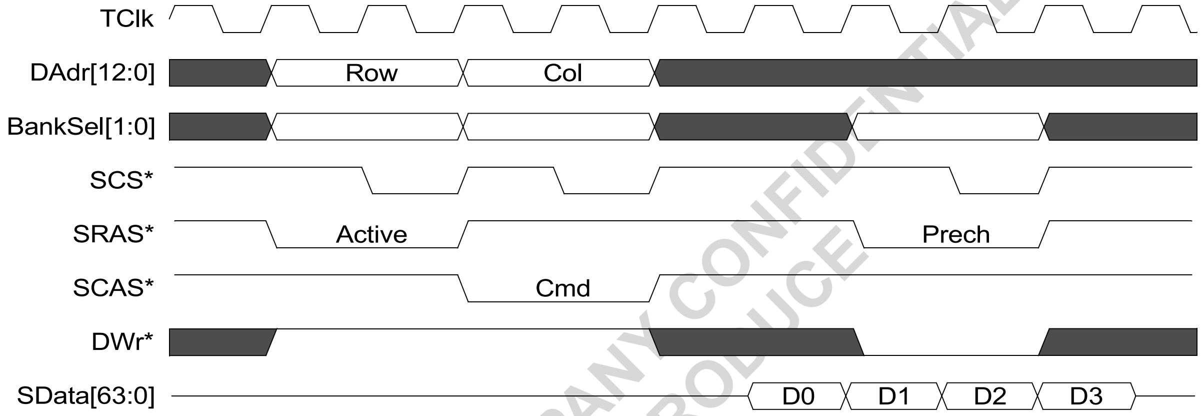
On the particular bank that is accessed, use the Force Refresh Command to execute a refresh cycle.

5.12 Heavy Load Interface

When interfacing heavy load, unbuffered DIMMs (above 50 pF), the GT–64242 might not meet the DRAM control lines AC spec at 100MHz. 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 100MHz, 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–64242 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–64242 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–64242 is designed to interface SDRAM at 100MHz, assuming both the GT–64242 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–64242 clock). However, the GT–64242 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–64242 and DRAM IBIS models.

5.13.1 SDRAM Clock Output

The GT–64242 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 23 on page 327).

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–64242 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 105 on page 108.

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

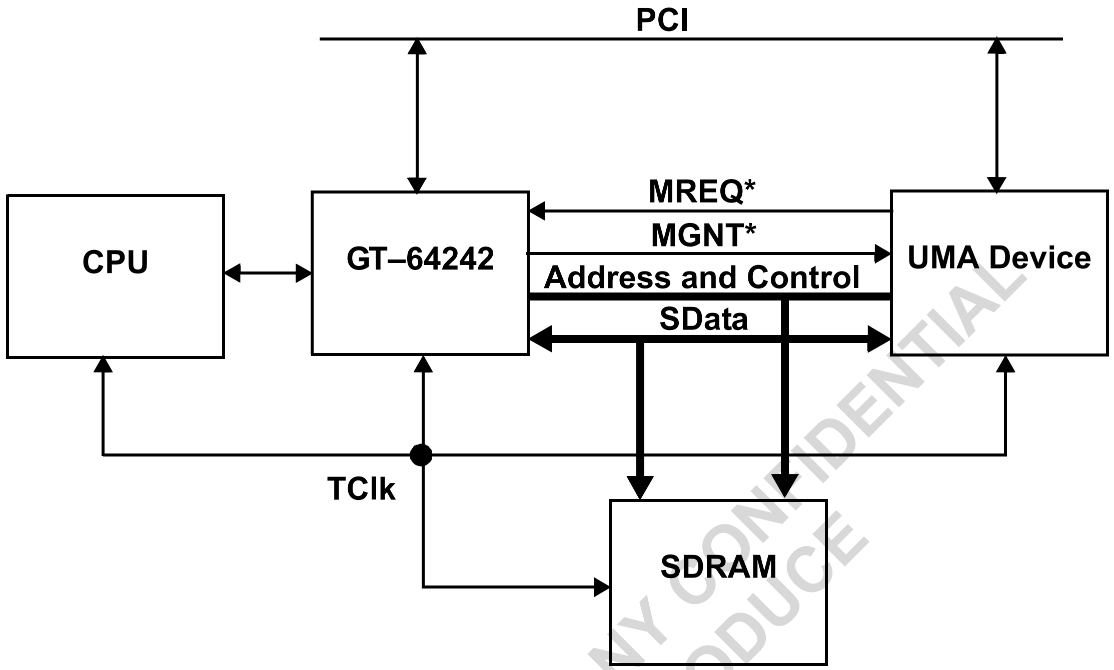
With the additional sampling stage, DRAM read latency is increased by one cycle

5.14 Unified Memory Architecture Support

The GT–64242 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–64242.

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–64242 Sharing SDRAM



At reset, the GT–64242 can be configured to act as a UMA master or slave. This is particularly required when the DRAM is shared between multiple GT–64242 devices. With two GT–64242 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–64242 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 15.1 “MPP Multiplexing” on page 275.

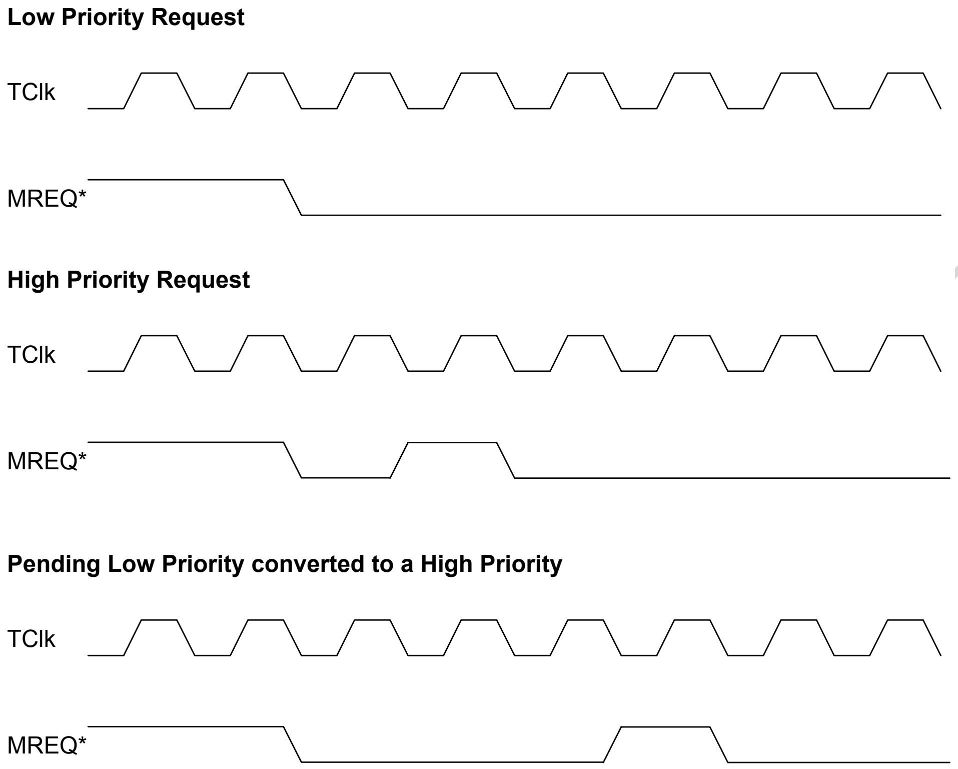
5.14.1 SDRAM Bus Arbitration

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–64242, 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–64242 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–64242 asserts MGNT* and release the bus, as soon as it's done with the current outstanding transaction.

NOTE: When the GT–64242 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–64242 interfaces request access to SDRAM or MREQ* is deasserted, the GT–64242 deasserts MGNT* to indicate that it requires bus ownership.

After reset deassertion, the GT–64242 generates DRAM initialization sequence. It responds to MREQ* only after initialization completes.

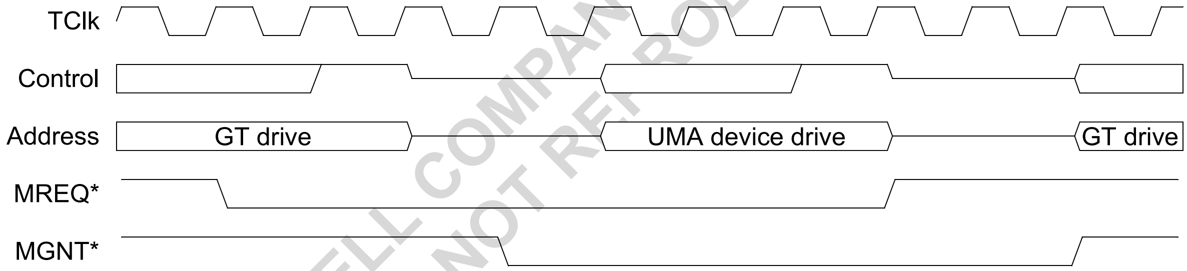
The following rules must be followed by a UMA master device:

- 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–64242 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–64242 regains access to SDRAM. During this period, the UMA device must drive these signals to access SDRAM.

When the GT–64242 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–64242 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–64242 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–64242 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–64242 also contains a UMA Bus Release Counter that determines how many cycles after gaining bus ownership the GT–64242 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–64242 devices sharing the same DRAM.

NOTE: When the GT–64242 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 99: SDRAM Configuration Register Map

Register	Offset	Page
SDRAM Configuration	0x448	page 106
SDRAM Operation Mode	0x474	page 107
SDRAM Address Control	0x47c	page 108
SDRAM Timing Parameters	0x4b4	page 108
SDRAM UMA Control	0x4a4	page 109
SDRAM Interface Crossbar Control (Low)	0x4a8	page 109
SDRAM Interface Crossbar Control (High)	0x4ac	page 110
SDRAM Interface Crossbar Timeout	0x4b0	page 110

Table 100: SDRAM Banks Parameters Register Map

Register	Offset	Page
SDRAM Bank0 Parameters	0x44c	page 110
SDRAM Bank1 Parameters	0x450	page 111

Table 100: SDRAM Banks Parameters Register Map (Continued)

Register	Offset	Page
SDRAM Bank2 Parameters	0x454	page 111
SDRAM Bank3 Parameters	0x458	page 112

Table 101: Error Report Register Map

Register	Offset	Page
SDRAM Error Data (Low)	0x484	page 112
SDRAM Error Data (High)	0x480	page 112
SDRAM Error Address	0x490	page 112
SDRAM Received ECC	0x488	page 112
SDRAM Calculated ECC	0x48c	page 113
SDRAM ECC Control	0x494	page 113
SDRAM ECC Error Counter	0x498	page 113

5.15.1 SDRAM Configuration Registers

Table 102: 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 102: 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 read Bit[28] - Reserved Bit[29] - Comm ports read Bit[30] - IDMA channels 0/1/2/3 read Bit[31] - Reserved	0x36

Table 103: 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 104: 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 91.	0x2
31:4	Reserved	Reserved.	0x0

Table 105: 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
31:14	Reserved	Reserved.	0x0

Table 106: 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 107: 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 access 0x4 - Reserved 0x5 - Comm unit access 0x6 - IDMA channels 0/1/2/3 access 0x7 - 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

Table 107: SDRAM Interface Crossbar Control (Low), Offset: 0x4a8 (Continued)

Bits	Field name	Function	Initial Value
27:24	Arb6	Slice 6 of device controller "pizza" arbiter.	0x0
31:28	Arb7	Slice 7 of device controller "pizza" arbiter.	0x0

Table 108: 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 109: 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 110: SDRAM Bank0 Parameters, Offset: 0x44c

Bits	Field name	Function	Initial Value
13:0	Reserved	Reserved.	0x0

Table 110: SDRAM Bank0 Parameters, Offset: 0x44c (Continued)

Bits	Field name	Function	Initial Value
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 111: 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 112: 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 113: 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 114: 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 115: 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 116: 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 117: SDRAM Received ECC, Offset: 0x488

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

Table 117: SDRAM Received ECC, Offset: 0x488

Bits	Field Name	Function	Initial Value
31:8	Reserved	Reserved.	0x0

Table 118: SDRAM Calculated ECC, Offset: 0x48c

Bits	Field Name	Function	Initial Value
7:0	ECCCalc	ECC code calculated by the GT–64242.	0x0
31:8	Reserved	Reserved.	0x0

Table 119: 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
23:16	ThrEcc	Threshold ECC Interrupt Number of single bit errors that occur before the GT–64242 generates an interrupt. **NOTE:** If set to 0x0, the GT–64242 does not generate an interrupt in case of a single bit error.	0x0
31:24	Reserved	Reserved.	0x0

Table 120: 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–64242 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–64242 samples data parity driven by the CPU with each data.

When a parity error occurs, the GT–64242 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–64242 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–64242 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 121. For example, data bit 12 has the check value of 01100001, and check bit 5 has the check value of 00100000.

Table 121: ECC Code Matrix

Check Bit	Data Bit	ECC Code Bits								Number of 1s in syndrome
		7	6	5	4	3	2	1	0	
	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 121: 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 121: 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 121: ECC Code Matrix (Continued)

Check Bit	Data Bit	ECC Code Bits								Number of 1s in syndrome
		7	6	5	4	3	2	1	0	
	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–64242 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 121, 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–64242 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–64242 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 121 shows that this syndrome corresponds to check bit 3. The GT–64242 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 121. In this case, the GT–64242 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–64242 reports an error but will not change the data.

6.2.2 SDRAM Interface Operation

On SDRAM reads, the GT–64242 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–64242 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–64242 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–64242 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–64242 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 119 on page 113.

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–64242 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 112.

The GT–64242 reports an ECC error whenever it detects but cannot correct an error (2, 3, or 4 bits errors).

The GT–64242 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 119 on page 113.

- If the threshold is set to ‘0’, there is no report on single bit errors.
- If set to ‘1’, GT–64242 reports each single bit error.
- If set to ‘n’, GT–64242 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–64242 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 14.3 "GPP Interrupts" on page 272).

6.4 PCI Parity Support

The GT–64242 implements all parity features required by the PCI spec, including PAR, PAR64*, PERR*, and SERR* generation and checking.

As an initiator, the GT–64242 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–64242 detects bad parity and the Status and Command Configuration register's PErrEn bit is set (see Table 241 on page 200), it asserts PERR*.

As a target, the GT–64242 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–64242 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 232 on page 194), the GT–64242 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 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–64242 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–64242 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–64242 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–64242 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–64242 does not support READ byte enables.

The GT–64242 does not support multiple masters on the AD bus or access to the different GT–64242 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–64242 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–64242. The minumum setting is 0x3.

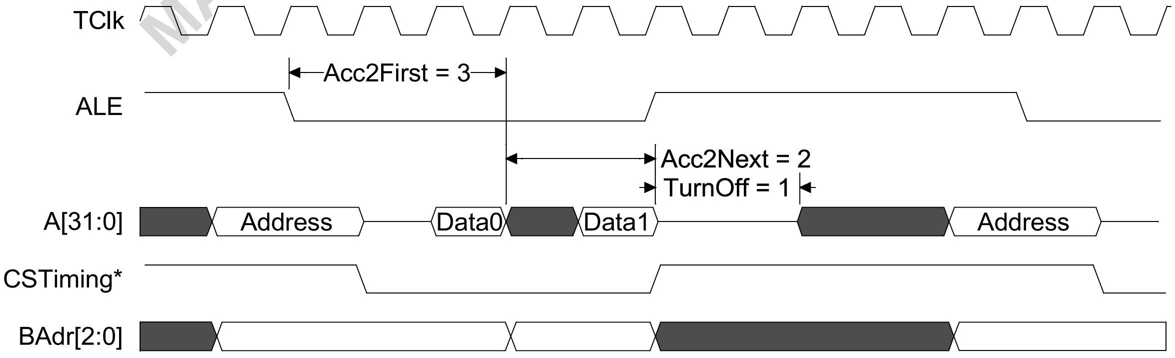
NOTE: Extend this parameter by extending the Ready* pin, see 7.4.

7.2.3 Acc2Next

The Acc2Next parameter defines the number of TClk cycles between the cycle that samples the first read data by GT–64242 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.

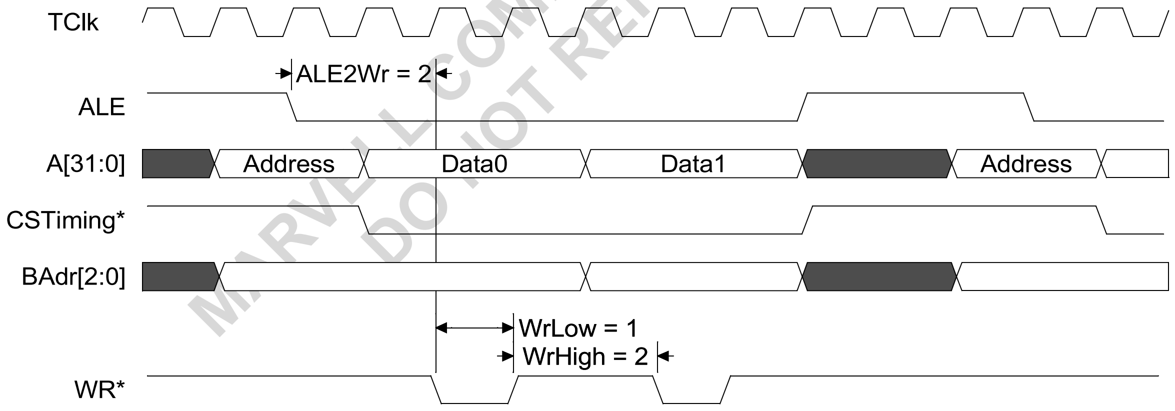
7.2.6 WrHigh

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



7.3 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 205 on page 187); the IDMA BurstLimit must not exceed 32 bytes (see Table 348 on page 252); the Ethernet SDMA BSZ Burst is limited to 4 64bit words (see Table 482 on page 354); and, the MPSC's SDMA BSZ is limited to 4 64bit words (Table 545 on page 430.

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).

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 127 on page 130). 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 26, Figure 27, and Figure 28 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–64242 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 26: Ready* Extending Acc2First

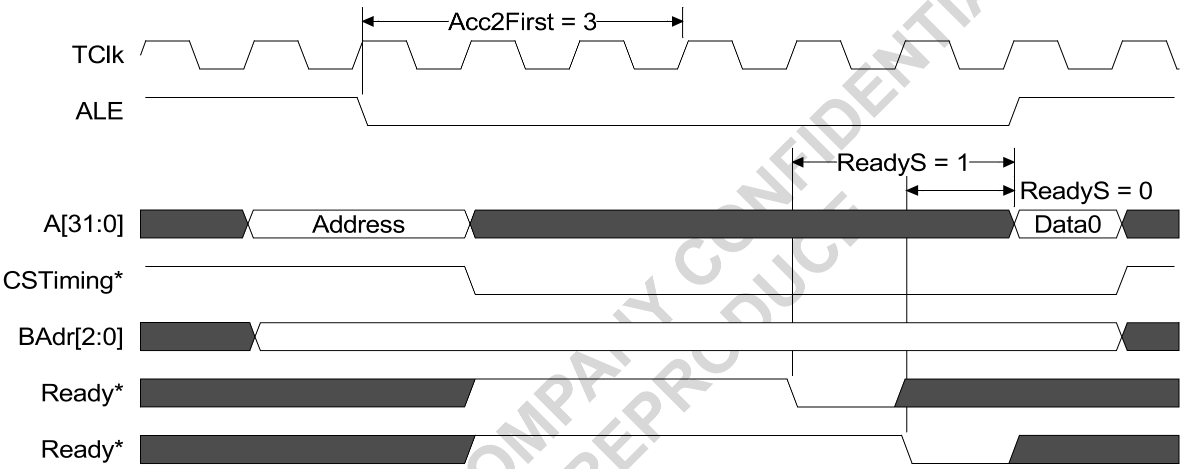


Figure 27: Ready* Extending Acc2Next

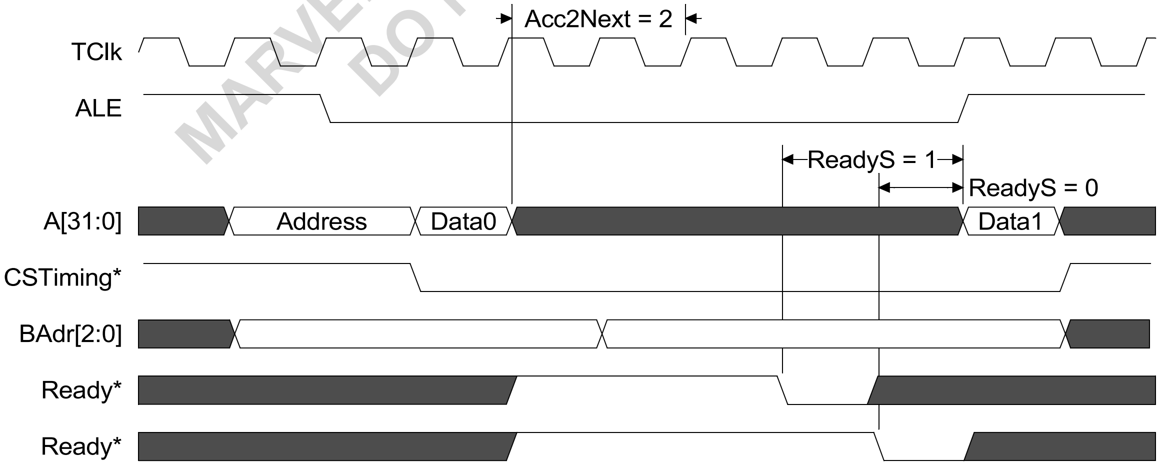
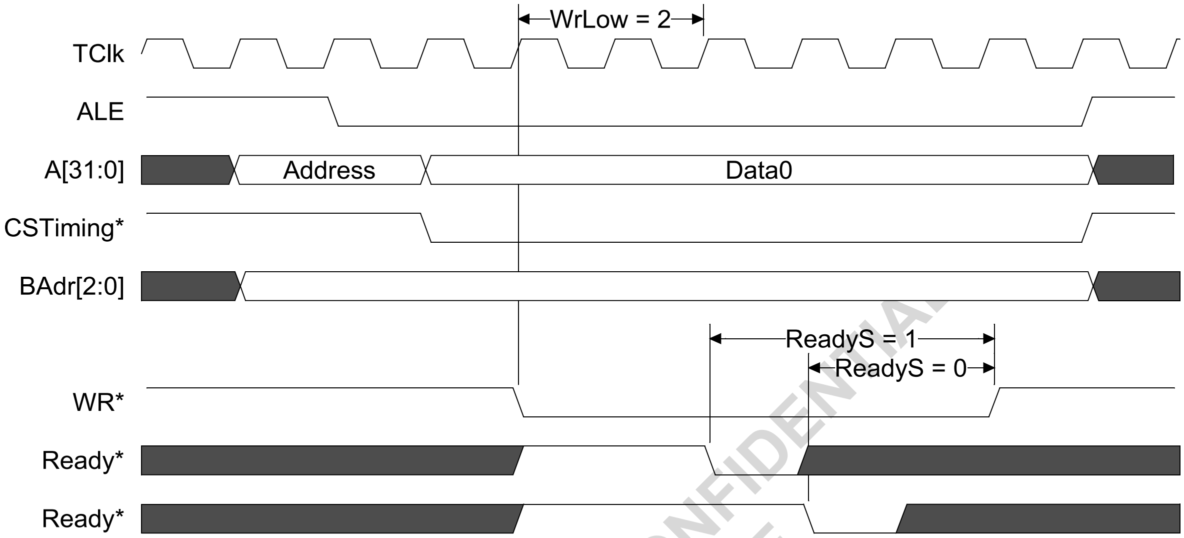


Figure 28: Ready* Extending WrLow Parameter



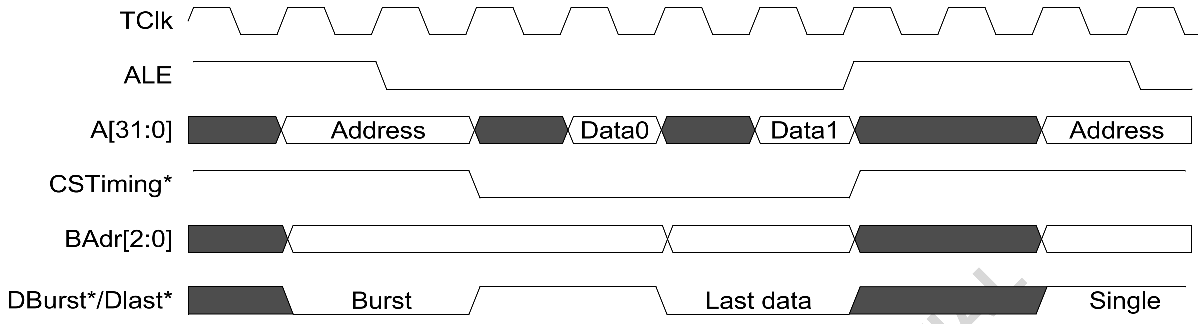
The Device Interface Control register’s ReadyS bit [19] determines the Ready* input sample window, see Table 132 on page 132. If set to ‘1’, the device controller samples read data two cycles after Ready* assertion on a read access, and de-asserts Wr* two cycles after Ready* assertion on a write access. 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 23. “AC Timing” on page 327, 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–64242 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 29 shows an example.

Figure 29: 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 122: 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 123: 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 124: 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 125: Device Control Register Map

Register	Offset	Page
Device Bank0 Parameters	0x45c	page 130
Device Bank1 Parameters	0x460	page 131

Table 125: Device Control Register Map (Continued)

Register	Offset	Page
Device Bank2 Parameters	0x464	page 131
Device Bank3 Parameters	0x468	page 131
Boot Device Parameters	0x46c	page 132
Device Interface Control	0x4c0	page 132
Device Interface Crossbar Control (Low)	0x4c8	page 132
Device Interface Crossbar Control (High)	0x4cc	page 133
Device Interface Crossbar Timeout	0x4c4	page 133

Table 126: Device Interrupts Register Map

Register	Offset	Page
Device Interrupt Cause	0x4d0	page 134
Device Interrupt Mask	0x4d4	page 134
Device Error Address	0x4d8	page 134

7.8.1 Device Control Registers

Table 127: 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–64242.	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–64242 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 127: 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
31:28	Reserved	Reserved.	0xf

Table 128: Device Bank1 Parameters, Offset: 0x460

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Device Bank0.	0xffefffff

Table 129: Device Bank2 Parameters, Offset: 0x464

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Device Bank0.	0xffefffff

Table 130: Device Bank3 Parameters, Offset: 0x468

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Device Bank0.	0xffefffff

Table 131: 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 132: 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
17:16	Reserved	Must be 0x3.	0x3
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.	0x0
31:19	Reserved	Reserved.	0x0

Table 133: 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 access 0x4 - Reserved 0x5 - Comm unit access 0x6 - IDMA channels 0/1/2/3 access 0x7 - 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

Table 133: Device Interface Crossbar Control (Low), Offset: 0x4c8 (Continued)

Bits	Field Name	Function	Initial Value
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 134: 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 135: 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 136: 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 137: 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 138: 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–64242 supports one 64-bit PCI interfaces, compliant to PCI specification rev. 2.2.

NOTE: When configured as a 32-bit bus, the GT–64242 drives PAD[63:32], CPE[7:4], PAR64 pins; a pull-up is not required.

8.1 PCI Master Operation

When the CPUor IDMA units 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–64242 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–64242 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 241 on page 200

The GT–64242 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–64242 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–64242 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 193 on page 176.

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 193 on page 176.
- 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 241 on page 200.
- 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 193 on page 176.

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–64242 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 195 on page 181

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–64242 supports both external arbiter or internal arbiter configuration through the PCI Arbiter Control register’s EN bit [31], see Table 198 on page 182. If the bit is set to ‘1’, the GT–64242 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 15.1 “MPP Multiplexing” on page 275. 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–64242 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–64242 integrates one PCI arbiter.

The PCI arbiters implements a weighted Round Robin (RR) arbitration mechanism. Each agent is assigned a programmable priority (either high or low) and the arbitration is done according to these priorities. A simple Round Robin arbitration is performed within each priority level, while a weighted function is implemented for arbitrating between the high priority and the low priority groups.

Figure 30 shows the arbitration flow.

The arbitration works as follows:

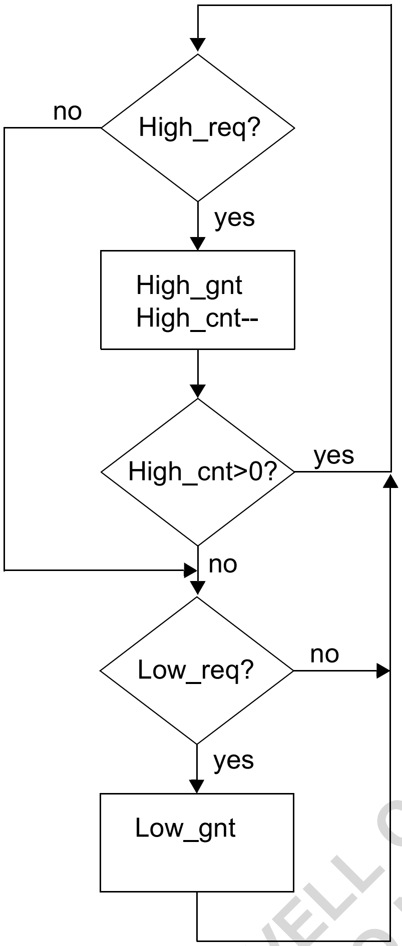
- The two request signals (High_req, Low_req) are generated by “ANDing” each of the request lines with its respective priority attribute, and “ORing” the results. For example:

  High_req = (req_[0] AND (req_prio[0]==high)) OR (req_[1] AND (req_prio[1]==high)) OR...

- There is a counter associated with the priority scheme - High_cnt. The counter assigns different weights to each priority level. This countdown counter decrements each time a high priority request (High_req) is granted. When High_cnt expires, a slot is opened for low priority requests and the counter is set to its preset value.

Each time a low priority request (Low_req) is granted, the High_cnt counter is preset.

Figure 30: Internal PCI Arbiter Flow



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 198 on page 182.

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.

8.4 PCI Master Configuration Cycles

The GT–64242 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–64242's internal configuration registers.

The GT–64242 contains two registers to support configuration accesses: PCI Configuration Address (Table 229 on page 193) and PCI Configuration Data (Table 230 on page 194). 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–64242 to translate that Configuration Address value to the requested cycle on the PCI bus or internal configuration space.

The GT–64242 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 139 shows DevNum to IdSel mapping (type 0 configuration access).

Table 139: 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 DevNum field is 0x1f.
- The function number is 0x7.
- The register offset is 0x0.

The GT–64242 configuration registers are accessed from the PCI bus when the GT–64242 is a target responding to PCI configuration read and write cycles.

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

[1] "Resistive Coupling" also means "hook a resistor from ADx to IdSel" on a given device.

transactions.

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 35. For an exact list of base address registers and size registers, see Section 8.18.1 “PCI Slave Address Decoding Registers” on page 167.

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.

8.5.1 SDRAM and Device BARs

The GT–64242 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–64242. Byte swapping is controlled via the Access Control registers. For more details, see Section 8.6 “PCI Access Protection” on page 142.

8.5.2 Internal Space Address Decoding

PCI accesses the GT–64242 internal registers using memory or I/O transactions.

There is a dedicated BAR for PCI. 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–64242 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 8.5.5.

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–64242 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 192 on page 176.

With the Expansion ROM disabled, the GT–64242 does not support expansion ROM BAR, offset 0x30 in the configuration space is reserved.

8.5.4 64-bit Addressing BARs

The GT–64242 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

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–64242 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.

8.5.5 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–64242 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.

8.5.6 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–64242 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–64242 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 8.7 and Section 8.12.

8.7 PCI Target Operation

The GT–64242 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–64242 does not act as a target for Interrupt Acknowledge and Special cycles (these cycles are ignored). The GT–64242 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.

8.7.1 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 205 on page 187. 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 MBurst 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.

8.7.2 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 205 on page 187).

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 MBurst 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 target device, the slave might need to insert wait states (TRDY* deasserted) between the data phases, according to the data rate from the target.

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.

8.7.3 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 reissue 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: A read marked as non-prefetchable is never treated as a delayed read (even if marked as delayed read through DReadEn bit [13]). The reason is that a delayed read buffer might be discarded if there is no read completion in time. If that happens, the read data is lost and the read becomes destructive.

8.7.4 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–64242 supports a Discard Timer register, see Table 196 on page 181. 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.

8.7.5 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 PCI Target Termination

The GT–64242 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.

8.8.1 Timeout Termination

The GT–64242 includes two 8-bit timeout registers (see Table 195 on page 181) – 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'.

On PCI access to expansion ROM, the slave behaves as if timeout0 and timeout1 are programed to never retry (0x0), to allow for the long default access time from a 8-bit boot ROM (similar to GT-64120 and GT-64130 implementation).

8.8.2 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.

8.9 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–64242 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–64242's space. The GT–64242 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.

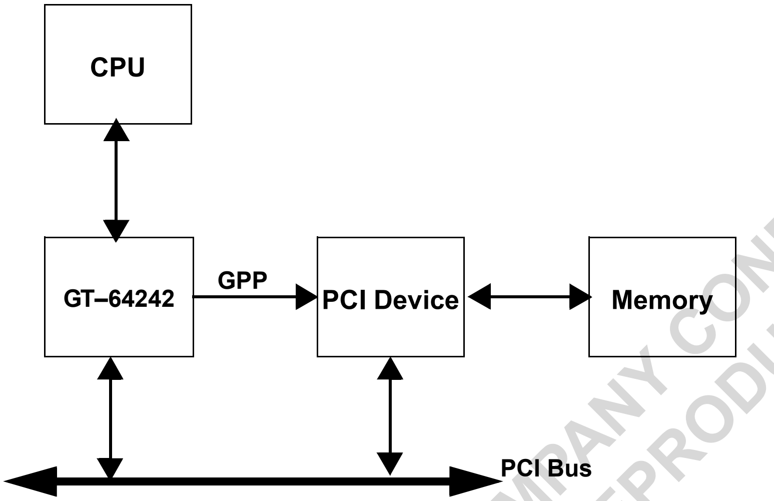
If serial ROM initialization is enabled, any PCI access to the GT–64242 is terminated with Retry. This lasts until the end of the initialization. See Section 20. "Reset Configuration" on page 311 for more details.

8.10 Synchronization Barrier

The GT–64242 supports a sync barrier mechanism. This mechanism is a hardware hook to help software synchronize between the CPU and PCI activities. The GT–64242 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–64242 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–64242 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 8.7.4.

An alternative method for generating the PCI slave sync barrier is reading from the PCI Sync Barrier Virtual register, see Table 204 on page 187. 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–64242 without suffering from sync barrier latency.

8.11 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–64242 has no special synch modes, for different frequency ranges. The only restriction is that the TClk frequency must be greater than the Clk frequency.

8.12 Data Endianess

The GT–64242 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–64242 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–64242 internal data path width). Table 140 shows an example of the data 0x0011223344556677.

Table 140: 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 141: 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 142: 64-bit PCI Byte and Word Swap Settings

	Little Endian PCI agent	Big Endian PCI agent
Big endian CPU bus	Byte swapping	No swapping

8.12.1 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–64242.

The GT–64242 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 193 on page 176), as in the GT-64120/130 devices.

The GT–64242 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–64242 supports data swapping also on the PCI access to internal registers via the PCI Command register's SIntSwap bits [26:24].

8.12.2 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 CPUand IDMAs units to interface different PCI targets with different endianess conventions.

The GT–64242 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 193 on page 176.

See the following sections for further details about transaction initiator endianess configuration:

- For CPU details: Section 4.11 "CPU Endian Support" on page 58.
- For IDMA details: Section 10.7 "Big and Little Endian Support" on page 243.

8.13 64-bit PCI Interface

The GT–64242 supports a 64-bit PCI interface. To operate as a 64-bit device, the REQ64* pin must be sampled LOW on RST* rise as required by PCI spec (Hold time of REQ64* in respect to RST* rise is '0').

When the GT–64242 is configured to 64-bit PCI, both master and target interfaces are configured to execute 64-bit transactions, whenever it is possible.

NOTE: Since PCI interface supports CompactPCI Hot Swap Ready, P64EN* pin is used to detect a 64-bit PCI bus rather than REQ64*. If not using CompactPCI, connect PCI REQ64* to P64EN* pin.

8.13.1 PCI Master 64-bit Interface

The PCI master interface always attempts to generate 64-bit transactions (asserts REQ64*), 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 ACK64*, 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 193 on page 176. For example the requested address is 0x4, the master issues a 64-bit transaction (assert REQ64*) with byte enables 0x0f. If the target does not respond with ACK64*, 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 REQ64*). 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 ACK64* to 64-bit transaction, the PCI master can be configured to always assert REQ64*, even if the amount of data needs to be transferred is less than or equal to 64-bit. Each initiating interface (CPUor IDMA 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 REQ64* is allowed only when the target PCI device responds with ACK64* 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 REQ64* is implemented to support 64-bit Big Endian devices on the PCI bus. The hook of forcing REQ64* is not fully compliant with the PCI specification, and must be used carefully.

8.13.2 PCI Slave 64-bit interface

The PCI target interface always responds with ACK64* to a 64-bit transaction, except for accesses to configuration space, internal registers, I_2O space, or I/O transaction.

8.14 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–64242 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.

8.15 PCI Parity and Error Support

The GT–64242 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 142 and Section 8.8 “PCI Target Termination” on page 146 for more details.

The PCI parity support is detailed in Section 6. “Address and Data Integrity” on page 115.

8.16 Configuration Space

The PCI slave supports Type 00 configuration space header as defined in PCI specification. The GT–64242 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–64242 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.

8.16.1 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–64242 responds to BAR sizing requests based on the values programmed into the Bank Size Registers (see Table 244 on page 203). 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–64242 serial ROM initialization, see Section 20. "Reset Configuration" on page 311 for more details.

Figure 32: PCI Configuration Space Header

Function 0 Header

Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Rev ID	08h
BIST	Header	Latency	Line Size	0Ch
SCS[0] BAR				10h
SCS[1] BAR				14h
SCS[2] BAR				18h
SCS[3] BAR				1Ch
Mem Mapped Internal BAR				20h
IO Mapped Internal BAR				24h
Reserved				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM BAR				30h
Reserved			Cap. Ptr	34h
Reserved				38h
Max_Lat	Min_Gnt	Int. Pin	Int. Line	3Ch

Function 1 Header

▨		▨		00h
▨		▨		04h
▨			▨	08h
▨	▨	▨	▨	0Ch
CS[0] BAR				10h
CS[1] BAR				14h
CS[2] BAR				18h
CS[3] BAR				1Ch
BootCS BAR				20h
Reserved				24h
Reserved				28h
▨		▨		2Ch
Reserved				30h
Reserved			Reserved	34h
Reserved				38h
▨	▨	▨	▨	3Ch

Function 2 Header

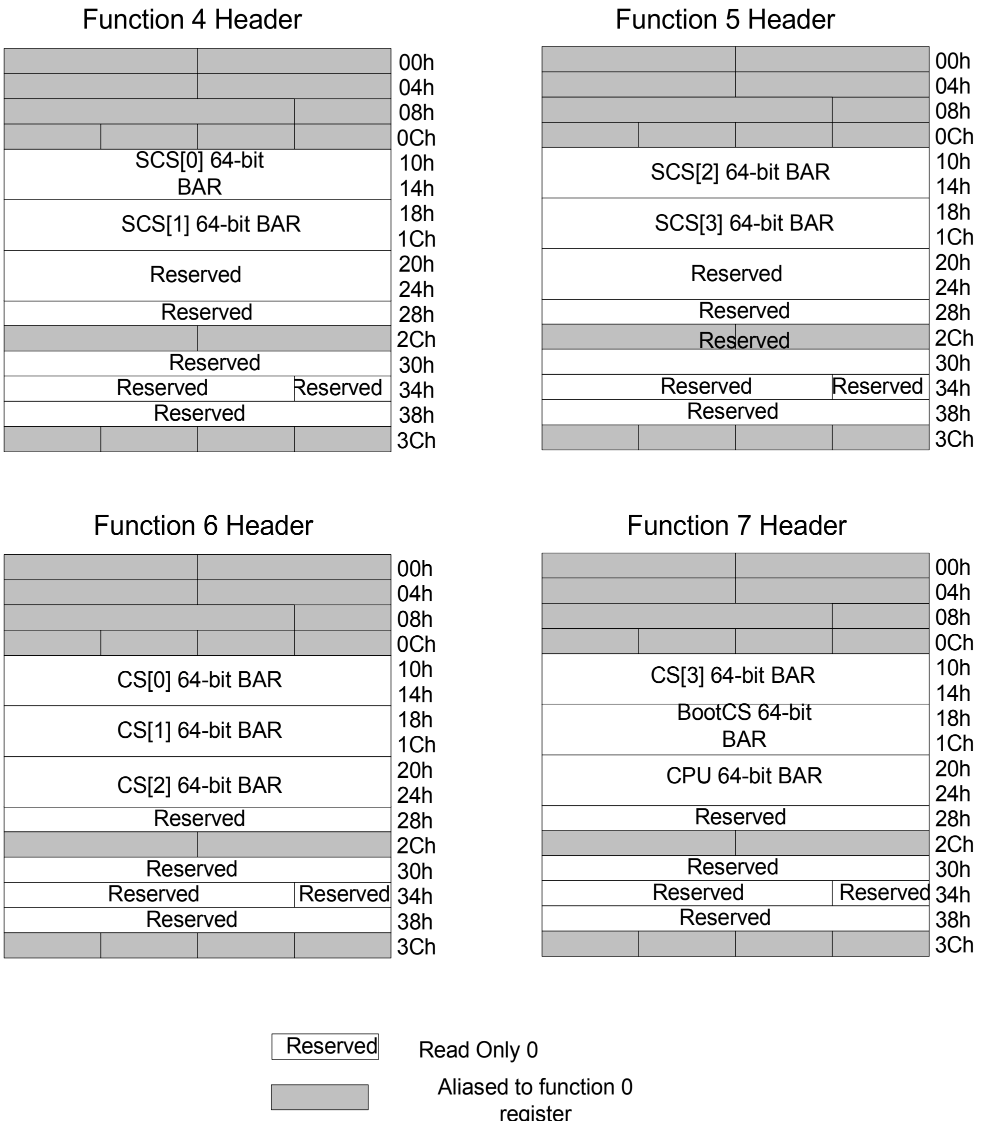
▨		▨		00h
▨		▨		04h
▨			▨	08h
▨	▨	▨	▨	0Ch
Reserved				10h
Reserved				14h
Reserved				18h
CPU BAR				1Ch
Reserved				20h
Reserved				24h
Reserved				28h
▨		▨		2Ch
Reserved				30h
Reserved			Reserved	34h
Reserved				38h
▨	▨	▨	▨	3Ch

Function 3 Header

▨		▨		00h
▨		▨		04h
▨			▨	08h
▨	▨	▨	▨	0Ch
Reserved				10h
Reserved				14h
Reserved				18h
Reserved				1Ch
Reserved				20h
Reserved				24h
Reserved				28h
▨		▨		2Ch
Reserved				30h
Reserved			Reserved	34h
Reserved				38h
▨	▨	▨	▨	3Ch

Reserved	Read Only 0
▨	Aliased to function 0 register

Figure 33: PCI Configuration Space Header[1]



[1] Function 7 CPU 64-bit BAR is a Reserved register

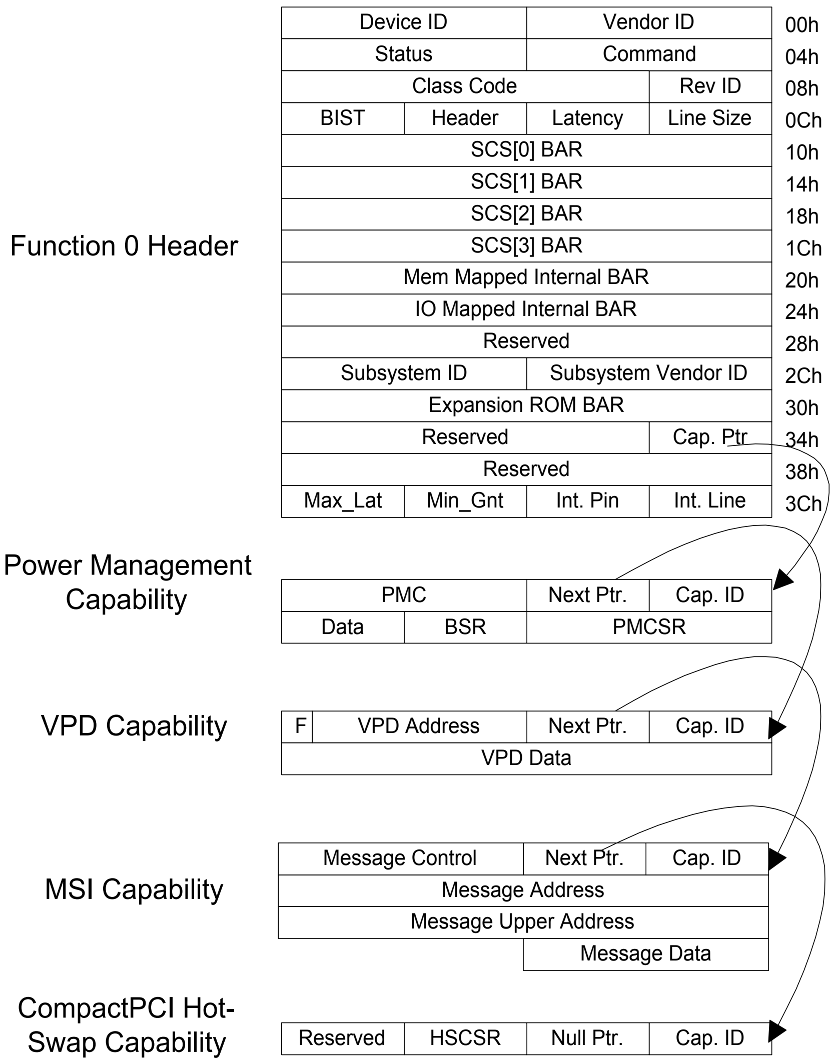
8.17 PCI Special Features

The GT–64242 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.

Figure 34: GT–64242 Capability List



Function 0 Header

Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Rev ID	08h
BIST	Header	Latency	Line Size	0Ch
SCS[0] BAR				10h
SCS[1] BAR				14h
SCS[2] BAR				18h
SCS[3] BAR				1Ch
Mem Mapped Internal BAR				20h
IO Mapped Internal BAR				24h
Reserved				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM BAR				30h
Reserved			Cap. Ptr	34h
Reserved				38h
Max_Lat	Min_Gnt	Int. Pin	Int. Line	3Ch

Power Management Capability

PMC		Next Ptr.	Cap. ID
Data	BSR	PMCSR	

VPD Capability

F	VPD Address		Next Ptr.	Cap. ID
VPD Data				

MSI Capability

Message Control		Next Ptr.	Cap. ID
Message Address			
Message Upper Address			
		Message Data	

CompactPCI Hot-Swap Capability

Reserved	HSCSR	Null Ptr.	Cap. ID

8.17.1 Power Management

The GT–64242 implements the required configuration registers defined by the PCI specification for supporting system Power Management as well as PME* pin.

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 updates the PCI Power Management Control and Status register's Power State bits [1:0] (see Table 255 on page 207), 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–64242 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 PME pin are multiplexed on the GT–64242 MPP pins. If PME* support is required, first program the MPP pins to the appropriate configuration. See Section 15. "MPP Multiplexing" on page 275 for details.

NOTE: The GT–64242 does not support it's own power down. It only supports a software capability to power down the CPU or other on board devices.

8.17.2 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–64242 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–64242 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 192 on page 176). 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–64242 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.

8.17.3 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–64242 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–64242 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–64242 can drive.
- bits[6:4] - Multiple Message Enable. Defines the number of DIFFERENT MSI messages the system allocates for the GT–64242.
- bit[7] - 64-bit address capable. Enables 64-bit addressing messages.

As soon as PCI enables MSI (set MSI enable bit), GT–64242 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–64242 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–64242 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).

8.17.4 Hot Swap

The GT–64242 is CompactPCI Hot-Swap ready compliant. It implements the required configuration registers defined by CompactPCI Hot-Swap specification as well as three required pins.

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–64242 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–64242 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–64242 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–64242 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–64242 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–64242 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–64242 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–64242 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–64242 supports the following hot swap device requirements:

- All PCI outputs floats when RST* is asserted.
- All GT–64242 PCI state machines are kept in their idle state while RST* is asserted.
- The GT–64242 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–64242 has no assumptions on clock behavior prior to it’s setup to the rising edge of RST#.
- The GT–64242 is tolerant of the 1V precharge voltage during insertion.
- The GT–64242 can be powered from Early Vcc.

8.17.5 BIST (Built In Self Test)

The GT–64242 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–64242 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.

8.18 PCI Interface Registers

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 performs a PCI configuration cycle on PCI's Status and Command Register, the register is located at 0x004.

NOTE:

Table 143: PCI Slave Address Decoding Register Map

Register	Offset	Page
PCI SCS[0]* BAR Size	0xc08	page 167
PCI SCS[1]* BAR Size	0xd08	page 167
PCI SCS[2]* BAR Size	0xc0c	page 167
PCI SCS[3]* BAR Size	0xd0c	page 167
PCI CS[0]* BAR Size	0xc10	page 167
PCI CS[1]* BAR Size	0xd10	page 168
PCI CS[2]* BAR Size	0xd18	page 168
PCI CS[3]* BAR Size	0xc14	page 168
PCI Boot CS* BAR Size	0xd14	page 168
PCI DAC SCS[0]* BAR Size	0xe00	page 168
PCI DAC SCS[1]* BAR Size	0xe04	page 168
PCI DAC SCS[2]* BAR Size	0xe08	page 169
PCI DAC SCS[3]* BAR Size	0xe0c	page 169
PCI DAC CS[0]* BAR Size	0xe10	page 169
PCI DAC CS[1]* BAR Size	0xe14	page 169
PCI DAC CS[2]* BAR Size	0xe18	page 169
PCI DAC CS[3]* BAR Size	0xe1c	page 170
PCI DAC Boot CS* BAR Size	0xe20	page 170
PCI Expansion ROM BAR Size	0xd2c	page 170
PCI Base Address Registers' Enable	0xc3c	page 170
PCI SCS[0]* Base Address Remap	0xc48	page 172
PCI SCS[1]* Base Address Remap	0xd48	page 172

Table 143: PCI Slave Address Decoding Register Map (Continued)

Register	Offset	Page
PCI SCS[2]* Base Address Remap	0xc4c	page 173
PCI SCS[3]* Base Address Remap	0xd4c	page 173
PCI CS[0]* Base Address Remap	0xc50	page 173
PCI CS[1]* Base Address Remap	0xd50	page 173
PCI CS[2]* Base Address Remap	0xd58	page 173
PCI CS[3]* Address Remap	0xc54	page 173
PCI Boot CS* Address Remap	0xd54	page 174
PCI DAC SCS[0]* Base Address Remap	0xf00	page 174
PCI DAC SCS[1]* Base Address Remap	0xf04	page 174
PCI DAC SCS[2]* Base Address Remap	0xf08	page 174
PCI DAC SCS[3]* Base Address Remap	0xf0C	page 174
PCI DAC CS[0]* Base Address Remap	0xf10	page 174
PCI DAC CS[1]* Base Address Remap	0xf14	page 175
PCI DAC CS[2]* Base Address Remap	0xf18	page 175
PCI DAC CS[3]* Base Address Remap	0xf1c	page 175
PCI DAC BootCS* Base Address Remap	0xf20	page 175
PCI Expansion ROM Base Address Remap	0xf38	page 175
PCI Address Decode Control	0xd3c	page 176

Table 144: PCI Control Register Map

Register	Offset	Page
PCI Command	0xc00	page 176
PCI Mode	0xd00	page 179
PCI Timeout & Retry	0xc04	page 181
PCI Read Buffer Discard Timer	0xd04	page 181
PCI MSI Trigger Timer	0xc38	page 181
PCI Arbiter Control	0x1d00	page 182
PCI Interface Crossbar Control (Low)	0x1d08	page 184
PCI Interface Crossbar Control (High)	0x1d0c	page 184

Table 144: PCI Control Register Map (Continued)

Register	Offset	Page
PCI Interface Crossbar Timeout **NOTE:** Reserved for Galileo Technology usage.	0x1d04	page 185
PCI Read Response Crossbar Control (Low)	0x1d18	page 186
PCI Read Response Crossbar Control (High)	0x1d1c	page 186
PCI Sync Barrier Virtual Register	0x1d10	page 187
PCI Access Control Base 0 (Low)	0x1e00	page 187
PCI Access Control Base 0 (High)	0x1e04	page 188
PCI Access Control Top 0	0x1f08	page 189
PCI Access Control Base 1 (Low)	0x1e10	page 189
PCI Access Control Base 1 (High)	0x1e14	page 189
PCI Access Control Top 1	0x1e18	page 189
PCI Access Control Base 2 (Low)	0x1e20	page 189
PCI Access Control Base 2 (High)	0x1e24	page 190
PCI Access Control Top 2	0x1e28	page 190
PCI Access Control Base 3 (Low)	0x1e30	page 190
PCI Access Control Base 3 (High)	0x1e34	page 190
PCI Access Control Top 3	0x1e38	page 190
PCI Access Control Base 4 (Low)	0x1e40	page 191
PCI Access Control Base 4 (High)	0x1e44	page 191
PCI Access Control Top 4	0x1e48	page 191
PCI Access Control Base 5 (Low)	0x1e50	page 191
PCI Access Control Base 5 (High)	0x1e54	page 191
PCI Access Control Top 5	0x1e58	page 192
PCI Access Control Base 6 (Low)	0x1e60	page 192
PCI Access Control Base 6 (High)	0x1e64	page 192
PCI Access Control Top 6	0x1e68	page 192
PCI Access Control Base 7 (Low)	0x1e70	page 192
PCI Access Control Base 7 (High)	0x1e74	page 193
PCI Access Control Top 7	0x1e78	page 193

Table 145: PCI Configuration Access Register Map

Register	Offset	Page
PCI Configuration Address	0xcf8	page 193
PCI Configuration Data Virtual Register	0xcfc	page 194
PCI Interrupt Acknowledge Virtual Register	0xc34	page 194

Table 146: PCI Error Report Register Map

Register	Offset	Page
PCI SErr Mask	0xc28	page 194
PCI Error Address (Low)	0x1d40	page 195
PCI Error Address (High)	0x1d44	page 196
PCI Error Data (Low)	0x1d48	page 196
PCI Error Data (High)	0x1d4C	page 196
PCI Error Command	0x1d50	page 196
PCI Error Cause	0x1d58	page 197
PCI Error Mask	0x1d5c	page 199

Table 147: PCI Configuration, Function 0, Register Map

Register	Offset from CPU or PCI	Page
PCI Device and Vendor ID	0x00	page 200
PCI Status and Command	0x04	page 200
PCI Class Code and Revision ID	0x08	page 202
PCI BIST, Header Type, Latency Timer, and Cache Line	0x0c	page 202
PCI SCS[0]* Base Address	0x10	page 203
PCI SCS[1]* Base Address	0x14	page 203
PCI SCS[2]* Base Address	0x18	page 204
PCI SCS[3]* Base Address	0x1c	page 204
PCI Internal Registers Memory Mapped Base Address	0x20	page 204
PCI Internal Registers I/O Mapped Base Address	0x24	page 204
PCI Subsystem ID and Subsystem Vendor ID	0x2c	page 205
PCI Expansion ROM Base Address	0x30	page 205

Table 147: PCI Configuration, Function 0, Register Map (Continued)

Register	Offset from CPU or PCI	Page
PCI Capability List Pointer	0x34	page 205
PCI Interrupt Pin and Line	0x3c	page 206
PCI Power Management Capability	0x40	page 206
PCI Power Management Status and Control	0x44	page 207
PCI VPD Address	0x48	page 208
PCI VPD Data	0x4c	page 208
PCI MSI Message Control	0x50	page 208
PCI MSI Message Address	0x54	page 209
PCI MSI Message Upper Address	0x58	page 209
PCI Message Data	0x5c	page 210
PCI CompactPCI Hot Swap Capability	0x60	page 210

Table 148: PCI Configuration, Function 1, Register Map

Register	Offset from CPU or PCI	Page
PCI CS[0]* Base Address	0x10	page 211
PCI CS[1]* Base Address	0x14	page 211
PCI CS[2]* Base Address	0x18	page 211
PCI CS[3]* Base Address	0x1c	page 211
PCI Boot CS* Base Address	0x20	page 211

Table 149: PCI Configuration, Function 4, Register Map

Register	Offset from CPU or PCI	Page
PCI DAC SCS[0]* Base Address (Low)	0x10	page 212
PCI DAC SCS[0]* Base Address (High)	0x14	page 212
PCI DAC SCS[1]* Base Address (Low)	0x18	page 212
PCI DAC SCS[1]* Base Address (High)	0x1c	page 212

Table 150: PCI Configuration, Function 5, Register Map

Register	Offset from CPU or PCI	Page
PCI DAC SCS[2]* Base Address (Low)	0x10	page 213
PCI DAC SCS[2]* Base Address (High)	0x14	page 213
PCI DAC SCS[3]* Base Address (Low)	0x18	page 213
PCI DAC SCS[3]* Base Address (High)	0x1c	page 213

Table 151: PCI Configuration, Function 6, Register Map

Register	Offset from CPU or PCI_0	Page
PCI DAC CS[0]* Base Address (Low)	0x10	page 213
PCI DAC CS[0]* Base Address (High)	0x14	page 214
PCI DAC CS[1]* Base Address (Low)	0x18	page 214
PCI DAC CS[1]* Base Address (High)	0x1c	page 214
PCI DAC CS[2]* Base Address (Low)	0x20	page 214
PCI DAC CS[2]* Base Address (High)	0x24	page 214

Table 152: PCI Configuration, Function 7, Register Map

Register	Offset from CPU or PCI	Page
PCI DAC CS[3]* Base Address (Low)	0x10	page 215
PCI DAC CS[3]* Base Address (High)	0x14	page 215
PCI DAC Boot CS* Base Address (Low)	0x18	page 215
PCI DAC Boot CS* Base Address (High)	0x1c	page 215

8.18.1 PCI Slave Address Decoding Registers

Table 153: PCI SCS[0] BAR Size

- PCI Offset: 0xc08

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 154: PCI SCS[1]* BAR Size

- PCI Offset: 0xd08

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 155: PCI SCS[2]* BAR Size

- PCI Offset: 0xc0c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 156: PCI SCS[3]* BAR Size

- PCI Offset: 0xd0c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 157: PCI CS[0]* BAR Size

- PCI Offset: 0xc10

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 158: PCI CS[1]* BAR Size

- PCI Offset: 0xd10

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 159: PCI CS[2]* BAR Size

- PCI Offset: 0xd18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x00fff000

Table 160: PCI CS[3]* BAR Size

- PCI Offset: 0xc14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 161: PCI Boot CS* BAR Size

- PCI Offset: 0xd14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 162: PCI DAC SCS[0] BAR Size

- PCI Offset: 0xe00

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 163: PCI DAC SCS[1] BAR Size

- PCI Offset: 0xe04

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 164: PCI DAC SCS[2] BAR Size

- PCI Offset: 0xe08

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 165: PCI DAC SCS[3] BAR Size

- PCI Offset: 0xe0c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 166: PCI DAC CS[0]* BAR Size

- PCI Offset: 0xe10

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 167: PCI DAC CS[1]* BAR Size

- PCI Offset: 0xe14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 168: PCI DAC CS[2]* BAR Size

- PCI Offset: 0xe18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x00fff000

Table 169: PCI DAC CS[3]* BAR Size

- PCI Offset: 0xe1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 170: PCI DAC BootCS* BAR Size

- PCI Offset: 0xe20

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 171: PCI Expansion ROM BAR Size

- PCI Offset: 0xd2c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Bank Size.	0x007ff000

Table 172: PCI Base Address Registers Enable

- PCI Offset: 0xc3c

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
4	CS0En	CS[0]* BAR Enable 0 - Enable 1 - Disable	0x0

Table 172: PCI Base Address Registers Enable (Continued)

- PCI Offset: 0xc3c

Bits	Field Name	Function	Initial Value
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
13:11	Reserved	Reserved.	0x0
14	Reserved	Must be 1.	0x1
15	DSCS0En	DAC SCS[0]* BAR Enable 0 - Enable 1 - Disable	0x1
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

Table 172: PCI Base Address Registers Enable (Continued)

- PCI Offset: 0xc3c

Bits	Field Name	Function	Initial Value
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
25:24	Reserved	Reserved.	0x0
26	Reserved	Must be 1.	0x1
31:27	Reserved	Reserved.	0x1f

1. The GT–64242 prevents disabling both memory mapped and I/O mapped BARs (bits 9 and 10 cannot simultaneously be set to 1).

Table 173: PCI SCS[0]* Base Address Remap

- PCI Offset: 0xc48

Bits	Field Name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	SCS0Remap	SCS[0]* BAR Remap Address	0x0

Table 174: PCI SCS[1]* Base Address Remap

- PCI Offset: 0xd48

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x00800000

Table 175: PCI SCS[2]* Base Address Remap

- PCI Offset: 0xc4c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x01000000

Table 176: PCI SCS[3]* Base Address Remap

- PCI Offset: 0xd4c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x01800000

Table 177: PCI CS[0]* Base Address Remap

- PCI Offset: 0xc50

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1c000000

Table 178: PCI CS[1]* Base Address Remap

- PCI Offset: 0xd50

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1c800000

Table 179: PCI CS[2]* Base Address Remap

- PCI Offset: 0xd58

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1d000000

Table 180: PCI CS[3]* Base Address Remap

- PCI Offset: 0xc54

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1f00000

Table 181: PCI BootCS* Base Address Remap

- PCI Offset: 0xd54

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1f800000

Table 182: PCI DAC SCS[0]* Base Address Remap

- PCI Offset: 0xf00

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x0

Table 183: PCI DAC SCS[1]* Base Address Remap

- PCI Offset: 0xf04

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x00800000

Table 184: PCI DAC SCS[2]* Base Address Remap

- PCI Offset: 0xf08

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x01000000

Table 185: PCI DAC SCS[3]* Base Address Remap

- PCI Offset: 0xf0c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x01800000

Table 186: PCI DAC CS[0]* Base Address Remap

- PCI Offset: 0xf10

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1c000000

Table 187: PCI DAC CS[1]* Base Address Remap

- PCI Offset: 0xf14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1c800000

Table 188: PCI DAC CS[2]* Base Address Remap

- PCI Offset: 0xf18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1d000000

Table 189: PCI DAC CS[3]* Base Address Remap

- PCI Offset: 0xf1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1f000000

Table 190: PCI DAC BootCS* Base Address Remap

- PCI Offset: 0xf20

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1f800000

Table 191: PCI Expansion ROM Base Address Remap

- PCI Offset: 0xf38

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address Remap.	0x1f000000

Table 192: PCI Address Decode Control

- PCI Offset: 0xd3c

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
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–64242 internal space.	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

8.18.2 PCI Control Registers

Table 193: PCI Command

- PCI Offset: 0xc00

Bits	Field Name	Function	Initial Value
0	MByteSwap	PCI Master Byte Swap When set to '0', the GT–64242 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

Table 193: PCI Command (Continued)

- PCI Offset: 0xc00

Bits	Field Name	Function	Initial Value
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
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–64242 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–64242 PCI slave swaps the 32-bit words of the incoming and outgoing PCI data. **NOTE:** GT-64120 and GT-64130 compatible.	0x0
12	IntBusCtl	PCI Interface Unit Internal Bus Control 0 - Enable internal bus sharing between master and slave interfaces. 1 - Disable internal bus sharing between master and slave interfaces. **NOTE:** Reserved for Galileo Technology usage.	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

Table 193: PCI Command (Continued)

- PCI Offset: 0xc00

Bits	Field Name	Function	Initial Value
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–64242 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]
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

Table 193: PCI Command (Continued)

- PCI Offset: 0xc00

Bits	Field Name	Function	Initial Value
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
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–64242 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			0x0
31:28	Reserved	Read only.	0x0

Table 194: PCI Mode

- PCI Offset: 0xd00

Bits	Field Name	Function	Initial Value
0	PciID	PCI Interface ID Read Only 0.	PCI: 0x0
1	Reserved	Reserved.	0x0

Table 194: PCI Mode (Continued)

- PCI Offset: 0xd00

Bits	Field Name	Function	Initial Value
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.
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: AD[17]
9	VPD	VPD Enable When set to ‘1’, VPD is supported. Read Only from PCI.	Sampled at reset. PCI: AD[21]
10	MSI	MSI Enable When set to ‘1’, MSI is supported. Read Only from PCI.	Sampled at reset. PCI: AD[23]
11	PMG	Power Management Enable When set to ‘1’, PMG is supported. Read Only from PCI.	Sampled at reset. PCI: AD[19]
12	HotSwap	CompactPCI Hot Swap Enable When set to ‘1’, HotSwap is supported. Read Only from PCI.	Sampled at reset. PCI: AD[25]
13	BIST	BIST Enable If set to ‘1’, BIST is enabled. Read only from PCI.	Sampled at reset. PCI: AD[26]
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 195: PCI Timeout and Retry

- PCI Offset: 0xc04

Bits	Field Name	Function	Initial Value
7:0	Timeout0	Specifies the number of PClk cycles the GT–64242 slave holds the PCI bus before terminating a transaction with RETRY.	0x0f
15:8	Timeout1	Specifies the number of PClk cycles the GT–64242 slave holds the PCI bus before terminating a transaction with DISCONNECT.	0x07
23:16	RetryCtr	Retry Counter Specifies the number of retries of the GT–64242 Master. The GT–64242 generates an interrupt when this timer expires. A 0x00 value means a “retry forever”.	0x0
31:24	Reserved	Reserved.	0x0

Table 196: PCI Read Buffer Discard Timer

- PCI Offset: 0xd04

Bits	Field Name	Function	Initial Value
15:0	Timer	Specifies the number of PClk cycles the GT–64242 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 197: PCI MSI Trigger Timer

- PCI Offset: 0xc38

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 198: PCI Arbiter Control

- PCI Offset: 0x1d00

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	PAEn	Priority Arbitration Enable 0 - Low priority requests are granted only when no high priority request is pending 1 - Weighted round robin arbitration is performed between high priority and low priority groups. **NOTE:** If HPPV (bits [28:21]) is set to ‘0’ and PAEn is set to ‘1’, priority scheme is reversed. This means that high priority requests are granted if no low priority request is pending.	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–64242 PCI master), since they require GNT* assertion for two cycles.	0x0

Table 198: PCI Arbiter Control (Continued)

- PCI Offset: 0x1d00

Bits	Field Name	Function	Initial Value
13:7	P[6:0]	Priority These bits assign priority levels to the requests connected to the PCI arbiter. When a PM bit is set to '1', priority of the associated request is high. The mapping between P[6:0] bits and the request/grant pairs are as follows: • P[0] - internal PCI master • P[1] - external REQ0/GNT0 • P[2] - external REQ1/GNT1 • P[3] - external REQ2/GNT2 • P[4] - external REQ3/GNT3 • P[5] - external REQ4/GNT4 • P[6] - external REQ5/GNT5	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
28:21	HPPV	High Priority Preset Value This is the preset value of the high priority counter (High_cnt). This counter decrements each time a high priority request is granted. When the counter reaches zero, it reloads with this preset value. The counter reloads when a low priority request is granted.	0x0
30:29	Reserved	Reserved.	0x0
31	EN	Enable Setting this bit to '1' enables operation of the arbiter.	0x0

Table 199: PCI Interface Crossbar Control (Low)

- PCI Offset: 0x1d08

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: NULL request 0x4 - Reserved 0x5 - Comm unit access 0x6 - IDMA channels 0/1/2/3 access 0x7 - 0xf - Reserved	0x2
7:4	Arb1	Slice 1 of PCI master “pizza” arbiter.	0x4
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.	0x3
27:24	Arb6	Slice 6 of PCI master “pizza” arbiter.	0x3
31:28	Arb7	Slice 7 of PCI master “pizza” arbiter.	0x3

Table 200: PCI Interface Crossbar Control (High)

- PCI Offset: 0x1d0c

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.	0x4
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
23:20	Arb13	Slice 13 of PCI master “pizza” arbiter.	0x3
27:24	Arb14	Slice 14 of PCI master “pizza” arbiter.	0x3

Table 200: PCI Interface Crossbar Control (High) (Continued)

- PCI Offset: 0x1d0c

Bits	Field Name	Function	Initial Value
31:28	Arb15	Slice 15 of PCI master “pizza” arbiter.	0x3

Table 201: PCI Interface Crossbar Timeout

- PCI Offset: 0x1d04

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 202: PCI Read Response Crossbar Control (Low)

- PCI Offset: 0x1d18

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: NULL 0x4 - Reserved 0x5 - Comm unit internal registers read data 0x6 - IDMA 0/1/2/3 internal registers read data 0x7 - 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.	0x4
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.	0x3

Table 203: PCI Read Response Crossbar Control (High)

- PCI Offset: 0x1d1c

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.	0x4
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.	0x3

Table 204: PCI Sync Barrier Virtual Register

- PCI Offset: 0x1d10

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 205: PCI Access Control Base 0 (Low)

- PCI Offset: 0x1e00

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
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

Table 205: PCI Access Control Base 0 (Low) (Continued)

- PCI Offset: 0x1e00

Bits	Field Name	Function	Initial Value
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 206: PCI Access Control Base 0 (High)

- PCI Offset: 0x1e04

Bits	Field Name	Function	Initial Value
31:0	Addr	Base Address High Corresponds to address bits[63:32].	0x0

Table 207: PCI Access Control Top 0

- PCI Offset: 0x1e08

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

Table 208: PCI Access Control Base 1 (Low)

- PCI Offset: 0x1e10

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 205 on page 187	0x1001fff

Table 209: PCI Access Control Base 1 (High)

- PCI Offset: 0x1e14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 206 on page 188.	0x0

Table 210: PCI Access Control Top 1

- PCI Offset: 0x1e18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base Top 0. See Table 207 on page 189.	0x0

Table 211: PCI Access Control Base 2 (Low)

- PCI Offset: 0x1e20

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 205 on page 187.	0x1001fff

Table 212: PCI Access Control Base 2 (High)

- PCI Offset: 0x1e24
-

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 206 on page 188.	0x0

Table 213: PCI Access Control Top 2

- PCI Offset: 0x1e28

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0. See Table 207 on page 189.	0x0

Table 214: PCI Access Control Base 3 (Low)

- PCI Offset: 0x1e30

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 205 on page 187.	0x1001fff

Table 215: PCI Access Control Base 3 (High)

- PCI Offset: 0x1e34

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 206 on page 188.	0x0

Table 216: PCI Access Control Top 3

- PCI Offset: 0x1e38

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0. See Table 207 on page 189.	0x0

Table 217: PCI Access Control Base 4 (Low)

- PCI Offset: 0x1e40

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 205 on page 187.	0x1001fff

Table 218: PCI Access Control Base 4 (High)

- PCI Offset: 0x1e44

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 206 on page 188.	0x0

Table 219: PCI Access Control Top 4

- PCI Offset: 0x1e48

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0. See Table 207 on page 189.	0x0

Table 220: PCI Access Control Base 5 (Low)

- PCI Offset: 0x1e50

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 205 on page 187.	0x1001fff

Table 221: PCI Access Control Base 5 (High)

- PCI Offset: 0x1e54

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 206 on page 188.	0x0

Table 222: PCI Access Control Top 5

- PCI Offset: 0x1e58

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0. See Table 207 on page 189.	0x0

Table 223: PCI Access Control Base 6 (Low)

- PCI Offset: 0x1e60

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 205 on page 187.	0x1001fff

Table 224: PCI Access Control Base 6 (High)

- PCI Offset: 0x1e64

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 206 on page 188.	0x0

Table 225: PCI Access Control Top 6

- PCI Offset: 0x1e68

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0. See Table 207 on page 189.	0x0

Table 226: PCI Access Control Base 7 (Low)

- PCI Offset: 0x1e70

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (Low). See Table 205 on page 187.	0x1001fff

Table 227: PCI Access Control Base 7 (High)

- PCI Offset: 0x1e74

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0 (High). See Table 206 on page 188.	0x0

Table 228: PCI Access Control Top 7

- PCI Offset: 0x1e78

Bits	Field Name	Function	Initial Value
31:0	Various	Same as in Access Control Base 0. See Table 207 on page 189.	0x0

8.18.3 PCI Configuration Access Registers

Table 229: PCI Configuration Address

- PCI Offset: 0xcf8

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 230: PCI Configuration Data

- PCI Offset: 0xcfc

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–64242 to perform a Configuration or Special cycle on the PCI bus.	0x000

Table 231: PCI Interrupt Acknowledge

- PCI Offset: 0xc34

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

8.18.4 PCI Error Report Registers

Table 232: PCI SERR* Mask

- PCI Offset: 0xc28

NOTE: The GT–64242 asserts SERR* only if SERR* is enabled via the PCI Status and Command register, see Table 241 on page 200.

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
6	MRdPerr	If set to ‘1’, asserts SERR* upon bad data parity detection during a PCI master read transaction.	0x0
7	Reserved	Reserved.	0x0

Table 232: PCI SERR* Mask (Continued)

- PCI Offset: 0xc28

NOTE: The GT–64242 asserts SERR* only if SERR* is enabled via the PCI Status and Command register, see Table 241 on page 200.

Bits	Field Name	Function	Initial Value
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 233: PCI Error Address (Low)

- PCI Offset: 0x1d40

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 234: PCI Error Address (High)

- PCI Offset: 0x1d44

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 235: PCI Error Data (Low)

- PCI Offset: 0x1d48

Bits	Field Name	Function	Initial Value
31:0	ErrData	PCI data bits [31:0] are latched upon error condition.	0x0

Table 236: PCI Error Data (High)

- PCI Offset: 0x1d4c

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 237: PCI Error Command

- PCI Offset: 0x1d50

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 238: PCI Interrupt Cause [1,2]

- PCI Offset: 0x1d58

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
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 238: **PCI Interrupt Cause** (Continued)[1,2]

- PCI Offset: 0x1d58

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 239: PCI Error Mask

- PCI Offset: 0x1d5c

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

8.18.5 Function 0 Configuration Registers

Table 240: PCI Device and Vendor ID

- PCI Offset from CPU or PCI: 0x00

Bits	Field Name	Function	Initial Value
15:0	VenID	Galileo's Vendor ID. Read only from PCI.	0x11ab
31:16	DevID	GT–64242 Device ID. Read only from PCI.	0x6430

Table 241: PCI Status and Command

- PCI Offset from CPU or PCI: 0x04

Bits	Field Name	Function	Initial Value
0	IOEn	Controls the GT–64242's ability to response to PCI I/O accesses. 0 - Disable 1 - Enable	0x0
1	MEMEn	Controls the GT–64242's ability to response to PCI Memory accesses. 0 - Disable 1 - Enable	0x0
2	MasEn	Controls the GT–64242's ability to act as a master on the PCI bus. 0 - Disable 1 - Enable	0x0
3	SpecialEn	Controls the GT–64242's ability to respond to PCI special cycles. Read only 0 (GT–64242 PCI slave does not support special cycles).	0x0
4	MemWrInv	Controls the GT–64242'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

Table 241: PCI Status and Command (Continued)

- PCI Offset from CPU or PCI: 0x04

Bits	Field Name	Function	Initial Value
6	PErrEn	Controls the GT–64242'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–64242 PCI master performs address stepping only on configuration accesses. Read only from the PCI.	0x0
8	SErrEn	Controls the GT–64242's ability to assert the SErr* pin. 0 - Disable 1 - Enable	0x0
9	FastBTBEn	Controls the GT–64242'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–64242 configuration header includes capability list. Read only from the PCI.	Reset initialization
21	66MHzEn	66MHz Capable The GT–64242 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–64242 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–64242 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–64242's DevSel timing (medium). Read only from the PCI.	0x1
27	SlaveTabort	Set when the GT–64242's slave terminates a transaction with Target Abort. Clear only by writing 1.	0x0

Table 241: PCI Status and Command (Continued)

- PCI Offset from CPU or PCI: 0x04

Bits	Field Name	Function	Initial Value
28	MasterTabort	Set when the GT–64242's master detects a Target Abort termination. Clear only by writing 1.	0x0
29	MAbort	Set when the GT–64242's master generates a Master Abort (except of special cycle). Clear only by writing 1.	0x0
30	SysErr	Set when the GT–64242 asserts SERR*. Clear only by writing 1.	0x0
31	DetParErr	Set upon the GT–64242 detection of Parity error (both as master and slave). Clear only by writing 1.	0x0

Table 242: PCI Class Code and Revision ID

- PCI Offset from CPU or PCI: 0x08

Bits	Field Name	Function	Initial Value
7:0	RevID	Indicates the GT–64242 Revision number. Read only from PCI.	0x1
15:8	Reserved	Read only.	0x0
23:16	SubClass	Indicates the GT–64242 Subclass. Read only from PCI.	0x80
31:24	BaseClass	Indicates the GT–64242 Base Class. Read only from PCI.	0x05

Table 243: PCI BIST, Header Type, Latency Timer, and Cache Line

- PCI Offset from CPU or PCI: 0x0c

Bits	Field Name	Function	Initial Value
7:0	CacheLine	Specifies the GT–64242's cache line size.	0x00
15:8	LatTimer	Specifies in units of PCI bus clocks the latency timer value of the GT–64242.	0x00
23:16	HeadType	Specifies Configuration Header Type Read only from PCI.	0x80

Table 243: PCI BIST, Header Type, Latency Timer, and Cache Line (Continued)

- PCI Offset from CPU or PCI: 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.	Sampled at reset. AD[26]

Table 244: PCI SCS[0]* Base Address

- PCI Offset from CPU or PCI: 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 245: PCI SCS[1]* Base Address

- PCI Offset from CPU or PCI: 0x14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 244 on page 203.	0x00800008

Table 246: PCI SCS[2]* Base Address

- PCI Offset from CPU or PCI: 0x18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 244 on page 203.	0x01000008

Table 247: PCI SCS[3]* Base Address

- PCI Offset from CPU or PCI: 0x1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 244 on page 203.	0x01800008

Table 248: PCI Internal Registers Memory Mapped Base Address

- PCI Offset from CPU or PCI: 0x20

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.	0x0
15:4	Reserved	Read only.	0x0
31:16	Base	Base Address	0x1400

Table 249: PCI Internal Registers I/O Mapped Base Address

- PCI Offset from CPU or PCI: 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

Table 249: PCI Internal Registers I/O Mapped Base Address (Continued)

- PCI Offset from CPU or PCI: 0x24

Bits	Field Name	Function	Initial Value
15:4	Reserved	Read only.	0x0
31:16	Base	Base Address	0x1400

Table 250: PCI Subsystem Device and Vendor ID

- PCI Offset from CPU or PCI: 0x2c

Bits	Field Name	Function	Initial Value
15:0	VenID	Subsystem Manufacturer ID Number	0x0
31:16	DevID	Subsystem Device ID Number	0x0

Table 251: PCI Expansion ROM Base Address Register

- PCI Offset from CPU or PCI: 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 252: PCI Capability List Pointer Register

- PCI Offset from CPU or PCI: 0x34

Bits	Field Name	Function	Initial Value
7:0	CapPtr	Capability List Pointer Read only.	Reset initialization
31:8	Reserved	Reserved.	0x0

Table 253: PCI Interrupt Pin and Line

- PCI Offset from CPU or PCI: 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–64242. Read only from PCI.	0x1
31:16	Reserved	Read only.	0x0

Table 254: PCI Power Management Capability

- PCI Offset from CPU or PCI: 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.	Reset initialization
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–64242 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
26	D2Sup	D2 Power Management State Support Read only from PCI. 0 - Not supported 1 - Supported	0x1

Table 254: PCI Power Management Capability (Continued)

- PCI Offset from CPU or PCI: 0x40

Bits	Field Name	Function	Initial Value
31:27	PMESup	PME* Signal Support Indicates in which power states the GT–64242 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 255: PCI Power Management Control and Status Register

- PCI Offset from CPU or PCI: 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–64242 asserts PME* pin.	0x0
23:16	Reserved	Reserved.	0x0
31:24	Data	State Data Read only from PCI.	0x0

Table 256: PCI VPD Address

- PCI Offset from CPU or PCI: 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	Reset initialization
30:16	Addr	VPD Address Points to the location of the VPD structure in memory. **NOTE:** The GT–64242 also implements remapping of the high address bits through the PCI Address Decoding Control register.	0x0
31	Flag	Flag Flipped by System or GT–64242 during VPD Access On VPD writes, system sets the flag to ‘1’ indicating VPD write is required. The GT–64242 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–64242 sets the flag to ‘1’ when the read is done (data has been read from memory and put in VPD Data register).	0x0

Table 257: PCI VPD Data

- PCI Offset from CPU or PCI: 0x4c

Bits	Field Name	Function	Initial Value
31:0	Data	VPD Data	0x0

Table 258: PCI MSI Message Control

- PCI Offset from CPU or PCI: 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.	Reset initialization

Table 258: PCI MSI Message Control (Continued)

- PCI Offset from CPU or PCI: 0x50
-

Bits	Field Name	Function	Initial Value
16	MSIEn	MSI Enable 0 - Disable The GT–64242 generates a PCI interrupt. 1 - Enabled The GT–64242 generates MSI messages instead of interrupts.	0x0
19:17	MultiCap	Multiple Messages Capable The GT–64242 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–64242 (must be smaller or equal to MultiCap).	0x0
23	Addr64	64-bit Addressing Capable Indicates whether the GT–64242 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 259: PCI MSI Message Address

- PCI Offset from CPU or PCI: 0x54

Bits	Field Name	Function	Initial Value
31:0	Addr	Message Address	0x0

Table 260: PCI MSI Message Upper Address

- PCI Offset from CPU or PCI: 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–64242 issues MSI message as DAC cycle.	0x0

Table 261: PCI MSI Data Control

- PCI Offset from CPU or PCI: 0x5c

Bits	Field Name	Function	Initial Value
15:0	Data	Message Data	0x0
31:16	Reserved	Read only 0.	0x0

Table 262: PCI CompactPCI HotSwap Capability

- PCI Offset from CPU or PCI: 0x60

NOTE: CompactPCI Hot Swap is only supported on the PCI 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

8.18.6 Function 1 Configuration Registers

Table 263: PCI CS[0]* Base Address

- PCI Offset from CPU or PCI: 0x10

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 244 on page 203.	0x1c000000

Table 264: PCI CS[1]* Base Address

- PCI Offset from CPU or PCI: 0x14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 244 on page 203.	0x1c800000

Table 265: PCI CS[2]* Base Address

- PCI Offset from CPU or PCI: 0x18

Bits	Field Name	Function	Initial Value
31:12	Various	Same as SCS[0]* Base Address. See Table 244 on page 203.	0x1d000000

Table 266: PCI CS[3]* Base Address

- PCI Offset from CPU or PCI: 0x1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 244 on page 203.	0x1f000000

Table 267: PCI Boot CS* Base Address

- PCI Offset from CPU or PCI: 0x20
-

Bits	Field Name	Function	Initial Value
31:0	Various	Same as SCS[0]* Base Address. See Table 244 on page 203.	0x1f800000

8.18.7 Function 2 Configuration Registers

8.18.8 Function 4 Configuration Registers

Table 268: PCI DAC SCS[0]* Base Address (Low)

- PCI Offset from CPU or PCI: 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 269: PCI DAC SCS[0]* Base Address (High)

- PCI Offset from CPU or PCI: 0x14

Bits	Field Name	Function	Initial Value
31:0	BaseHigh	Base High Address	0x0

Table 270: PCI DAC SCS[1]* Base Address (Low)

- PCI Offset from CPU or PCI: 0x18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x0080000c

Table 271: PCI DAC SCS[1]* Base Address (High)

- PCI Offset from CPU or PCI: 0x1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address.	0x0

8.18.9 Function 5 Configuration Registers

Table 272: PCI DAC SCS[2]* Base Address (Low)

- PCI Offset from CPU or PCI: 0x10

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x0100000c

Table 273: PCI DAC SCS[2]* Base Address (High)

- PCI Offset from CPU or PCI: 0x14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x0

Table 274: PCI DAC SCS[3]* Base Address (Low)

- PCI Offset from CPU or PCI: 0x18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x0180000c

Table 275: PCI DAC SCS[3]* Base Address (High)

- PCI Offset from CPU or PCI: 0x1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x0

8.18.10 Function 6 Configuration Registers

Table 276: PCI DAC CS[0]* Base Address (Low)

- PCI Offset from CPU or PCI: 0x10

Bits	Field Name	Function	Initial Value
31:12	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x1c000004

Table 277: PCI DAC CS[0]* Base Address (High)

- PCI Offset from CPU or PCI: 0x14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x0

Table 278: PCI DAC CS[1]* Base Address (Low)

- PCI Offset from CPU or PCI: 0x18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x1c800004

Table 279: PCI DAC CS[1]* Base Address (High)

- PCI Offset from CPU or PCI: 0x1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x0

Table 280: PCI DAC CS[2]* Base Address (Low)

- PCI Offset from CPU or PCI: 0x20

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x1d000004

Table 281: PCI DAC CS[2]* Base Address (High)

- PCI Offset from CPU or PCI: 0x24

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x0

8.18.11 Function 7 Configuration Registers

Table 282: PCI DAC CS[3]* Base Address (Low)

- PCI Offset from CPU or PCI: 0x10

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x1f000004

Table 283: PCI DAC CS[3]* Base Address (High)

- PCI Offset from CPU or PCI: 0x14

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x0

Table 284: PCI DAC BootCS* Base Address (Low)

- PCI Offset from CPU or PCI: 0x18

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x1f800004

Table 285: PCI DAC BootCS* Base Address (High)

- PCI Offset from CPU or PCI: 0x1c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as DAC SCS[0]* Base Address. See Table 182 on page 174.	0x0

9. Messaging Unit

The GT–64242 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–64242 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–64242 and a MIPS processor, for example.) This same message passing mechanism may be used to pass messages between peers in a system.

The GT–64242 Messaging Unit is implemented in both PCI interfaces. It allows for messaging between the CPU and PCI and inter-PCI interfaces messaging.

The GT–64242 Messaging Unit registers are accessible from the PCI through the GT–64242 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 8.7 "PCI Target Operation" on page 143.

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 301 on page 229. 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–64242 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–64242's local CPU and received by an external PCI agent.
- Inbound messages sent by an external PCI bus agent and received by the GT–64242'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.

9.2 Doorbell Registers

The GT–64242 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–64242'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').

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’).

9.3 Circular Queues

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–64242. There are two inbound and two outbound circular queues in the Messaging Unit (MU).

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 286.

Table 286: 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–64242 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.

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–64242. Instead, you are reading and writing pointers into the circular queues (located in SDRAM or Device) controlled by the GT–64242. 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$_2$O Circular Queue Operation

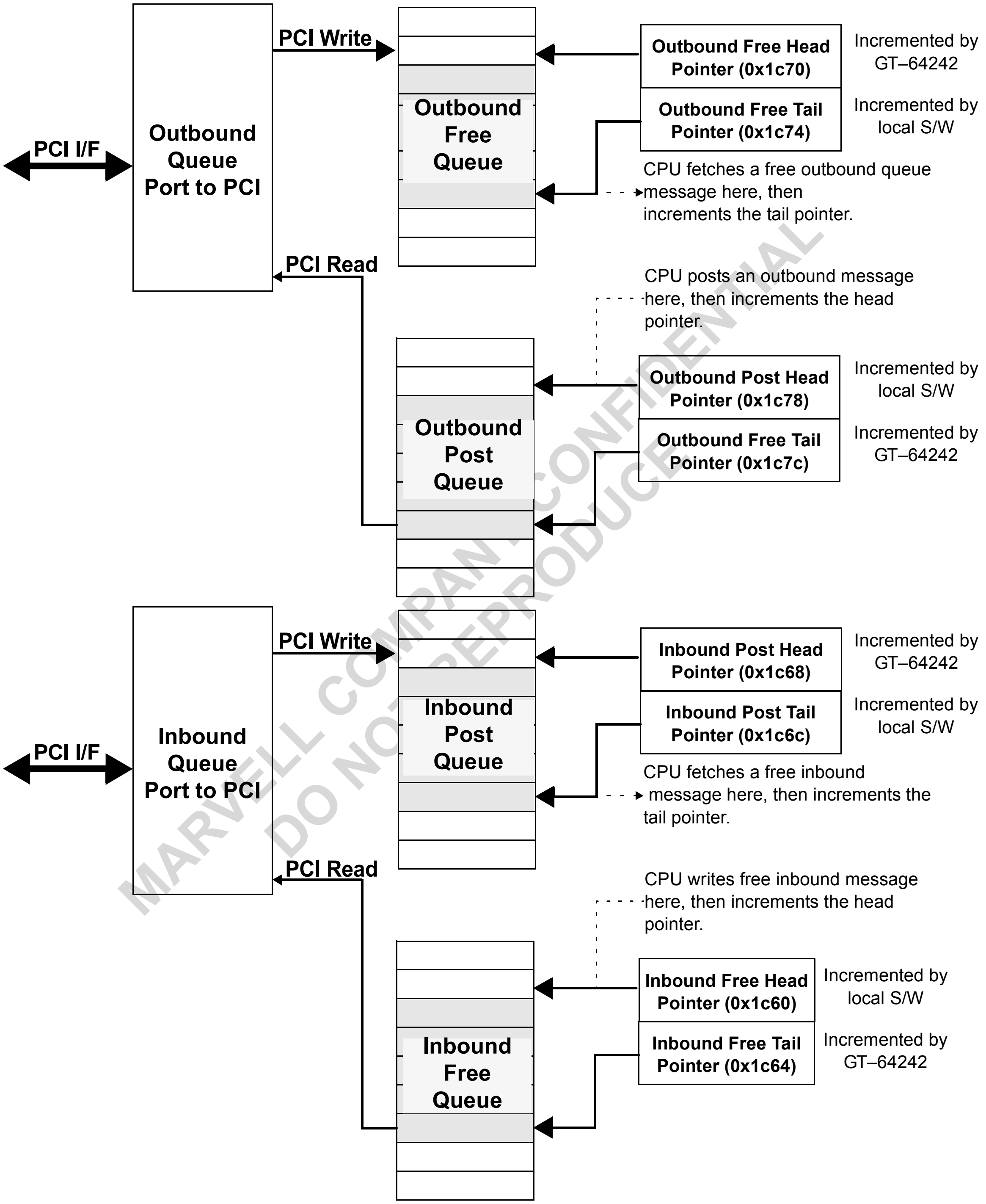

PCI I/F
Outbound Queue Port to PCI
PCI Write
PCI Read
Outbound Free Queue
Outbound Free Head Pointer (0x1c70)
Incremented by GT–64242
Outbound Free Tail Pointer (0x1c74)
Incremented by local S/W
CPU fetches a free outbound queue message here, then increments the tail pointer.
CPU posts an outbound message here, then increments the head pointer.
Outbound Post Queue
Outbound Post Head Pointer (0x1c78)
Incremented by local S/W
Outbound Free Tail Pointer (0x1c7c)
Incremented by GT–64242
PCI I/F
Inbound Queue Port to PCI
PCI Write
PCI Read
Inbound Post Queue
Inbound Post Head Pointer (0x1c68)
Incremented by GT–64242
Inbound Post Tail Pointer (0x1c6c)
Incremented by local S/W
CPU fetches a free inbound message here, then increments the tail pointer.
CPU writes free inbound message here, then increments the head pointer.
Inbound Free Queue
Inbound Free Head Pointer (0x1c60)
Incremented by local S/W
Inbound Free Tail Pointer (0x1c64)
Incremented by GT–64242

Table 287: 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–64242	CPU
Inbound Free		Yes, when queue is full.	No	CPU	GT–64242
Out-bound Post	Out-bound Queue Port	Yes, when queue is not empty.	No	CPU	GT–64242
Out-bound Free		No	Yes, when queue is full	GT–64242	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–64242 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–64242 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–64242 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–64242; 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–64242 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–64242 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–64242 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 8.12 "Data Endianess" on page 149.

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–64242 internal registers space.

If the register is accessed from PCI through the SCS[0] BAR space, remove the offset's 0x1c prefix. For example, in the SCS[0] BAR space, the PCI Outbound Interrupt Cause register is located at offset 0x30.

Table 288: Messaging Unit Register Map

Register	Offset	Page
Inbound Message Register 0	0x1c10	page 225
Inbound Message Register 1	0x1c14	page 225
Outbound Message Register 0	0x1c18	page 225
Outbound Message Register 1	0x1c1c	page 226
Inbound Doorbell Register	0x1c20	page 226
Inbound Interrupt Cause Register	0x1c24	page 226
Inbound Interrupt Mask Register	0x1c28	page 227
Outbound Doorbell Register	0x1c2c	page 227
Outbound Interrupt Cause Register	0x1c30	page 228
Outbound Interrupt Mask Register	0x1c34	page 228
Inbound Queue Port Virtual Register	0x1c40	page 229
Outbound Queue Port Virtual Register	0x1c44	page 229
Queue Control Register	0x1c50	page 229
Queue Base Address Register	0x1c54	page 230
Inbound Free Head Pointer Register	0x1c60	page 231

Table 288: Messaging Unit Register Map (Continued)

Register	Offset	Page
Inbound Free Tail Pointer Register	0x1c64	page 231
Inbound Post Head Pointer Register	0x1c68	page 231
Inbound Post Tail Pointer Register	0x1c6c	page 232
Outbound Free Head Pointer Register	0x1c70	page 232
Outbound Free Tail Pointer Register	0x1c74	page 232
Outbound Post Head Pointer Register	0x1cf8	page 233
Outbound Post Tail Pointer Register	0x1cfc	page 233

Table 289: Inbound Message0

- PCI Offset: 0x1c10

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 290: Inbound Message1

- PCI Offset: 0x1c14

Bits	Field Name	Function	Initial Value
31:0	InMsg1	Same as Inbound Message0.	0x0

Table 291: Outbound Message0

- PCI Offset: 0x1c18

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 292: Outbound Message1

- PCI Offset: 0x1c1

Bits	Field Name	Function	Initial Value
31:0	OutMsg1	Same as Outbound Message0.	0x0

Table 293: Inbound Doorbell

- PCI Offset: 0x1c20

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 294: Inbound Interrupt Cause

- PCI Offset: 0x1c24

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
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

Table 294: Inbound Interrupt Cause (Continued)

- PCI Offset: 0x1c24

Bits	Field Name	Function	Initial Value
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 295: Inbound Interrupt Mask

- PCI Offset: 0x1c28

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
31:24	Reserved	Reserved.	0x0

Table 296: Outbound Doorbell

- PCI Offset: 0x1c2c

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 deassert the interrupt.	0x0

Table 297: Outbound Interrupt Cause

- PCI Offset: 0x1c30

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
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 298: Outbound Interrupt Mask Register

- PCI Offset: 0x1c34

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

Table 298: Outbound Interrupt Mask Register (Continued)

- PCI Offset: 0x1c34

Bits	Field Name	Function	Initial Value
17	OutDoorH	If set to '1', Outbound Doorbell 31:16] interrupt is enabled.	0x1
31:18	Reserved	Reserved.	0x0

Table 299: Inbound Queue Port Virtual Register

- PCI Offset: 0x1c40

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 300: Outbound Queue Port Virtual Register

- PCI Offset: 0x1c44

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 301: Queue Control

- PCI Offset: 0x1c50

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

Table 301: Queue Control (Continued)

- PCI Offset: 0x1c50

Bits	Field Name	Function	Initial Value
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
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	Polarity	Polarity Select 0 - Inbound and Outbound Mask register bits are active high (1 means that interrupt is masked), Inbound and Outbound Doorbell registers bits toggle when writing 1, Inbound and Outbound Interrupt Cause registers bits are cleared by writing ’1’. 1 - Inbound and Outbound Mask register bits are active low (0 means that interrupt is masked), Inbound and Outbound Doorbell registers bits toggle when writing 0, Inbound and Outbound Interrupt Cause registers bits are cleared by writing ’0’.	0x0
31:9	Reserved		0x0

Table 302: Queue Base Address Register

- PCI Offset: 0x1c54

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 303: Inbound Free Head Pointer Register

- PCI Offset: 0x1c60

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 304: Inbound Free Tail Pointer Register

- PCI Offset: 0x1c64

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–64242 after the PCI read from the Inbound port.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

Table 305: Inbound Post Head Pointer Register

- PCI Offset: 0x1c68

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–64242 after the PCI write to the Inbound port.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

Table 306: Inbound Post Tail Pointer Register

- PCI Offset: 0x1c6c

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
31:20	QBAR	Queue Base Address Register Read only.	0x0

Table 307: Outbound Free Head Pointer Register

- PCI Offset: 0x1c70

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–64242 after the PCI write to the Outbound port.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

Table 308: Outbound Free Tail Pointer Register

- PCI Offset: 0x1c74

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 309: Outbound Post Head Pointer Register

- PCI Offset: 0x1c78

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 310: Outbound Post Tail Pointer Register

- PCI Offset: 0x1c7c

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–64242 after the PCI read from the Outbound port.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

10. IDMA CONTROLLER

The GT–64242 has four 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, or PCI. 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 the PCI 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 a 2Kbyte buffer. The buffer is coupled to four IDMA channels - channels 0-3. 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 the PCI to Device 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 243.

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

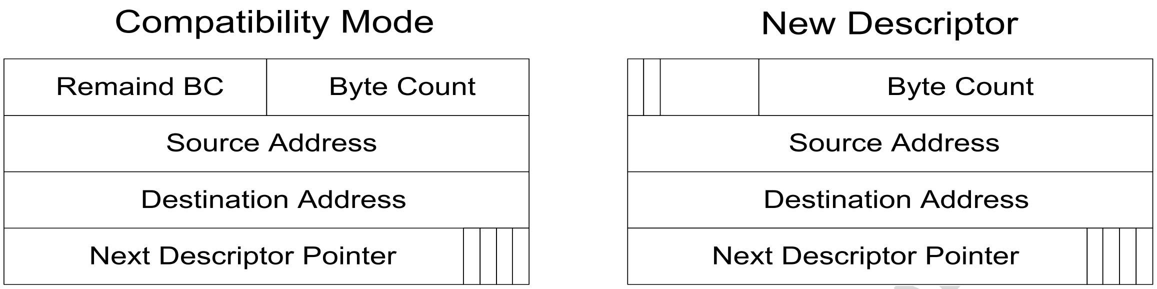


Table 311: DMA Descriptor Definitions

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 252.
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 242.

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 238shows 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 33.

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 75.

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 252.

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 348 on page 252.

10.5.2 Burst Limit

The whole IDMA byte count is chopped into small bursts.

The burst limit can vary from 1 to 128 bytes in modulo-2 steps (i.e. 1, 2, 4, 8..., 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–64242 device bus (see Section 7.3 "Data Pack/ Unpack and Burst Support" on page 124).

If an IDMA accesses a chache coherent DRAM regions, the burst limit must not exceed 32 bytes.

The Burst Limit must be smaller than the IDMA byte count.

If the Channel Control (High) register's BLMode bit [31] (see Table 349 on page 255) 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 348 on page 252). 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.

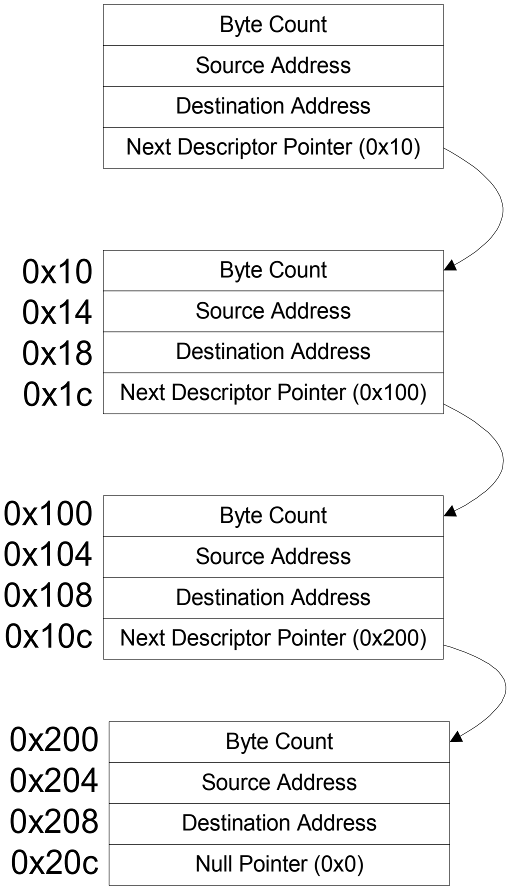
10.5.3 Chain Mode

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 348 on page 252.

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 239) 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 348 on page 252).

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–64242 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 15.1 “MPP Multiplexing” on page 275 section for more information.

NOTE: Program the number of TClk cycles that DMAAck* is asserted through the DMAAck_Width bit [4], see Figure 348 on page 252.

DMAAck* cannot be targeted to both the source and destination devices. See the Channel x Control register’s DMAAckDir bits [30:29] (

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 123). When setting the DMAAck-Mode bit to '0', DMAAck is asserted for one cycle, as soon as the IDMA engine issues the transaction to the target unit.

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–64242 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–64242 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–64242 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–64242 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 15.1 "MPP Multiplexing" on page 275). 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–64242 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.

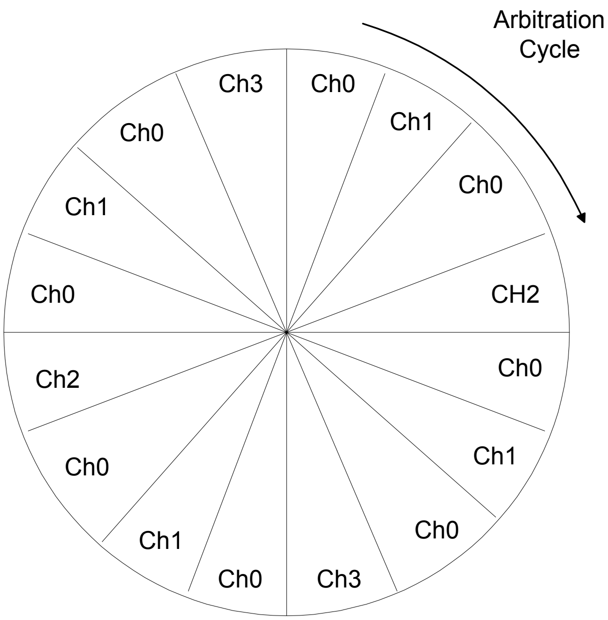
Each IDMA channel has a Current Descriptor Pointer register (CDPTR) associated with it. This register is used for closing the current descriptor before fetching the next descriptor. The register is a read/write register but the

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–64242 supports both Little and Big Endian convention.

The device endianess is determined by the CPU Configuration register’s Endianess bit, see Table 68 on page 71.

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–64242 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 58.

When the GT–64242 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 349 on page 255.

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 312: IDMA Descriptor Register Map

Register	Offset	Page
Channel 0 DMA Byte Count	0x800	page 247
Channel 1 DMA Byte Count	0x804	page 247
Channel 2 DMA Byte Count	0x808	page 247
Channel 3 DMA Byte Count	0x80c	page 247
Channel 0 DMA Source Address	0x810	page 248
Channel 1 DMA Source Address	0x814	page 248

Table 312: IDMA Descriptor Register Map (Continued)

Register	Offset	Page
Channel 2 DMA Source Address	0x818	page 248
Channel 3 DMA Source Address	0x81c	page 248
Channel 0 DMA Destination Address	0x820	page 248
Channel 1 DMA Destination Address	0x824	page 248
Channel 2 DMA Destination Address	0x828	page 248
Channel 3 DMA Destination Address	0x82c	page 249
Channel 0 Next Descriptor Pointer	0x830	page 249
Channel 1 Next Descriptor Pointer	0x834	page 249
Channel 2 Next Descriptor Pointer	0x838	page 249
Channel 3 Next Descriptor Pointer	0x83c	page 249
Channel 0 Current Descriptor Pointer	0x870	page 249
Channel 1 Current Descriptor Pointer	0x874	page 250
Channel 2 Current Descriptor Pointer	0x878	page 250
Channel 3 Current Descriptor Pointer	0x87c	page 250
Channel 0 Source High PCI Address	0x890	page 250
Channel 1 Source High PCI Address	0x894	page 250
Channel 2 Source High PCI Address	0x898	page 250
Channel 3 Source High PCI Address	0x89c	page 251
Channel 0 Destination High PCI Address	0x8a0	page 251
Channel 1 Destination High PCI Address	0x8a4	page 251
Channel 2 Destination High PCI Address	0x8a8	page 251
Channel 3 Destination High PCI Address	0x8ac	page 251
Channel 0 Next Descriptor High PCI Address	0x8b0	page 251
Channel 1 Next Descriptor High PCI Address	0x8b4	page 251
Channel 2 Next Descriptor High PCI Address	0x8b8	page 252
Channel 3 Next Descriptor High PCI Address	0x8bc	page 252

Table 313: IDMA Control Register Map

Register	Offset	Page
Channel 0 Control (Low)	0x840	page 252
Channel 0 Control (High)	0x880	page 255
Channel 1 Control (Low)	0x844	page 256
Channel 1 Control (High)	0x884	page 256
Channel 2 Control (Low)	0x848	page 257
Channel 2 Control (High)	0x888	page 257
Channel 3 Control (Low)	0x84c	page 257
Channel 3 Control (High)	0x88c	page 257
Channels 0-3 Arbiter Control	0x860	page 257
Channels 0-3 Crossbar Timeout	0x8d0	page 258

Table 314: IDMA Interrupt Register Map

Register	Offset	Page
Channels 0-3 Interrupt Cause	0x8c0	page 258
Channels 0-3 Interrupt Mask	0x8c4	page 259
Channels 0-3 Error Address	0x8c8	page 260
Channels 0-3 Error Select	0x8cc	page 260

Table 315: IDMA Debug Register Map

NOTE: Reserved for Galileo Technology usage.

Register	Offset	Page
X0 Address	0x8e0	page 261
X0 Command and ID	0x8e4	page 261
X0 Write Data (Low)	0x8e8	page 261
X0 Write Data (High)	0x8ec	page 261
X0 Write Byte Enables	0x8f8	page 261
X0 Read Data (Low)	0x8f0	page 262
X0 Read Data (High)	0x8f4	page 262
X0 Read ID	0x8fc	page 262

10.9.1 IDMA Descriptor Registers

Table 316: 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
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 317: Channel 1 DMA Byte Count, Offset: 0x804

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Byte Count.	0x0

Table 318: Channel 2 DMA Byte Count, Offset: 0x808

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Byte Count.	0x0

Table 319: Channel 3 DMA Byte Count, Offset: 0x80c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Byte Count.	0x0

Table 320: 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 321: 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 322: 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 323: 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 324: 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 325: 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 326: 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 327: 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 328: 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 329: 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 330: 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 331: 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 332: 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 333: 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 334: 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 335: 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 336: 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 337: 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 338: 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 339: 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 340: 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 341: 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 342: 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 343: 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 344: 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 345: 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 346: 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 347: 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

10.9.2 IDMA Channel Control Registers

Table 348: Channel 0 Control (Low), Offset: 0x840

Bits	Field name	Function	Initial Value
2:0	Reserved	Read only 0.	0x0
3	SrcHold	Source Hold 0 - Increment source address. 1 - Hold in the same value.	0x0
4	Reserved	Read only 0.	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 - 1 Byte 110 - 2 Bytes 010 - 4 Bytes 000 - 8 Bytes 001 - 16 Bytes 011 - 32 Bytes 111 - 64 Bytes 100 - 128 Bytes	0x0
9	ChainMode	Chained Mode 0 - Chained mode 1 - Non-Chained mode	0x0

Table 348: Channel 0 Control (Low), Offset: 0x840 (Continued)

Bits	Field name	Function	Initial Value
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
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

Table 348: Channel 0 Control (Low), Offset: 0x840 (Continued)

Bits	Field name	Function	Initial Value
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
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 memory space 10–11 - Reserved	0x0
24:23	DAddrOvr	Override Destination Address 00 - No override. 01 - Destination address is in PCI memory space 10–11 - Reserved	0x0
26:25	NAddrOvr	Override Next Descriptor Address 00 - No override 01 - Descriptor address is in PCI memory space 10–11 - 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

Table 348: Channel 0 Control (Low), Offset: 0x840 (Continued)

Bits	Field name	Function	Initial Value
31	DescMode	Descriptor Mode 0 - Compatibility mode 1 - New descriptor structure	0x0

Table 349: 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 50 on page 66.	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 50 on page 66.	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 50 on page 66.	0x1
14	Reserved	Reserved	0x0

Table 349: Channel 0 Control (High), Offset: 0x880 (Continued)

NOTE: Program the High Control register prior to channel activation.

Bits	Field Name	Function	Initial Value
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 50 on page 66.	0x0
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 50 on page 66.	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 50 on page 66.	0x0
31:24	Reserved	Must be 0.	0x0

Table 350: Channel 1 Control (Low), Offset: 0x844

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (Low).	0x0

Table 351: Channel 1 Control (High), Offset: 0x884

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (High).	0x101010

Table 352: Channel 2 Control (Low), Offset: 0x848

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (Low).	0x0

Table 353: Channel 2 Control (High), Offset: 0x888

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (High).	0x101010

Table 354: Channel 3 Control (Low), Offset: 0x84c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (Low).	0x0

Table 355: Channel 3 Control (High), Offset: 0x88c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as Channel 0 Control (High).	0x101010

Table 356: 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
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

Table 356: Channels 0-3 Arbiter Control, Offset: 0x860 (Continued)

Bits	Field name	Function	Initial Value
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 357: 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

10.9.3 IDMA Interrupt Registers

Table 358: 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
5	EOT	Channel0 End of Transfer	0x0
7:6	Reserved	Reserved.	0x0

Table 358: Channels 0-3 Interrupt Cause, Offset: 0x8c0 (Continued)

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
13:8	Various	Same as channel0 cause bits.	0x0
15:14	Reserved	Reserved	0x0
21:16	Various	Same as channel0 cause bits.	0x0
23:22	Reserved	Reserved.	0x0
29:24	Various	Same as channel0 cause bits.	0x0
31:30	Reserved	Reserved.	0x0

Table 359: 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
29:24	Various	Same as channel0 mask bits.	0x0
31:30	Reserved	Reserved.	0x0

Table 360: 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 361: 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 - EOT Channel 0 0x6 - 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 0x15 - EOT Channel 2 0x15 - 0x17 - Reserved 0x18 - Comp Channel 3 0x19 - AddrMiss Channel 3 0x1a - AccProt Channel 3 0x1b - WrProt Channel 3 0x1c - Own Channel 3 0x1d - EOT Channel 3 0x1e - 0x1f - Reserved Read Only.	0x0

Table 361: Channels 0-3 Error Select, Offset: 0x8cc (Continued)

Bits	Field Name	Function	Initial Value
31:5	Reserved	Reserved.	0x0

10.9.4 IDMA Debug Registers

NOTE: Reserved for Galileo Technology usage.

Table 362: 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 363: 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 364: 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 365: 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 366: 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 367: 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 368: 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 369: 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

11. TIMER/COUNTERS

There are four 32-bit wide timer/counters on the GT–64242. 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.

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.

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 370: IDMA Descriptor Register Map

Register	Offset	Page
Timer/Counter 0	0x850	page 264
Timer/Counter 1	0x854	page 264
Timer/Counter 2	0x858	page 264
Timer/Counter 3	0x85c	page 264
Timer/Counter 0-3 Control	0x864	page 264
Timer/Counter 0-3 Interrupt Cause	0x868	page 266
Timer/Counter 0-3 Interrupt Mask	0x86c	page 266

Table 371: Timer/Counter 0, Offset: 0x850

Bits	Field Name	Function	Initial Value
31:0	TC0	Timer/Counter 0 Value	0x0

Table 372: Timer/Counter 1, Offset: 0x854

Bits	Field Name	Function	Initial Value
31:0	TC1	Timer/Counter 1 value.	0x0

Table 373: Timer/Counter 2, Offset: 0x858

Bits	Field Name	Function	Initial Value
31:0	TC2	Timer/Counter 2 value.	0x0

Table 374: Timer/Counter 3, Offset: 0x85c

Bits	Field Name	Function	Initial Value
31:0	TC3	Timer/Counter 3 value.	0x0

Table 375: 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’.	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
7:3	Reserved	Reserved.	0x0

Table 375: 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’.	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
15:11	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’.	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
23:192	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’.	0x0

Table 375: Timer/Counter 0-3 Control, Offset: 0x864 (Continued)

Bits	Field name	Function	Initial Value
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
31:27	Reserved	Reserved	0x0

Table 376: 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

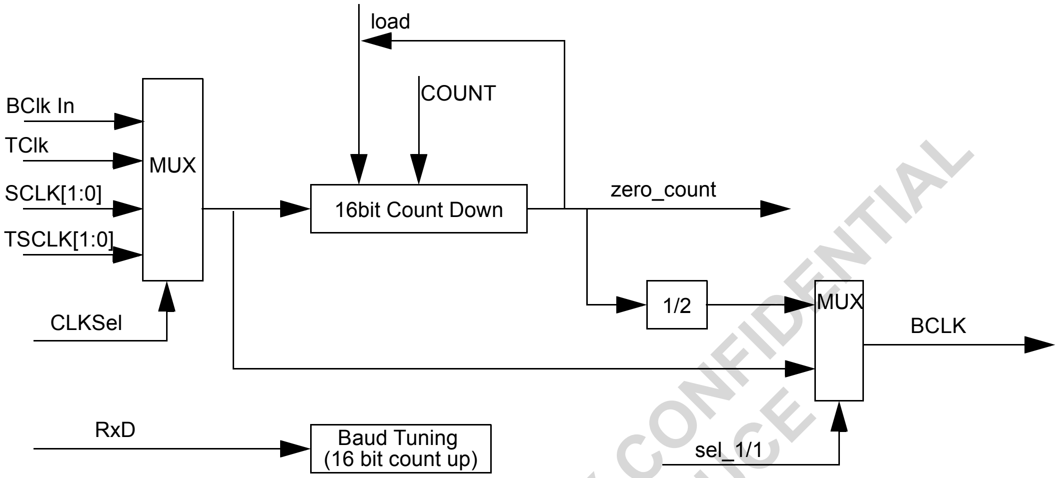
Table 377: Timer/Counter 0-3 Interrupt Mask, Offset: 0x86c

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

12. BAUDE RATE GENERATORS (BRG)

There are three baud rate generators (BRGs) in the GT–64242. Figure 39 shows a BRG block diagram.

Figure 39: Baud Rate Generator Block Diagram



12.1 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.

12.2 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.

12.3 BRG Registers

Table 378: BRG Registers Map

Register Name	Offset	Page
BRG0		
BRG0 Configuration Register (BCR0)	0xb200	page 268
BRG0 Baud Tuning Register (BTR0)	0xb204	page 269
BRG1		
BRG1 Configuration Register (BCR1)	0xb208	page 268
BRG1 Baud Tuning Register (BTR1)	0xb20c	page 269
BRG2		
BRG2 Configuration Register (BCR2)	0xb210	page 268
BRG2 Baud Tuning Register (BTR2)	0xb214	page 269
BRG Interrupts		
BRG Cause Register	0xb834	page 269
BRG Mask Register	0xb8b4	

Table 379: 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: $\mathrm{BaudRate} = \frac{\mathrm{InputClockRate}}{(\mathrm{CDV}+1) \times 2}$ 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 379: 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 380: 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 379 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 381: 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
2	BTR2	Baud Tuning 2 interrupt	0x0
31:3		Reserved.	0x0

NOTE: When a mask bit is set to ‘1’, the corresponding cause bit is also enabled.

13. WATCHDOG TIMER

The GT–64242 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 15. "MPP Multiplexing" on page 275). The watchdog timer can be activated only after configuring two MPP pins to act as WDE and WDNMI.

13.1 Watchdog Registers

Table 382: 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 383: 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

13.2 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–64242 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–64242 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–64242 must be reset. The GT–64242 holds WDE* asserted for the duration of 16 system cycles after reset assertion.

14. General Purpose Port

GT–64242 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–64242 MPP pins (see Section 15.1 "MPP Multiplexing" on page 275 section for more information).

14.1 GPP Control Registers

The GT–64242 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–64242 negates an active low input pin before latching it inside. It inverts an active low output pin before driving it outside.

14.2 GPP Value Register

The GT–64242 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).

14.3 GPP Interrupts

The GPP input pins can be used to register external interrupts. The GT–64242 supports only edge sensitive 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').

NOTE: If the external device is generating a level interrupt, the interrupt handler must clear the interrupt cause bit, and then keep polling on the GPP Value register, to make sure there is no more pending interrupt.

14.4 General Purpose Port Registers

Table 384: GPP Register Map

Register	Offset	Page
GPP I/O Control	0xf100	page 273
GPP Level Control	0xf110	page 273
GPP Value	0xf104	page 273
GPP Interrupt Cause	0xf108	page 274
GPP Interrupt Mask	0xf10c	page 274

Table 385: 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 386: 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 387: 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 388: 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 389: 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

15. MPP MULTIPLEXING

The GT–64242 has 32 MPP pins.

15.1 MPP Multiplexing

The GT–64242 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–64242 different interfaces (UMA control, DMA control, PCI arbiter signals), or as General Purpose Ports.

Table 390 shows each MPP pins' functionality as determined by the MPP Multiplex register.

Table 390: 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*	PME*	Reserved	Reserved	PME*	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*	Reserved	PME*	PME*	Reserved	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]
MPP[16]	**MPP[17]**	**MPP[18]**	**MPP[19]**	**MPP[20]**	**MPP[21]**	**MPP[22]**	**MPP[23]**

Table 390: MPP Function Summary (Continued)

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*	PME*	Reserved	Reserved	PME*	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*	Reserved	PME*	PME*	Reserved	MGNT*	MREQ*
MGNT*	MREQ*	Reserved	PME*	PME*	Reserved	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.

15.2 MPP Interface Registers

Table 391: GPP Interface Register Map

Register	Offset	Page
MPP Control0	0xf000	page 277
MPP Control1	0xf004	page 280

Table 391: GPP Interface Register Map (Continued)

Register	Offset	Page
MPP Control2	0xf008	page 283
MPP Control3	0xf00c	page 286

Table 392: 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 392: MPP Control0, Offset: 0xf000 (Continued)

Bits	Field Name	Function	Initial Value
11:8	MPPSel2	MPP2 Select 0x0 - GPP[2] 0x1 - DMAReq[1]* 0x2 - PME* 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 - Reserved 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 - Reserved 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 392: MPP Control0, Offset: 0xf000 (Continued)

Bits	Field Name	Function	Initial Value
23:20	MPPSel5	MPP5 Select 0x0 - GPP[5] 0x1 - DMAAck[2]* 0x2 - PME* 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 393: 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 - Reserved 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 393: MPP Control1, Offset: 0xf004 (Continued)

Bits	Field Name	Function	Initial Value
15:12	MPPSel11	MPP11 Select 0x0 - GPP[11] 0x1 - DMAAck[5]* 0x2 - PME* 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 - PME* 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 - Reserved 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 393: 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 394: 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 - PME* 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 394: MPP Control2, Offset: 0xf008 (Continued)

Bits	Field Name	Function	Initial Value
15:12	MPPSel19	MPP19 Select 0x0 - GPP[19] 0x1 - DMAAck[1]* 0x2 - Reserved 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 - Reserved 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 - PME* 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 394: 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 395: 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 - Reserved 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 395: MPP Control3, Offset: 0xf00c (Continued)

Bits	Field Name	Function	Initial Value
15:12	MPPSel27	MPP27 Select 0x0 - GPP[27] 0x1 - DMAAck[5]* 0x2 - PME* 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 - PME* 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 - Reserved 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 395: 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

16. I^{2}C INTERFACE

The GT–64242 has full I^{2}C support. It can act as master generating read/write requests and as a slave responding to read/write requests. It fully supports multiple I^{2}C masters environment (clock synchronization, bus arbitration).

The I^{2}C interface can be used for various applications. It can be used to control other I^{2}C on board devices, to read DIMM SPD ROM and is also used for serial ROM initialization. For more details, see Section 20. “Reset Configuration” on page 311.

16.1 I^{2}C Bus Operation

The I^{2}C port consists of two open drain signals:

- SCL (Serial Clock)
- SDA (Serial address/data)

The I^{2}C 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 I^{2}C 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 I^{2}C master ends the transaction with stop condition.

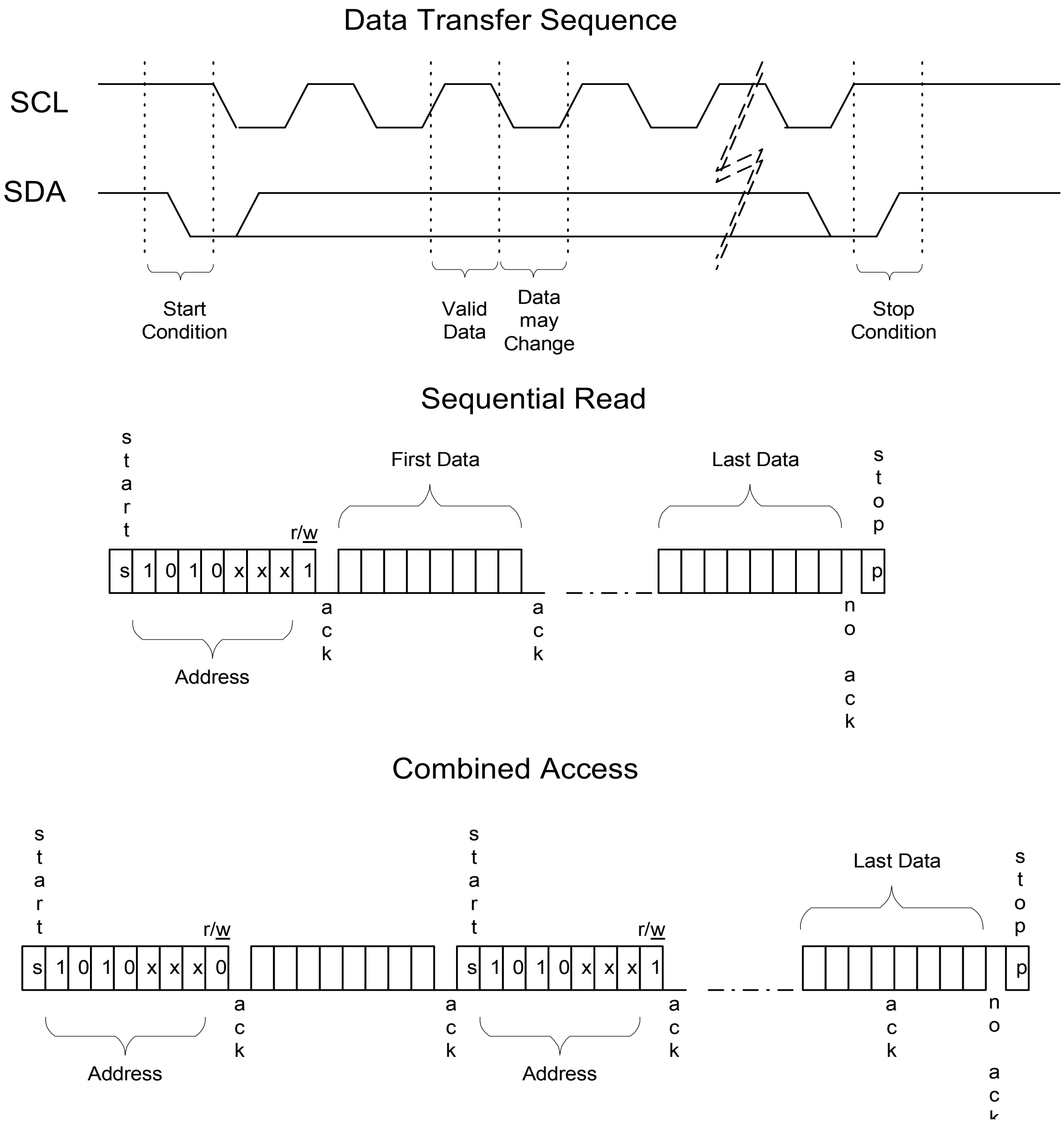
In case of read access, following the slave address acknowledge, the I^{2}C slave drives 8-bit data and the master responds with acknowledge. This read access (8-bit data followed by acknowledge) continues until the I^{2}C master ends the transaction by responding with no acknowledge to the last 8-bit 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 I^{2}C master.

I^{2}C examples are shown in Figure 40. For full I^{2}C protocol description look in Philips Semiconductor I^{2}C spec.

Figure 40: I^{2}C Examples



16.2 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.

16.2.1 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 399 and Table 400 on page 297).

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).

16.2.2 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.

16.2.3 I²C Control Register

This 8-bit register contains the following bits:

Table 396: 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 396: I^{2}C Control Register Bits (Continued)

Bit	Function	Description
3	Interrupt Flag	If any of the interrupt events occur, set to ‘1’ by I^{2}C hardware, see Section 16.2.4 “I2C Status Register” on page 292. If set to ‘1’ and I^{2}C interrupts are enabled through bit[7], an interrupt is asserted.
4	Stop Bit	When set to ‘1’, the I^{2}C master initiates a stop condition on the bus. The bit is set only. It is cleared by I^{2}C hardware after a stop condition is driven on the bus.
5	Start Bit	When set to ‘1’, the I^{2}C 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 I^{2}C hardware after a start condition is driven on the bus.
6	I^{2}C Enable	If set to ‘1’, the I^{2}C 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 I^{2}C slave does not respond to any address on the bus.
7	Interrupt Enable	If set to ‘1’, I^{2}C interrupts are enabled. **NOTE:** It is highly recommended to use I^{2}C interrupt to interface the I^{2}C module, rather than using register polling method.

16.2.4 I^{2}C Status Register

This 8-bit register contains the current status of the I^{2}C interface. Bits[7:3] are the status code, bits[2:0] are Reserved (read only 0). Table 397 summarizes all possible status codes.

Table 397: I^{2}C 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 397: I^{2}C 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.

16.2.5 Baude Rate Register

I^2C spec defines SCL frequency of 100KHz (400KHz in fast mode). The I^2C 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:

$$F_{SCL} = \frac{F_{TClk}}{10 \cdot (M+1) \cdot 2^{(N+1)}}$$

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.

16.3 I^2C Master Operation

The CPU can initiate I^2C master read and write transactions via I^2C registers, as described in the following sections.

16.3.1 Master Write Access

Master write access consists of the following steps:

1. The CPU sets the I^2C Control register's Start bit [5] to '1', see Table 402 on page 297. The I^2C 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 I^2C 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 I^2C 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 I^2C 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 I^2C master to generate a stop condition on the bus, and go back to idle state. To restart a new transaction, the CPU sets the I^2C Control register's Start bit and clears the Interrupt flag, causing I^2C 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.

16.3.2 Master Read Access

1. Generating start condition, exactly the same as in the case of write access, see Section 16.3.1 “Master Write Access” on page 294).
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^{2}C Control register’s Acknowledge bit (when clearing the Interrupt bit), causing the I^{2}C 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.

16.4 I^{2}C Slave Operation

The I^{2}C 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.

16.4.1 Slave Read 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 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^{2}C 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^{2}C 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^{2}C slave interface returns back to idle state.

If the master generates a stop condition after driving an acknowledge bit, the I^{2}C slave interface returns back to idle state.

16.4.2 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.

16.5 I^{2}C Interface Registers

Table 398: I^{2}C Interface Register Map

Register	Offset	Page
I^{2}C Slave Address	0xc000	page 296
I^{2}C Extended Slave address	0xc010	page 297
I^{2}C Data	0xc004	page 297
I^{2}C Control	0xc008	page 297
I^{2}C Status/Baude Rate	0xc00c	page 298
I^{2}C Soft Reset	0xc01c	page 298

Table 399: 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 400: I^2C 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 401: I^2C Data, Offset: 0xc004

Bits	Field Name	Function	Initial Value
7:0	Data	Data/Address byte to be transmitted by the I^2C master or slave, or data byte received.	0x0
31:8	Reserved	Reserved.	0x0

Table 402: I^2C Control, Offset: 0xc008

Bits	Field Name	Function	Initial Value
1:0	Reserved	Read only.	0x0
2	ACK	Acknowledge When set to ‘1’, the I^2C master drives the acknowledge bit in response to received read data and to the I^2C 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 I^2C 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–64242 drives a stop condition on the bus. It is cleared by the I^2C hardware.	0x0
5	Start	Start When set to ‘1’, the GT–64242 drives a start condition as soon as the bus is free. It is cleared by the I^2C hardware.	0x0
6	I^2CEn	If set to ‘0’, the SDA and SCL inputs are not sampled and the I^2C 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 403: 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 404: 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 405: 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

17. INTERRUPT CONTROLLER

The GT–64242 includes an interrupt controller that routes internal interrupt requests (and optionally external interrupt requests) to both the CPU and the PCI bus.

The GT–64242 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–64242 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.

17.1 Interrupt Cause and Mask Registers

The GT–64242 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.

17.1.1 Interrupts Cause Registers

The GT–64242 units cause registers are:

Table 406: Interrupts Cause Registers

• CPU Cause register	• PCI Inbound Cause register	• BRG Cause register
• SDRAM Error Address register	• PCI Outbound Cause register	• GPP Cause register
• Device Interface Cause register	• IDMAs 0-3 Cause register	• I^{2}C Cause register
• PCI Cause register	• Timers 0-3 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.

17.1.2 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 two interrupts - PCI 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.

17.1.3 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–64242 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.

17.1.4 Error Report Registers

The GT–64242 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.

17.2 Interrupt Controller Registers

Table 407: Interrupt Controller Register Map

Register	Offset	Page
Main Interrupt Cause (Low)	0xc18	page 301
Main Interrupt Cause (High)	0xc68	page 302
CPU Interrupt Mask (Low)	0xc1c	page 303
CPU Interrupt Mask (High)	0xc6c	page 305

Table 407: Interrupt Controller Register Map (Continued)

Register	Offset	Page
CPU Select Cause	0xc70	page 304
PCI Interrupt Mask (Low)	0xc24	page 305
PCI Interrupt Mask (High)	0xc64	page 305
PCI Select Cause	0xc74	page 305
CPU Int[0]* Mask	0xe60	page 305
CPU Int[1]* Mask	0xe64	page 305
CPU Int[2]* Mask	0xe68	page 305
CPU Int[3]* Mask	0xe6c	page 305

Table 408: 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
7:6	Reserved	Reserved.	0x0
8	Timer0_1	Timers 0-1 Interrupt	0x0
9	Timer2_3	Timers 2-3 Interrupt	0x0
11:10	Reserved	Reserved.	0x0
12	PCI	PCI Interrupt **NOTE:** Summary of the PCI Cause register's bits[7:0].	0x0
13	PCI	PCI Interrupt **NOTE:** Summary of the PCI Cause register's bits[15:8].	0x0
14	PCI	PCI Interrupt **NOTE:** Summary of the PCI Cause register's bits[23:16].	0x0
15	PCI	PCI Interrupt **NOTE:** Summary of the PCI Cause register's bits[31:24].	0x0

Table 408: Main Interrupt Cause (Low), Offset: 0xc18[1] (Continued)

Bits	Field Name	Function	Initial Value
16	Reserved	**NOTE:** Reserved.	0x0
17	ECC	ECC Error Interrupt	0x0
20:18	Reserved	Reserved.	0x0
21	PCIOutL	PCI Outbound Interrupt Summary **NOTE:** Summary of the PCI Outbound Cause register's bits[15:0].	0x0
22	PCIOutH	PCI Outbound Interrupt Summary **NOTE:** Summary of the PCI Outbound Cause register's bits[31:16].	0x0
25:23	Reserved	Reserved.	0x0
26	PCIInL	PCI Inbound Interrupt **NOTE:** Summary of the PCI Inbound Cause register's bits[15:0].	0x0
27	PCIInH	PCI_0 Inbound Interrupt **NOTE:** Summary of the PCI Inbound Cause register's bits[31:16].	0x0
31:28	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 409: Main Interrupt Cause (High), Offset: 0xc68

Bits	Field Name	Function	Initial Value
4:0	Reserved	Reserved.	0x0
5	I^2C	I^2C Interrupt	0x0
6	Reserved	Reserved.	0x0
7	BRG	Baude Rate Generator Interrupt	0x0
10:7	Reserved	Reserved.	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

Table 409: Main Interrupt Cause (High), Offset: 0xc68 (Continued)

Bits	Field Name	Function	Initial Value
26	GPP23_16	GPP[23:16] Interrupt	0x0
27	GPP31_24	GPP[31:24] Interrupt	0x0
31:28	Reserved	Reserved.	0x0

Table 410: 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
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
12	PCI0	If set to ‘1’, PCI_0 interrupt is enabled.	0x0
13	PCI1	If set to ‘1’, PCI1 interrupt is enabled.	0x0
14	PCI2	If set to ‘1’, PCI2 interrupt is enabled.	0x0
15	PCI3	If set to ‘1’, PCI3 interrupt is enabled.	0x0
16	Reserved	Reserved.	0x0
17	ECC	If set to ‘1’, ECC interrupt is enabled.	0x0
20:18	Reserved	Reserved.	0x0
21	PCIOutL	If set to ‘1’, PCIOutL interrupt is enabled.	0x0
22	PCIOutH	If set to ‘1’, PCIOutH interrupt is enabled.	0x0
25:23	Reserved	Reserved.	0x0
26	PCIInL	If set to ‘1’, PCIInL interrupt is enabled.	0x0
27	PCIInH	If set to ‘1’, PCIInH interrupt is enabled.	0x0
31:28	Reserved	Reserved.	0x0

Table 411: CPU Interrupt Mask (High), Offset: 0xc6c

Bits	Field Name	Function	Initial Value
4:0	Reserved	Reserved.	0x0
5	I^{2}C	If set to '1', I^{2}C interrupt is enabled.	0x0
6			0x0
7	BRG	If set to '1', BRG interrupt is enabled.	0x0
10:7	Reserved	Reserved.	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
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 412: 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 413: PCI Interrupt Mask (Low), Offset: 0xc24

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Interrupt Mask (Low).	0x0

Table 414: PCI Interrupt Mask (High), Offset: 0xc64

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Interrupt Mask (High).	0x0

Table 415: PCI Select Cause, Offset: 0xc74

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Select Cause.	0x0

Table 416: 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 417: CPU Int[1]* Mask, Offset: 0xe64

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Int[0]* Mask.	0x0

Table 418: CPU Int[2]* Mask, Offset: 0xe68

Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Int[1]* Mask.	0x0

Table 419: CPU Int[3]* Mask, Offset: 0xe6c

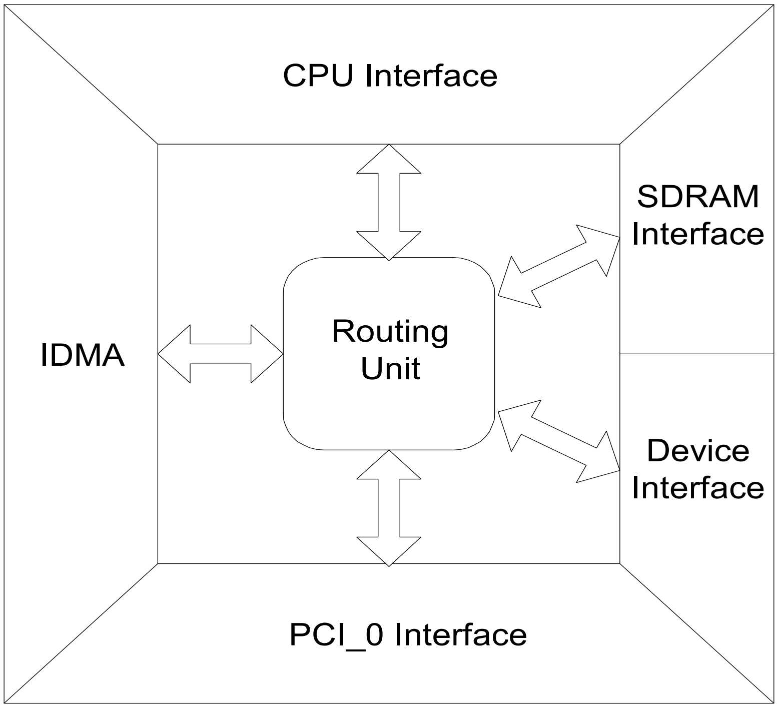
Bits	Field Name	Function	Initial Value
31:0	Various	Same as CPU Int[2]* Mask.	0x0

18. INTERNAL ARBITRATION CONTROL

The GT–64242 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, and PCI read from device at the same time), as well as transaction pipelining (issue multiple transactions in parallel between the same source and destination).

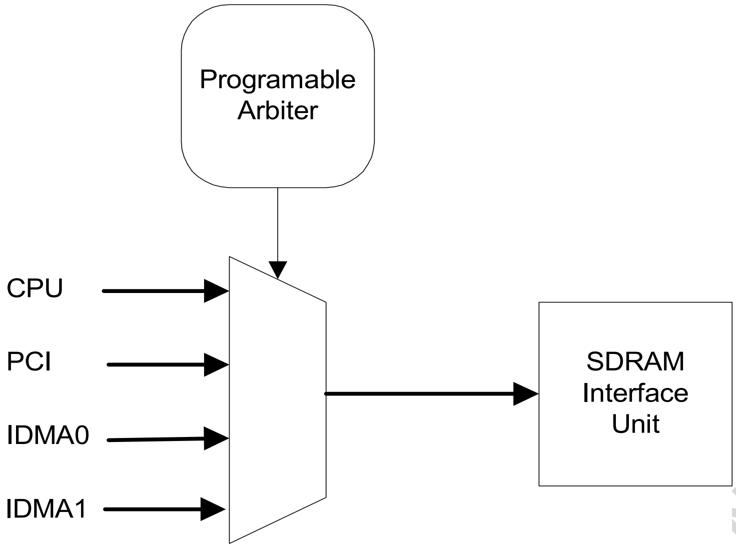
Figure 41 shows how the data path routing is controlled via a central routing unit (also called Crossbar).

Figure 41: GT–64242 Inter Units Connect



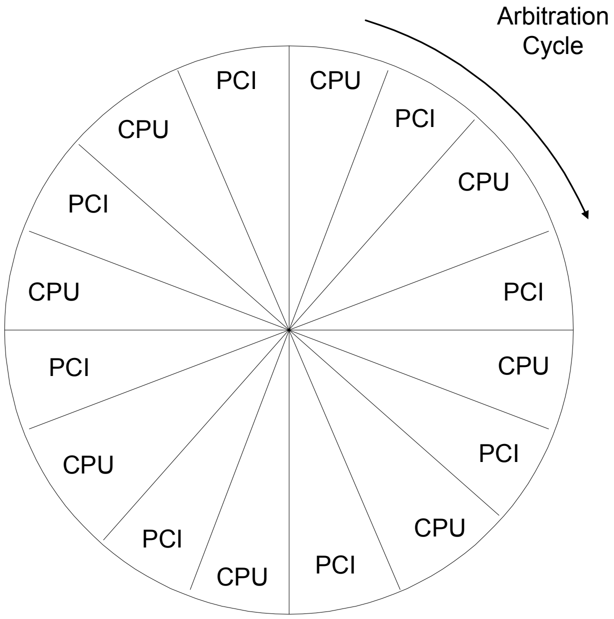
Sometimes conflicts may occur over resources. For example, if the CPU, PCI, and IDMA request access to SDRAM simultaneously, these requests cannot be served at the same time. The central routing unit contains programmable arbitration mechanisms to optimize device performance, according to the system requirements, as shown in Figure 42.

Figure 42: SDRAM Interface Arbitration



Each arbiter is a user defined round-robin arbiter (called a “pizza arbiter”). Figure 43 shows an example of the Device interface arbiter.

Figure 43: 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.
- 50%for PCItransactions.

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.

19. RESET PINS

The GT–64242 supports two reset pins:

- SysRst* which is the main reset pin.
- RST* which is the PCI interface reset pin.

Separating SysRst* from the PCI reset pin 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.

SysRst* is the main GT–64242 reset pin. When asserted, all GT–64242 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 98).

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–64242 output pins to remain floated for three cycles after reset deassertion.

The PCI reset pin is independent. The PCI interface is kept in its reset state as long as its 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 pin must never be deasserted prior to SysRst* deassertion.

Since the GT–64242 supports SysRst* deassertion prior to the PCI reset pin 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] for this purpose, see Table 194 on page 179. Upon PCI reset deassertion, the bit is set to ‘1’.

20. Reset Configuration

The GT–64242 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–64242 to function as required.

The GT–64242 supports two methods of reset configuration:

- Pins sampled on SysRst* deassertion (requires pins pulled up/down to Vcc/GND).
- Serial ROM initialization.

20.1 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 420: 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 20.2 “Serial ROM Initialization” on page 314.
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 420: Reset Configuration (Continued)

Pin	Configuration Function
AD[8]	Reserved
	NOTE: Must pull down.
AD[9]	Multiple GT–64242 Support
0- 1-	Not supported Supported
AD[11:10]	Multi-GT–64242 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 Expansion ROM Support
0- 1-	Not supported Supported
AD[18]	Reserved

Table 420: Reset Configuration (Continued)

Pin	Configuration Function
AD[19]	PCI Power Management Support
0- 1-	Not supported Supported
AD[20]	Reserved
	Reserved
AD[21]	PCI VPD Support
0- 1-	Not supported Supported
AD[22]	Reserved
	Reserved
AD[23]	PCI MSI Support
0- 1-	Not supported Supported
AD[24]	Reserved
	Reserved
AD[25]	CompactPCI Hot Swap
0- 1-	Not supported Supported
AD[26]	PCI BIST Support
0- 1-	Not supported Supported
AD[27]	Reserved
	Reserved
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.

Table 420: Reset Configuration (Continued)

Pin	Configuration Function
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–64242 samples PCI 64EN pin PCI 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–64242 PCI 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.

20.2 Serial ROM Initialization

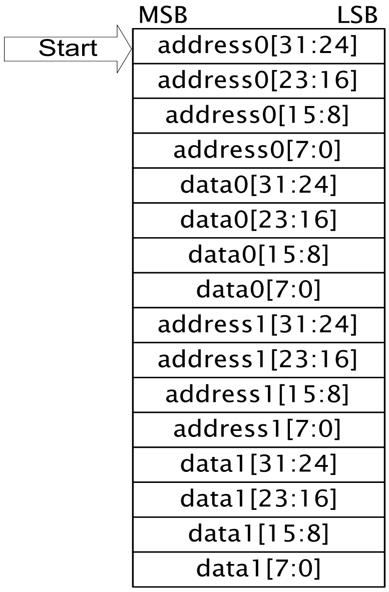
The GT–64242 supports initialization of ALL it’s internal and configuration registers and other system components through the I^{2}C master interface. If serial ROM initialization is enabled (AD[0] pin sampled High on SysRst* deassertion), the GT–64242 I^{2}C master starts reading initialization data from serial ROM and writes it to the appropriate registers (or to any of GT–64242 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.

20.2.1 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 44.

Figure 44: Serial ROM Data Structure



The GT–64242 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–64242 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–64242 internal registers by setting addresses that match the CPU internal space (default address is 0x14000XXX).
- Program the GT–64242 PCI configuration registers using the PCI 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–64242 reaches last data, it stops the initialization sequence.

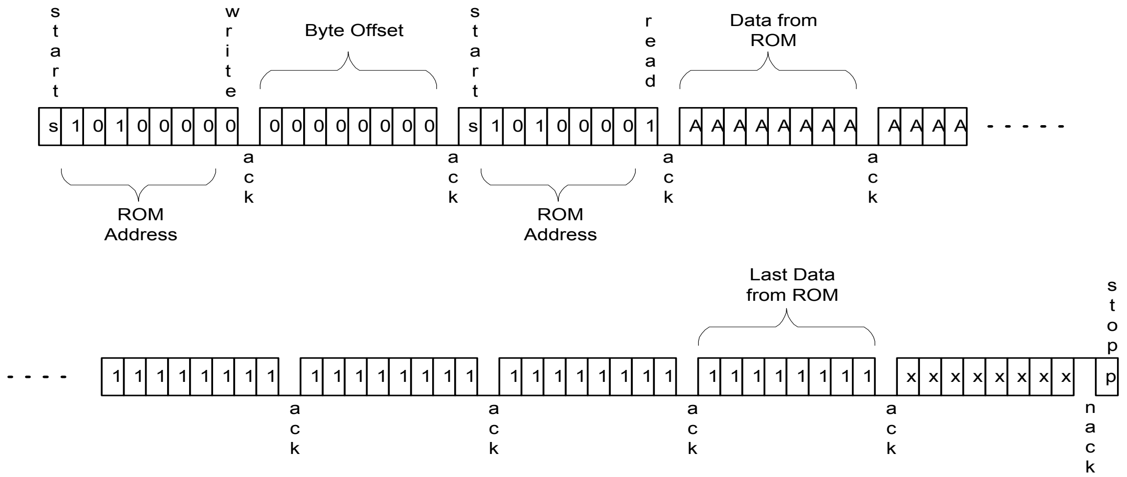
NOTE: The 32-bit address must always be in Little Endian convention.

When using the ROM for initializing the GT–64242’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.

20.2.2 Serial ROM Initialization Operation

On SysRst* deassertion, the GT–64242 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 45.

Figure 45: Serial ROM Read Example



For a detailed description of I^{2}C implementation, see Section 16. “I2C Interface” on page 289.

NOTE: Initialization data must be programmed in the serial ROM starting at offset 0x0

The GT–64242 assumes 7-bit serial ROM address of ‘b10100XX. The value of XX is sampled at reset (see section 20.2.3).

To set the ROM byte offset to ‘0’, the GT–64242 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–64242 receives an additional byte of dummy data. It responds with no-ack and then asserts the stop bit.

20.2.3 Serial ROM Initialization in Multi-GT Configuration

In multi-GT configuration, each GT–64242 device must have its own serial ROM initialization code.

The Serial ROM address bits[1:0] are sampled at reset. Each GT–64242 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–64242 device has its own ROM on the I^{2}C bus. Other serial ROMs use these bits as an internal page select. In this case, one serial ROM is shared between all GT–64242 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.

20.2.4 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.

20.2.5 Other Interfaces During Initialization

During initialization, any PCI attempt to access the GT–64242 results in retry termination. This allows the initialization sequence to program all PCI related registers, prior to an OS access to the GT–64242.

Also, the DRAM initialization sequence is postponed until serial initialization completes, see Section 5.10 “SDRAM Initialization” on page 98. 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 15.1 “MPP Multiplexing” on page 275).

20.2.6 Serial ROM Initialization Registers

Table 421: 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 422: Serial Init Last Data, Offset: 0xf324

Bits	Field Name	Function	Initial Value
31:0	DLast	Last Serial Data The GT–64242 finishes with serial ROM initialization when it reaches data that equals this register.	0xffffffff

Table 423: 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 424: 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^{2}C Status register bits[7:3]. Read only.	0x1f
31:5	Reserved	Reserved.	0x0

21. GT–64242 Clocking

The GT–64242 supports up to three clock domains:

- TClk (core and DRAM clock)
- SysClk (CPU bus clock)
- PClk

NOTE: In addition, each serial port has a different clock.

TClk is the fastest clock domain. It can run up to 100MHz and drives an internal PLL, that generates the GT–64242 core clock.

TClk is also used as the DRAM interface clock. The same clock source must drive the GT–64242 TClk input and the SDRAM clock (up to 0.5ns clocks skew). The GT–64242 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 101.

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.

22. 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.

22.1 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 420 on page 311.

Table 425: 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 426: 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 427: Pin Capacitance

Symbol	Parameter	Min.	Typ.	Max.	Unit
C	Pin Capacitance	5.2	8.7	9	pF

22.2 DC Electrical Characteristics Over Operating Range

Table 428: 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: MPP[31:0],	IoH = 8 mA	3.3V	2.4		V

Table 428: 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: MPP[31:0],	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:						

Table 428: 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 = 100MHz TClk/66 Mhz PClk			500	mA
		Core VCC1.8 = 1.9 V f = 100MHz TClk/66 Mhz PClk			950	mA
	ACK64*, CBE*[7:0], CLK, DEVSEL*, ENUM*, FRAME*, GNT*, HS, IDSEL, INT, IRDY*, LED, P64EN*, PAD[63:0], PAR, PAR64, PERR*, REQ*, REQ64*, Rst*, SERR*, STOP*, TRDY*, VREF	See PCI Specification Rev. 2.2				

22.3 Thermal Data

Table 429 shows the package thermal data for the GT–64242.

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 429: Thermal Data for The GT–64242 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		

22.4 PLL Power Filter Circuit

The GT–64242 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.

22.4.1 PLL Power Supply

The GT–64242 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–64242 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.

22.4.2 PLL Power Filter With a 1.8V Power Supply Available On Board

Figure 46 shows a recommended circuit for the GT–64242 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–64242.

Figure 46: PLL Power Filter Circuit With Common On-board 1.8V Supply

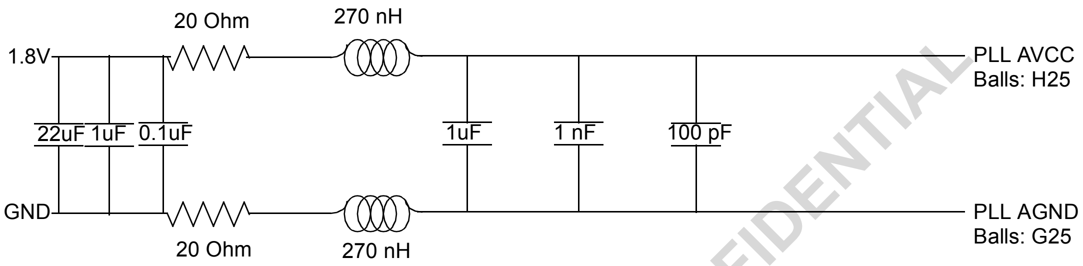
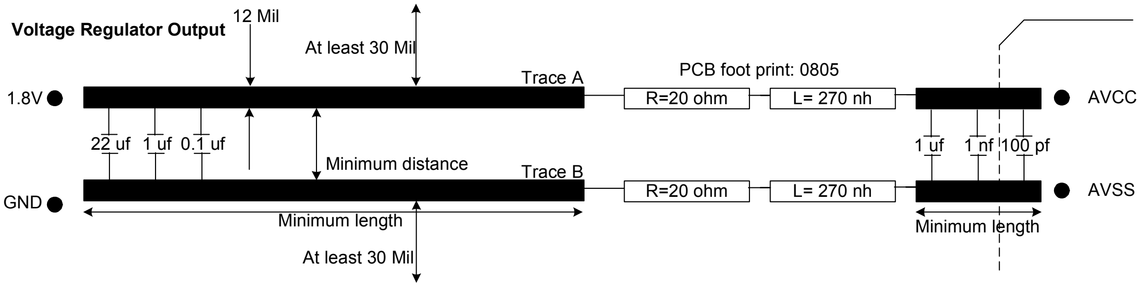


Figure 47: PLL Layout Guideline for a PCI Add-on Card



NOTE: In Figure 47, 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.

22.4.3 PLL Power Filter With No 1.8V Power Supply Available On-board (Backplane Layout)

Figure 48 shows a recommended circuit for the GT–64242 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–64242.

Figure 48: PLL Power Filter Circuit With Dedicated 1.8V Supply

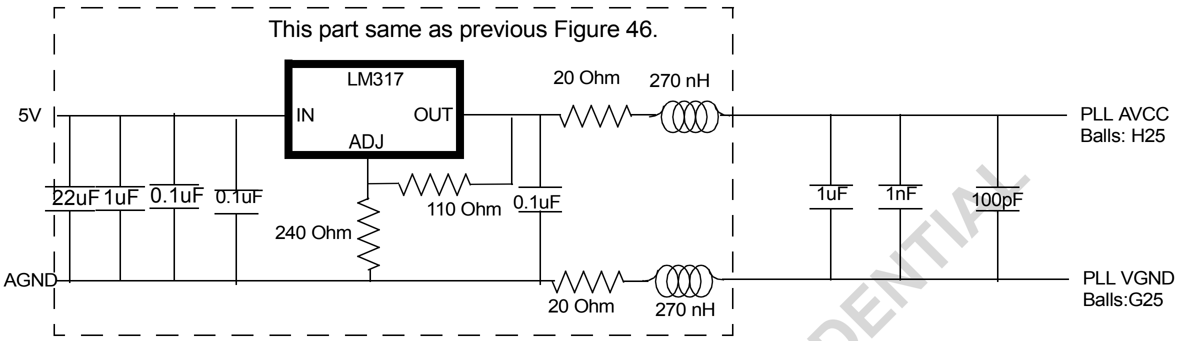
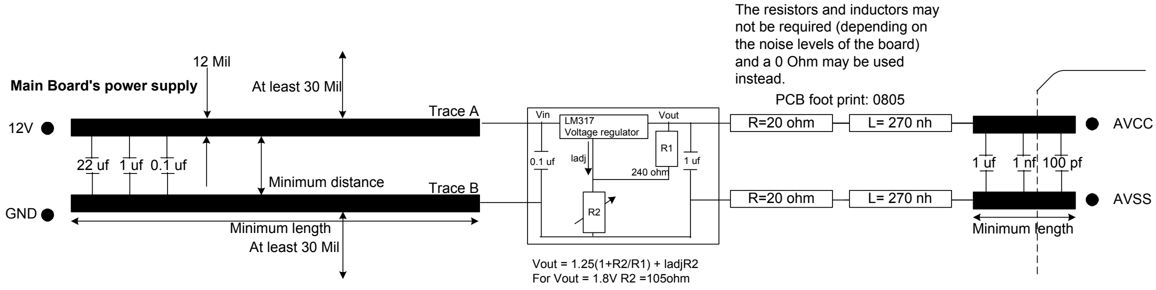


Figure 49: PLL Layout Guideline for Backplane Layout



The resistors and inductors may not be required (depending on the noise levels of the board) and a 0 Ohm may be used instead.

Vout = 1.25(1+R2/R1) + IadjR2
For Vout = 1.8V R2 =105ohm

22.4.4 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.

23. AC TIMING

NOTE: The following targets are subject to change.

Table 430: 100 MHz AC Timing

		100MHz			
Signals	**Description**	**Min.**	**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.					
PClk	Frequency		66	MHz	
PClk	Clock Period	15	∞	ns	

Table 430: 100 MHz AC Timing (Continued)

Signals	Description	100MHz		Units	Loading
		Min.	Max.		
Rst*	Active	1		ms	
FRAME*, IRDY*, TRDY*, STOP*, IDSEL, DEVSEL* REQ64*, ACK64*, PAR64, PERR*, PAD[63:0], CBE[7:0]*, PAR	Setup	3		ns	
GNT*	Setup	5		ns	
FRAME*, IRDY*, PAD[63:0], TRDY*, STOP*, IDSEL, PAR64, DEVSEL* GNT*, REQ64*, ACK64*,PAR, PERR*, CBE[7:0]*	Hold	0		ns	
FRAME*, TRDY*, IRDY* DEVSEL*, PAD[63:0], STOP*, CBE[7:0]*, REQ64*, ACK64*, REQ*, PAR PERR*, SERR*, PAR64 **NOTE:** Output delays are measured as specified in PCI spec rev. 2.2 section 7.6.4.3	Output Delay	2	6	ns	15pF
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	2		ns	
SData[63:0], ECC[7:0]	Hold	1		ns	
BankSel[1:0], DAdr[12:0], SRAS*, SCAS*, SCS[3:0]*, SDQM*[7:0], DWr*	Output Delay	1	5	ns	50pF
SData[63:0]	Output Delay	1	4.5	ns	30pF
SData[63:0], ECC[7:0]	Output Delay	1	6	ns	30pF
sdclckout	Output Delay	1.5	3.5	ns	50pF
SDRAM Interface (SDClkOut) **NOTE:** All SDRAM interface Output Delays are referred to the **SDClkOut** rising edge.					
ECC[7:0], SData[63:0]	Setup	2		ns	
SData[63:0], ECC[7:0]	Hold	0.5		ns	

Table 430: 100 MHz AC Timing (Continued)

		100MHz			
Signals	**Description**	**Min.**	**Max.**	**Units**	**Loading**
SData[63:0], ECC[7:0]	Output Delay	0.5	3	ns	30pF
SDQM*[7:0]	Output Delay	0.5	2	ns	30pF
BankSel[1:0], DAdr[12:0], SRAS*, SCAS*, SCS[3:0]*, SDQM*[7:0], DWr*	Output Delay	0.5	2	ns	50pF
Device Interface **NOTE:** All Device interface Output Delays, Setup, and Hold times are referred to TClk rising edge.					
AD[31:0]	Setup	2		ns	
Ready*	Setup	3		ns	
AD[31:0], Ready*	Hold	1		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 125.					
CSTiming*, Wr[3:0]*, ALE, AD[31:0]	Output Delay	1	5.5	ns	30pF
BAdr[2:0]	Output Delay	1	6	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	TBD		ns	20pF
PCIReq[5:0]*	Setup	TBD		ns	20pF
MREQ*, MGNT*, EOT[7:0], DMAReq[7:0]*, TCEn[7:0],	Hold	TBD		ns	20pF
PCIReq[5:0]*	Hold	TBD		ns	20pF
GPP[31:0]	Hold	TBD		ns	20pF
DBurst*/Dlast*	Output Delay	TBD	TBD	ns	20pf
MREQ*,MGNT*, TCTCnt[7:0], GPP[31:0], InitAct	Output Delay	TBD	TBD	ns	20pf
PCIGnt[5:0]*	Output Delay	TBD	TBD	ns	20pf
DMAAck[7:0]*,	Output Delay	TBD	TBD	ns	20pF
PME, INT[4:0], WDE*, WDNMI* are asyncronus signals.					

Table 430: 100 MHz AC Timing (Continued)

Signals	Description	100MHz Min.	100MHz Max.	Units	Loading
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

24. Pinout Table, 665 Pin BGA

NOTE: The following table is sorted by ball number.

Table 431: GT–64242 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 431: GT–64242 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 431: GT–64242 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 431: GT–64242 Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
K01–K07, K25–K31		***M01–M06, M26–M31***		***N26–N31***	
K01	GND	M01	GND	N26	VCC Core
K02	GND	M02	GND	N27	SysAD[28]
K03	GND	M03	GND	N28	SysAD[29]
K04	GND	M04	GND	N29	NC
K05	AD[31]	M05	GND	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	GND
K26	NC	M28	SysADC[5]	P02	GND
K27	NC	M29	SysADC[3]	P03	NC
K28	NC	M30	SysAD[22]	P04	NC
K29	NC	M31	SysAD[24]	P05	NC
K30	NC	***N01–N06, N13–N19***		P06	NC
K31	NC	N01	GND	P13	GND
L01–L06, L26–L31		N02	NC	P14	GND
L01	GND	N03	NC	P15	GND
L02	GND	N04	NC	P16	GND
L03	GND	N05	GND	P17	GND
L04	GND	N06	VCC 3.3	P18	GND
L05	GND	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 431: GT–64242 Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
R01–R06, R13–R19, R26–R31		***T13–T31***		***U26–U31***	
R01	GND	T13	GND	U26	VCC 2.5
R02	GND	T14	GND	U27	SysAD[60]
R03	GND	T15	GND	U28	SysAD[44]
R04	GND	T16	GND	U29	SysADC[1]
R05	GND	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	GND
R16	GND	T28	SysAD[3]	V03	GND
R17	GND	T29	SysADC[0]	V04	GND
R18	GND	T30	SYSAD[17]	V05	GND
R19	GND	T31	SysAD[12]	V06	NC
R26	VCC 2.5	***U01–U06, U13–U19***		V13	GND
R27	SysAD[30]	U01	GND	V14	GND
R28	SysADC[6]	U02	NC	V15	GND
R29	SysAD[7]	U03	GND	V16	GND
R30	SysAD[50]	U04	GND	V17	GND
R31	SysAD[42]	U05	GND	V18	GND
T01–T06		U06	VCC Core	V19	GND
T01	GND	U13	GND	V26	SysAD[15]
T02	GND	U14	GND	V27	SysAD[19]
T03	GND	U15	GND	V28	SysAD[11]
T04	GND	U16	GND	V29	SysAD[57]
T05	GND	U17	GND	V30	SysAD[58]
T06	GND	U18	GND	V31	SysAD[46]
		U19	GND		

Table 431: GT–64242 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 431: GT–64242 Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
AD05–AD07, AD25–AD31		***AE30–AE31***		***AF27–AF31***	
AD05	MPP[27]	AE30	PAD[36]	AF27	PAD[38]
AD06	MPP[26]	AE31	PAD[37]	AF28	PAD[39]
AD07	VCC 3.3	***AF01–AF26***		AF29	PAD[40]
AD25	VCC Core	AF01	NC	AF30	PAD[41]
AD26	SysAD[53]	AF02	NC	AF31	PAD[42]
AD27	SysAD[49]	AF03	NC	***AG01–AG23***	
AD28	SysAD[40]	AF04	NC	AG01	NC
AD29	SysAD[38]	AF05	NC	AG02	NC
AD30	SysAD[34]	AF06	NC	AG03	GND
AD31	SysAD[0]	AF07	NC	AG04	NC
AE01–AE10, AE22–AE29		AF08	NC	AG05	NC
AE01	NC	AF09	NC	AG06	NC
AE02	NC	AF10	NC	AG07	NC
AE03	GND	AF11	VCC 3.3	AG08	NC
AE04	CLK	AF12	GND	AG09	NC
AE05	Rst*	AF13	VCC 3.3	AG10	NC
AE06	NC	AF14	NC	AG11	NC
AE07	GND	AF15	VCC 3.3	AG12	NC
AE08	VREF	AF16	REQ*	AG13	NC
AE09	VCC Core	AF17	VREF	AG14	NC
AE10	VCC Core	AF18	VCC 3.3	AG15	NC
AE22	VCC 3.3	AF19	GND	AG16	GNT*
AE23	VCC 3.3	AF20	IRDY*	AG17	PAD[27]
AE24	VCC Core	AF21	VCC 3.3	AG18	IDSEL
AE25	VCC 3.3	AF22	PAD[12]	AG19	PAD[19]
AE26	PAD[32]	AF23	PAD[7]	AG20	Frame*
AE27	PAD[33]	AF24	PAD[1]	AG21	SERR*
AE28	PAD[34]	AF25	HS	AG22	PAD[13]
AE29	PAD[35]	AF26	PAR64	AG23	CBE[0]*

Table 431: GT–64242 Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
AG24–AG31		***AH21–AH31***		***AJ18–AJ31***	
AG24	PAD[2]	AH21	PERR*	AJ18	PAD[24]
AG25	LED	AH22	PAD[14]	AJ19	PAD[21]
AG26	CBE[4]*	AH23	PAD[8]	AJ20	PAD[16]
AG27	PAD[59]	AH24	PAD[3]	AJ21	STOP*
AG28	PAD[43]	AH25	ENUM*	AJ22	PAD[15]
AG29	PAD[44]	AH26	P64EN*	AJ23	PAD[9]
AG30	PAD[45]	AH27	PAD[60]	AJ24	PAD[4]
AG31	PAD[46]	AH28	PAD[55]	AJ25	REQ64*
AH01–AH20		AH29	PAD[47]	AJ26	CBE[5]*
AH01	NC	AH30	PAD[48]	AJ27	PAD[61]
AH02	NC	AH31	PAD[49]	AJ28	PAD[56]
AH03	NC	***AJ01–AJ17***		AJ29	PAD[52]
AH04	NC	AJ01	NC	AJ30	PAD[51]
AH05	NC	AJ02	NC	AJ31	PAD[50]
AH06	NC	AJ03	NC	***AK02–AK15***	
AH07	NC	AJ04	VCC 3.3	AK02	VCC 3.3
AH08	NC	AJ05	NC	AK03	Pull-up
AH09	NC	AJ06	NC	AK04	VCC 3.3
AH10	NC	AJ07	NC	AK05	NC
AH11	NC	AJ08	NC	AK06	NC
AH12	NC	AJ09	NC	AK07	NC
AH13	NC	AJ10	NC	AK08	NC
AH14	NC	AJ11	NC	AK09	Pull-up
AH15	NC	AJ12	NC	AK10	NC
AH16	CLK	AJ13	NC	AK11	NC
AH17	PAD[28]	AJ14	NC	AK12	NC
AH18	CBE[3]*	AJ15	NC	AK13	NC
AH19	PAD[20]	AJ16	Rst*	AK14	NC
AH20	CBE[2]*	AJ17	PAD[29]	AK15	NC

Table 431: GT–64242 Pinout Table (Continued)

Ball #	Signal Name	Ball #	Signal Name	Ball #	Signal Name
AK16–AK30		***AL03–AL17***		***AL18–AL29***	
AK16	INT*	AL03	Pull-up	AL18	PAD[26]
AK17	PAD[30]	AL04	VCC 3.3	AL19	PAD[23]
AK18	PAD[25]	AL05	VCC 3.3	AL20	PAD[18]
AK19	PAD[22]	AL06	NC	AL21	TRDY*
AK20	PAD[17]	AL07	NC	AL22	PAR
AK21	DEVSEL*	AL08	NC	AL23	PAD[11]
AK22	CBE[1]*	AL09	VCC 3.3	AL24	PAD[6]
AK23	PAD[10]	AL10	NC	AL25	PAD[0]
AK24	PAD[5]	AL11	NC	AL26	CBE[7]*
AK25	ACK64*	AL12	NC	AL27	PAD[63]
AK26	CBE[6]*	AL13	NC	AL28	PAD[58]
AK27	PAD[62]	AL14	NC	AL29	PAD[54]
AK28	PAD[57]	AL15	NC		
AK29	PAD[53]	AL16	NC		
AK30	VCC 3.3	AL17	PAD[31]		

Figure 50: GT–64242 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	GND	GND	GND	GND	AD[31]	VCC 3.3	VCC 3.3									K
L	GND	GND	GND	GND	GND	VCC 3.3										L
M	GND	GND	GND	GND	GND	VCC 3.3										M
N	GND	NC	NC	NC	GND	VCC 3.3							GND	GND	GND	N
P	GND	GND	NC	NC	NC	NC							GND	GND	GND	P
R	GND	GND	GND	GND	GND	GND							GND	GND	GND	R
T	GND	GND	GND	GND	GND	GND							GND	GND	GND	T
U	GND	NC	GND	GND	GND	VCC Core							GND	GND	GND	U
V	I2CSCK	GND	GND	GND	GND	NC							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									AD
AD	MPP[31]	MPP[30]	MPP[29]	MPP[28]	MPP[27]	MPP[26]	VCC 3.3									AD
AE	NC	NC	GND	CLK	Rst*	NC	GND	VREF	VCC Core	VCC Core						AE
AF	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	VCC 3.3	GND	VCC 3.3	NC	VCC 3.3	AF
AG	NC	NC	GND	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	AG
AH	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	AH
AJ	NC	NC	NC	VCC 3.3	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	NC	AJ
AK		VCC 3.3	Pull-up	VCC 3.3	NC	NC	NC	NC	Pull-up	NC	NC	NC	NC	NC	NC	AK
AL			Pull-up	**VCC 3.3**	VCC 3.3	NC	NC	NC	VCC 3.3	NC	NC	NC	NC	NC	NC	AL
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

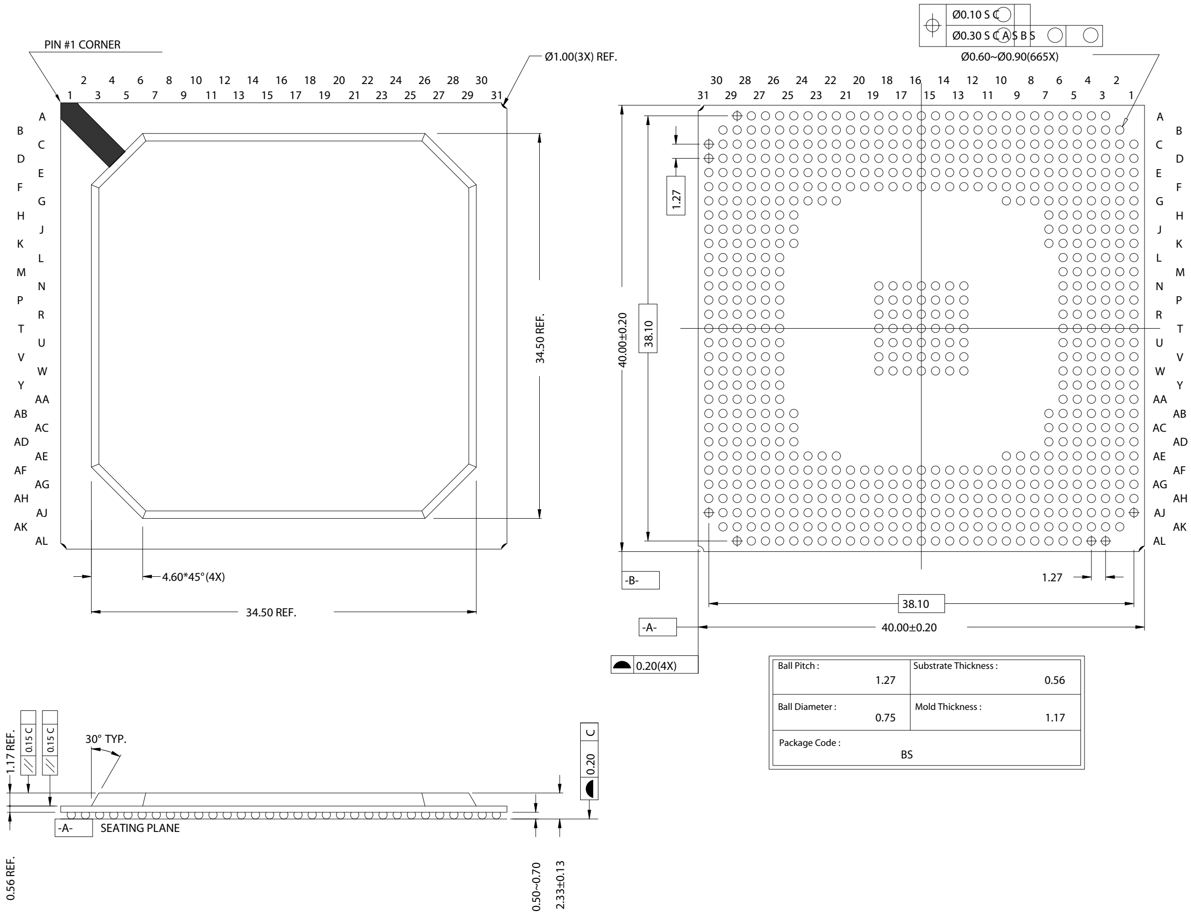
NOTE: VCC=VDD, GND=VSS, NC=Not Connected

Figure 51: GT–64242 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
AD										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	PAD[32]	PAD[33]	PAD[34]	PAD[35]	PAD[36]	PAD[37]	AE
AF	REQ*	VREF	VCC 3.3	GND	IRDY*	VCC 3.3	PAD0[12]	PAD[7]	PAD[1]	HS	PAR64	PAD[38]	PAD[39]	PAD[40]	PAD[41]	PAD[42]	AF
AG	GNT*	PAD[27]	IDSEL	PAD[19]	Frame*	SERR*	PAD[13]	CBE[0]*	PAD[2]	LED	CBE[4]*	PAD[59]	PAD[43]	PAD[44]	PAD[45]	PAD[46]	AG
AH	CLK	PAD[28]	CBE[3]*	PAD[20]	CBE[2]*	PERR*	PAD[14]	PAD[8]	PAD[3]	ENUM*	P64EN	PAD[60]	PAD[55]	PAD[47]	PAD[48]	PAD[49]	AH
AJ	Rst*	PAD[29]	PAD[24]	PAD[21]	PAD[16]	STOP*	PAD[15]	PAD[9]	PAD[4]	REQ64*	CBE[5]*	PAD[61]	PAD[56]	PAD[52]	PAD[51]	PAD[50]	AJ
AK	INT*	PAD[30]	PAD[25]	PAD[22]	PAD[17]	DEVSEL*	CBE[1]*	PAD[10]	PAD[5]	ACK64*	CBE[6]*	PAD[62]	PAD[57]	PAD[53]	VCC 3.3		AK
AL	NC	PAD[31]	PAD[26]	PAD[23]	PAD[18]	TRDY*	PAR	PAD[11]	PAD[6]	PAD[0]	CBE[7]*	PAD[63]	PAD[58]	PAD[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

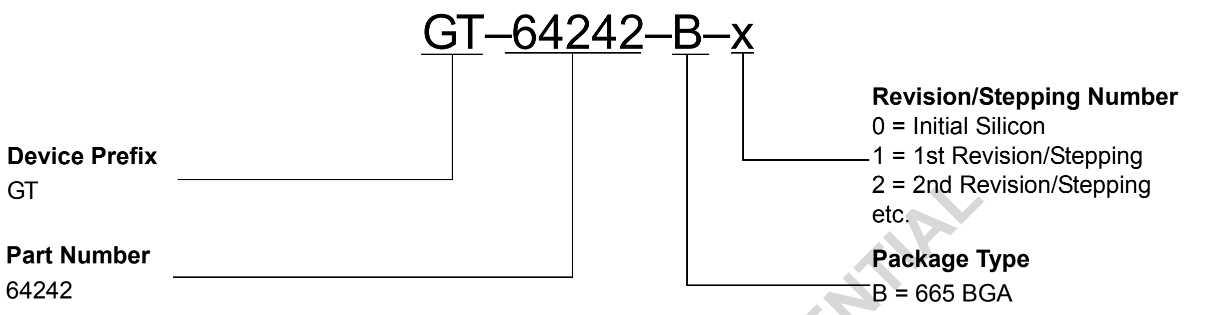
25. 665 PBGA Package Mechanical Information



Ball Pitch : 1.27	Substrate Thickness : 0.56
Ball Diameter : 0.75	Mold Thickness : 1.17
Package Code : BS	

26. GT–64242 PART NUMBERING

Figure 52: Sample Part Number



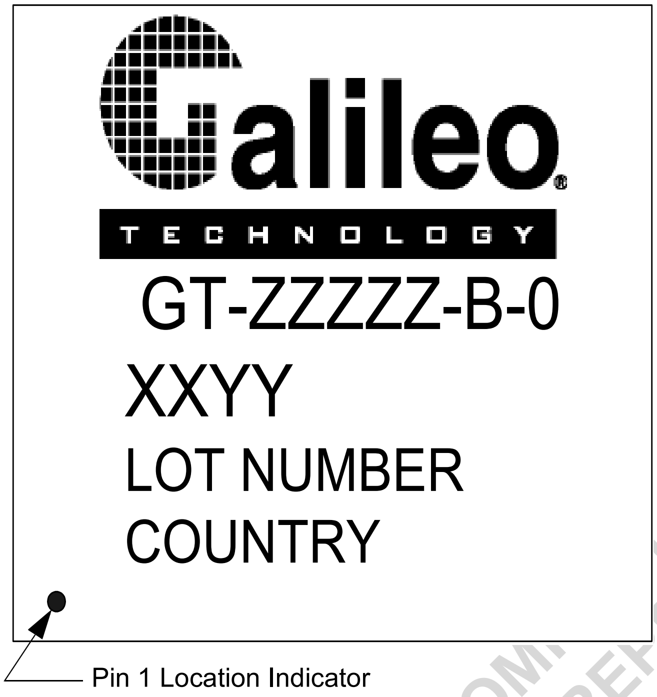
The part number for the GT–64242 is GT–64242-B-x.

This part number indicate that this is the commercial temperature grade, 100MHz version.

This is the only valid part number that can be used when ordering the GT–64242.

27. GT–64242 Part Marking and Pin Location

Figure 53: Package Markings and Pin 1 Location



<u>**Nomenclature:**</u>

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

28. REVISION HISTORY

Table 432: Revision History

Document Type	Revision	Date
Preliminary Datasheet	A	September 06, 2001
First datasheet revision.		