GT-64130

System Controller for PowerPC Processors

Product Review
Revision 1.1
DEC 15, 1999

Please contact Galileo Technology for possible updates before finalizing a design.

FEATURES

- Integrated system controller with PCI interface for high-performance embedded control applications.
- Supports Motorola/IBM PowerPC 64 bit CPUs: MPC603e/604e/740/750.
- Supports Motorola's PowerQUICC (MPC860) and PowerQUICC II (MPC8260) CPU.
- Up to 75MHz CPU bus frequency.
- Supports write-back or write-through L2 cache.
- Supports cache coherency on 60X bus in multi-CPU system.
- Supports multiple GT-64130 devices on the same 60X bus (up to 4).
- 64-byte CPU write posting buffer.
  - 64-bit wide, 8 levels deep.
  - Accepts CPU writes with zero wait-states.
- CPU address remapping to resources.
- SDRAM controller:
  - 3.3V (5V tolerant).
  - 512MB address space.
  - Supports 2-way & 4-way SDRAM bank interleaving.
  - Supports 16/64/128 Mbit SDRAM.
  - 256KB-256MB device depth.
  - 1- 4 banks supported.
  - 64-bit data width.
  - ECC support.
  - Zero wait-state interleaved burst accesses at 75MHz.
- Supports the VESA Unified Memory Architecture (VUMA) Standard.
  - Allows for external masters access to SDRAM directly.
- Device controller:
  - 5 chip selects.
  - Programmable timing for each chip select .
  - Supports many types of standard memory and I/O devices.
  - Up to 512MB address space.
  - Optional external wait-state support.
  - 8-,16-,32- and 64-bit width device support
  - Support for boot ROMs.
- Four channel DMA controller:
  - Chaining via linked-lists of records.
  - Byte address boundary for source and destination.
  - Moves data between PCI, memory, and devices.
  - Two 64-byte internal FIFOs allowing two transfers to take place concurrently.
  - Alignment of source and destination addresses.
  - DMAs can be initiated by the CPU writing to a register, external request via DMAReq* pin, or an internal timer/counter.
  - Termination of DMA transfer on each channel.
  - Descriptor ownership transfer to CPU.
  - Fly-By support for local data bus.
  - Override capability of source/destination/record address mapping.
- Two 32-bit or one 64-bit high-performance PCI 2.1 compliant devices.
  - Dual mode PCI interface can be used as two independent 32-bit interfaces (synchronous or asynchronous to each other) or as a single 64-bit interface.
  - 192-bytes of posted write and read prefetch buffers for each PCI interface.
  - 32/64-bit PCI master and target operations.
  - PCI bus speed of up to 66MHz with no wait states.
  - Universal PCI buffers.
  - Operates either synchronous or asynchronous to CPU clock.
  - Burst transfers used for efficient data movement.

  - Doorbell interrupts provided between CPU and PCI.
  - Supports flexible byte swapping through PCI interface.
  - Synchronization barrier support for PCI side.
  - PCI address remapping to resources.
  - PCI configuration registers can be accessed from both CPU and PCI side.
- Host to PCI bridge:
  - Translates CPU cycles into PCI I/O or Memory cycles.
  - Generates PCI Configuration, Interrupt Acknowledge, and Special cycles on PCI bus.
- Commercial Speed Grade offered at 75MHz and 66MHz.
- Industrial Speed Grade offered at 66MHz
- PCI to Main Memory bridge:
  - Supports fast back-to-back transactions.
  - Supports memory and I/O transactions to internal configuration registers.
  - Supports locked operations.
- PCI Hot-Plug and CompactPCI Hot-Swap capable compliant:
  - I_2O Industry Standard I_2O messaging unit on primary 32-bit PCI interface (also available in 64-bit mode).
  - Expansion ROM support.
- One 32-bit wide timer/counter, three 24-bit wide timer/counters.
- 3.3 V VCC and I/O
  - All inputs are 5V tolerant.
- Advanced 0.35 micron process.

Part Number: GT-64130
Publication Revision: 1.1

Galileo Technology, Inc.
142 Charcot Avenue
San Jose, California 95131
Phone: 1 408 367-1400
Fax: 1 (408) 367-1401
E-mail: info@galileot.com
www.galileoT.com

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

LIST OF TABLES

LIST OF TABLES (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

1. OVERVIEW

The GT-64130 provides a single-chip solution for designers building systems with PowerPC CPUs. These CPUs include the MPC603e, MPC604e, MPC740/750, PowerQUICC, and PowerQUICC II.

The architecture of the GT-64130 supports several system implementations for different applications and cost and performance points. It is possible to design a powerful system with minimal glue logic or add commodity logic (controlled by the GT-64130) for differentiated system architectures that attain higher performance.

The GT-64130 has a three or four bus architecture:

- A 64-bit interface to the CPU bus (multiplexed address/data).
- A 64-bit interface to the memory and device subsystem.
- Two independent 32-bit PCI interfaces or one 64-bit PCI interface.

The buses are de-coupled from each other in most accesses. This enables concurrent operation of the CPU bus, PCI devices, and accesses to memory. For example, the GT-64130 can simultaneously control the CPU bus writing to the on-chip write buffer, a DMA agent moving data from SDRAM to its own buffers, and a PCI device writing into an on-chip FIFO.

1.1 CPU Bus Interface

The GT-64130 A/D bus allows the CPU to access the PCI and memory and device buses. The CPU bus protocol supports all 60x-bus transfer sizes as well as 32-byte burst reads and writes (16-bytes in the case of MPC860). With a maximum frequency of 75MHz, the CPU can transfer in excess of 600 Mbytes/sec.

The GT-64130 has a 64-bit multiplexed address and data bus. Multiplex of CPU A[0-31] and DL[0-31] is done outside the GT-64130.

The GT-64130 supports two multiplex configurations, with 32-bit QuickSwitch mux or without 32-bit QuickSwitch mux. In QuickSwitch configuration the GT-64130 supports 1.5 address pipeline depth. The CPU is allowed to generate a new transaction address before a previous transaction data tenure is done. When working with MPC860 CPU, there is no need for address/data mux since it is a 32 bit CPU.

The GT-64130 also supports a 603e 32-bit bus configuration. In this mode, there is no need for external address/data mux.

The GT-64130 allows the CPU to connect to up to four GT-64130 devices. This increases the address space and flexibility of the system design significantly.

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

The GT-64130 also supports CPU address remapping to resources and configurations to operate in little or big endian mode.

The GT-64130 can also be used for multi CPU systems.

1.2 SDRAM and Device Interface

The GT-64130 has an SDRAM controller with a 64-bit wide interface. The SDRAM controller supports industry standard 16, 64, and 128 Mbit SDRAMs.

The SDRAM controller is 3.3V and 5V tolerant, works at frequencies up to 75MHz, and can address up to 512MBytes. Up to four banks of SDRAM may be connected. The controller supports 2 bank interleaving for 16 Mbit SDRAMs and 2 or 4 bank interleaving for 64/128 Mbit SDRAMs.

It also supports a UMA feature which enables external masters to arbitrate for direct access to SDRAM. This feature enhances system performance and gives flexibility when designing shared memory systems.

The GT-64130 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 Flash, EPROMs, FIFOs, and I/O controllers. Device widths of 8, 16, 32, and 64 bits are supported.

The GT-64130 supports ECC on 64-bit wide 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.

The GT-64130 also drives odd parity to CPU on read transaction.

1.3 PCI Interface

The GT-64130 interfaces directly with the PCI bus. It can be configured to have two 32-bit PCI devices (PCI_0 and PCI_1) or as one 64-bit PCI device (PCI_0) operating at a maximum frequency of 66MHz. Each PCI device of the GT-64130 can be either a master initiating a PCI bus operation or a target responding to a PCI bus operation.

The GT-64130 incorporates 192-bytes of posted write and read prefetch buffers per PCI interface for efficient data transfer between the CPU/DMA to PCI and PCI to main memory.

The GT-64130 becomes a PCI bus master when the CPU or the internal DMA engine initiates a bus cycle to a PCI device. The following PCI bus cycles are supported:

- Memory Read/Write
- Interrupt Acknowledge
- Special Cycle
- I/O Read/Write
- Configuration Read/Write

The GT-64130 acts as a target when a PCI device initiates a memory access or an I/O access, in the case of internal registers. It responds to all memory read/write accesses and all configuration and I/O cycles, in the case of internal registers. It is possible to program the PCI slave to retry all PCI transactions targeted to the GT-64130. The PCI slave performs PCI address remapping to resources.

The GT-64130 contains the required PCI configuration registers. All internal registers, including the PCI configuration registers, can be accessed from both the CPU bus and the PCI bus. The GT-64130 configuration register set is Plug and Play compatible, with industry standard I_2O support.

The GT-64130 also supports PCI Hot-Plug and CompactPCI Hot Swap Capable requirements.

The PCI interface can operate at 66MHz and is Universal PCI compatible (3.3V/5V).

The GT-64130 can also act as a PCI to Memory bridge, even without the presence of the CPU.

1.4 DMA Engines

GT-64130 incorporates four high performance DMA engines. Each DMA engine has the capability to transfer data between PCI devices, between PCI and main memory, or between devices residing on the 64-bit device bus.

The DMA uses two internal 64-byte FIFOs for temporary storage of DMA data. The pair of FIFOs allows two DMA channels to be working concurrently with each channel utilizing a FIFO. For example, while channel 0 reads data from SDRAM into one FIFO another FIFO writes data from the other FIFO to the PCI bus.

Source and destination addresses can be non-aligned on any byte address boundary. The DMA channels can be programmed by the CPU , or by PCI masters, or without CPU/PCI intervention via a linked list of records. This linked list is loaded by the DMA controller into the channel's working set when a DMA transaction ends. The DMA supports increment/decrement/hold on source and destination addresses independently and alignment of addresses towards source and destination. In addition, the GT-64130 provides an override capability of the source, destination, and record address mapping to force access to PCI_0 or PCI_1.

A DMA can be initiated:

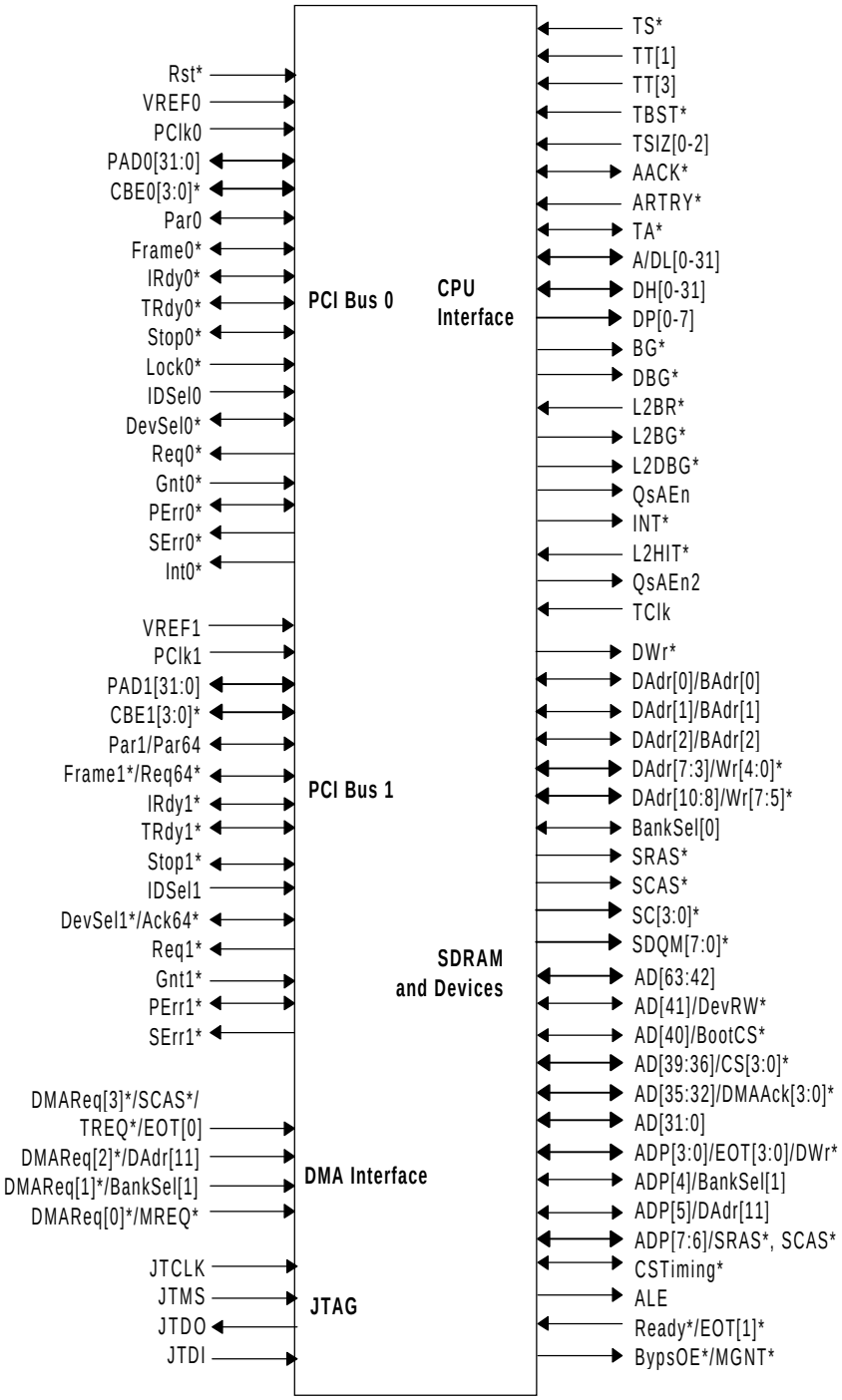
- By the CPU writing to a register.
- An external request via a DMAReq* pin
- An internal timer/counter terminal out.

Four End of Transfer pins act as inputs to the GT-64130 that allows for the terminating of a DMA transfer on a certain channel. In case of chained mode, after the transfer is ended, it is possible to transfer the descriptor to CPU ownership which calculates the number of remaining bytes in the buffer associated with the closed descriptor.

Fly-By DMA is also supported. This allows data to be transferred directly between two residents on the device or memory bus without having to go into a DMA FIFO. This effectively doubles the bandwith.

2. PIN INFORMATION

Figure 1: Pin Information



2.1 Pin Assignment Table

Table 1: Pin Assignments

Pin Name	I/O	Full Name	Description
CPU Interface (603e/740/750)			
TClk	I	Clock	The input clock to the GT-64130 (up to 75MHz). TClk is used for both the CPU and SDRAM interface. TClk must be driven for all applications, including those that do not use the CPU bus.
TS*	I	Transfer Start	Asserted on first address tenure cycle. The CPU drives valid address and attributes starting TS* assertion until the GT-64130 asserts AACK*.
TBST*	I	Transfer Burst	Signals that a burst transaction is in progress.
TSIZ[0-2]	I	Transfer Size	The number of bytes to be transferred. If TBST* is asserted, it is a 32-byte transfer regardless of TSIZ value
TT[1],TT[3]	I	Transfer type	TT[1] indicates read/write (read = 1). TT[3] indicates address-only transaction (address-only = 0).
A/DL[0-31]	I/O	Address/Data-Low Bus	A 32-bit multiplexed address and data bus for communication between the CPU and the GT-64130.
DH[0-31]	I/O	Data-High Bus	A 32-bit data-high bus for data transfer between the CPU and GT-64130.
DP[0-7]]	O	Data Parity	8-bit odd parity generated by the GT-64130 with returned read data (bit per byte).
Interrupt*	I/O	CPU Interrupt	An “OR” of all the internal interrupt sources on the GT-64130.
AACK*	O	Address Acknowledge	The GT-64130 signals the CPU that it can stop driving address and attributes. **NOTE:** This pin MUST have pull-up resistors to VCC (4.7KOhm recommended).
BG*	O	Bus Grant	Allows the CPU to drive a new transaction on the bus the next cycle after BG* assertion.
DBG*	O	Data Bus Grant	Allows the CPU to drive data bus on write transaction or accept data on read transaction the next cycle after DBG* assertion.
L2BR*	I	L2 Bus Request	L2 cache request for bus mastership. Used for driving write-back of a modified line back to memory).
L2BG*	O	L2 Bus Grant	Allows L2 cache to drive a new transaction on the bus, the next cycle after BG* assertion.
L2DBG*	O	L2 Data Bus Grant	Allows L2 cache to drive data bus on write transaction or accept data on read transaction the next cycle after DBG* assertion.

Table 1: Pin Assignments (Continued)

Pin Name	I/O	Full Name	Description
CPU Interface (603e/740/750) (Continued)			
TA*	O	Transfer Acknowl-edge	The GT-64130 signals the CPU that there is valid data for the CPU on reads or that it sampled the data bus and that the CPU can drive next data on writes. **NOTE:** This pin MUST have pull-up resistors to VCC (4.7KOhm recommended).
QsAEn	O	Quick Switch Address Enable	QuickSwitch control signals. When asserted, QuickSwitch connects A[0-31]. When deasserted, QuickSwitch connect DL[0-31].
QsAEn2	O	Quick Switch Address Enable Duplicate	Duplicate of QsAEn for load balancing.
ARTRY*	I	Address Retry	Driven by the CPU to indicate a transaction termination. The initiator CPU should retry the transaction.
L2Hit*	I	L2 Cache Tag Match	Asserted by tag RAM on secondary cache tag match.
CPU Interface (PowerQUICC)			
TClk	I	Clock	The input clock to the GT-64130 (up to 75MHz). TClk is used for both the CPU and SDRAM interface. TClk must be driven for all applications, including those that do not use the CPU bus.
TS*	I	Transfer Start	Asserted on first address tenure cycle. CPU drives valid address and attributes starting TS assertion until end of transaction.
BURST*	I	Transfer Burst	Signals that a burst transaction is in progress.
TSIZ[0-1]	I	Transfer Size	The number of bytes to be transferred. If TBST* is asserted, it is a 16-byte transfer, regardless of TSIZ value.
RD/WR*	I	Read/Write	Indicates read/write transaction (read = 1).
A[0-31]	I	Address Bus	A 32-bit address bus.
D[0-31]	I/O	Data Bus	A 32-bit data bus for data transfer between CPU and the GT-64130.
DP[0-3]	O	Data Parity	4-bit odd parity generated by the GT-64130 with returned read data (bit per byte).
Interrupt*	I/O	CPU Interrupt	An “OR” of all the internal interrupt sources on the GT-64130.
BG*	O	Bus Grant	Allows the CPU to drive a new address tenure on the bus the next cycle after BG* assertion.
TA*	O	Transfer Acknowl-edge	The GT-64130 signals the CPU that there is a valid data for the CPU on reads or that it sampled data bus and the CPU can drive next data on writes. **NOTE:** This pin MUST have pull-up resistors to VCC (4.7KOhm recommended).

Table 1: Pin Assignments (Continued)

Pin Name	I/O	Full Name	Description
PCI Bus 0			
Rst*	I	Reset	Resets the GT-64130 to its initial state. This signal must be asserted for at least 10 TClk cycles. When in the reset state, all PCI output pins are put into tristate and all open drain signals are floated.
VREF0	I	PCI_0 Voltage Reference	This pin must be connected directly to the 3.3V or the 5V power plane depending on which voltage level PCI_0 supports. VREF0 and VREF1 can be completely independent voltage levels.
PClk0	I	PCI_0 Clock	Provides the timing for the PCI transactions. The PCI clock range is between 0 and 66MHz. The PClk cycle must be higher than TClk cycle by at least 1ns. See Section 22. ”AC Timing” on page 250 for more information.
PAD0[31:0]	I/O	PCI_0 Address/Data	32-bit multiplexed PCI address and data lines. During the first clock of the transaction, PAD0[31:0] contains a physical byte address (32-bits). During subsequent clock cycles, PAD0[31:0] contains data.
CBE0[3:0]*	I/O	PCI_0 Command/ Byte Enable	During the address phase of the transaction, CBE0[3:0]* provides the PCI bus command. During the data phase, these lines provide the byte enables.
Par0	I/O	PCI_0 Parity	Calculated by the GT-64130 as an even parity bit for the PAD0[31:0] and CBE0[3:0]* lines.
Frame0*	I/O	PCI_0 Frame	Asserted by the GT-64130 to indicate the beginning and duration of a master transaction. Frame0* asserts to indicate the beginning of the cycle. While Frame0* is asserted, data transfer continues. Frame0* deasserts to indicate that the next data phase is the final data phase transaction. Frame* is monitored by the GT-64130 when it acts as a target.
IRdy0*	I/O	PCI_0 Initiator Ready	Indicates the bus master’s ability to complete the current data phase of the transaction. A data phase is completed on any clock when both TRdy0* and IRdy0* are asserted. Wait cycles are inserted until TRdy0* and IRdy0* are asserted together.
TRdy0*	I/O	PCI_0 Target Ready	Indicates the target agent’s ability to complete the current data phase of the transaction. A data phase is completed on any clock when both TRdy0* and IRdy0* are asserted. Wait cycles are inserted until TRdy0* and IRdy0* are asserted together.

Table 1: Pin Assignments (Continued)

Pin Name	I/O	Full Name	Description
PCI Bus 0 (Continued)			
Stop0*	I/O	PCI_0 Stop	Indicates that the current target is requesting the bus master to stop the current transaction. As a master, the GT-64130 responds to the assertion of Stop0* by disconnecting, retrying or aborting. As a target, the GT-64130 asserts Stop0* to retry or disconnect.
Lock0*	I	PCI_0 Lock	Indicates an atomic operation that may require multiple transactions to complete. When the GT-64130 is a PCI target, Lock0* is sampled on the rising edge of the PClk when Frame0* is asserted. If Lock0* is sampled asserted, the GT-64130 enters a locked state and remains in this state until Lock0* is sampled deasserted on the following rising edge of PClk, when Frame0* is sampled asserted.
IdSel0	I	PCI_0 Initialization Device Select	Asserted to act as a chip select during PCI configuration read and write transactions.
DevSel0*	I/O	PCI_0 Device Select	Asserted by the target of the current access. When the GT-64130 is bus master, it expects the target to assert DevSel0* within five bus cycles, confirming the access. If the target does not assert DevSel0* within the required bus cycles, the GT-64130 aborts the cycle. As a target, when the GT-64130 recognizes its transaction, it asserts DevSel0* in a medium speed (two cycles after the assertion of Frame0*).
Req0*	O	PCI_0 Bus Request	Asserted by the GT-64130 to indicate to the PCI bus arbiter that it requires use of the PCI bus.
Gnt0*	I	PCI_0 Bus Grant	Indicates to the GT-64130 that access to the PCI bus is granted.
PErr0*	I/O	PCI_0 Parity Error	Asserted when a data parity error is detected. This pin features a sustained tri-state output.
SErr0*	O	PCI_0 System Error	Asserted when a serious system error (not necessarily a PCI error) is detected. The GT-64130 asserts the SErr0* two cycles after the failing address. This pin features an open-drain output.
Int0*	O	PCI_0 Interrupt Request	Asserted by the GT-64130 when one of the unmasked internal interrupt sources is asserted. This pin features an open-drain output.

Table 1: Pin Assignments (Continued)

Pin Name	I/O	Full Name	Description
PCI Bus 1			
VREF1	I	PCI_1 Voltage Reference	This pin must be connected directly to the 3.3V or the 5V power plane depending on which voltage level PCI_1 supports. VREF0 and VREF1 can be completely independent voltage levels.
PClk1	I	PCI_1 Clock	This pin provides the timing for the PCI-related bus transaction. The PCI clock range is between 0 and 66MHz. On double PCI configuration, this clock frequency can be independent of both TClk and PClk0. On 64-bit PCI configuration, it must be tied to PClk0. The PClk1 cycle must be higher than TClk cycle by at least 1ns. See Section 22.1 ”TClk/PClk Restrictions” on page 224 for more information.
PAD1[31:0]/ PAD0[63:32]	I/O	PCI_1 Address/Data	If PCI_1 is enabled during the first clock of the transaction, PAD1[31:0] contains a physical byte address (32 bits). During subsequent clock cycles, PAD1[31:0] contains data.
		PCI_0 (64-bit) Address Data	If PCI_0 is configured for 64-bit, these pins are PAD0[63:32], the most significant 32-bits of data for PCI_0 transactions.
CBE1[3:0]*/ CBE0[7:4]*	I/O	PCI_1 Bus Command/Byte Enable	If PCI_1 is enabled during the address phase of the transaction, CBE1[3:0]* provide the PCI bus command. During the data phase, these lines provide the byte enables.
		PCI_0 Bus (64-bit) Byte Enable	If PCI_0 is configured for 64-bit, these pins are CBE0[7:4]* and are byte enables for the most significant 32-bits of PCI_0 data phases.
Par1/Par64	I/O	PCI_1 Parity	If PCI_1 is enabled, Par1 is calculated by the GT-64130 as an even parity bit for the PAD1[31:0] and CBE1[3:0]* lines during address/data phases.
		PCI_0 (64-bit) Parity	If PCI_0 is configured for 64-bit, this pin is Par64 and functions as Parity for Upper WORD of data, an even parity bit for PAD0[63:32], and CBE0[7:4]*.
Frame1*/ Req64*	I/O	PCI_1 Frame	If PCI_1 is enabled, the GT-64130 asserts this pin to indicate the beginning and duration of a master transaction. Frame1* asserts to indicate the beginning of the cycle. While Frame1* is asserted, data transfer continues. Frame1* deasserts to indicate that the next data phase is the final data phase transaction. Frame1* is monitored by the GT-64130 when it acts as a target.
		PCI_0 (64-bit) Request 64	If PCI_0 is configured for 64-bit, this pin is Req64* and functions as a request for a 64-bit transaction. Req64* has the same timing as Frame0*. This pin is sampled on RESET to configure the GT-64130 prior to boot-up. See Section 11. ”Reset Configuration” on page 139 for more information.

Table 1: Pin Assignments (Continued)

Pin Name	I/O	Full Name	Description
PCI Bus 1 (Continued)			
IRdy1*	I/O	PCI_1 Initiator Ready	If PCI_1 is enabled, this pin indicates the bus master's ability to complete the current data phase of the transaction. A data phase is completed on any clock when both TRdy1* and IRdy1* are asserted. Wait cycles are inserted until TRdy1* and IRdy1* are asserted together.
TRdy1*	I/O	PCI_1 Target Ready	If PCI_1 is enabled, this pin indicates the target agent's ability to complete the current data phase of the transaction. A data phase is completed on any clock when both TRdy1* and IRdy1* are asserted. Wait cycles are inserted until TRdy1* and IRdy* are asserted together.
Stop1*	I/O	PCI_1 Stop	If PCI_1 is enabled, the target requesting the bus master asserts this pin to stop the current transaction. As a master, the GT-64130 PCI device1 responds to the assertion of Stop1* by disconnecting, retrying or aborting. As a target, the GT-64130 asserts Stop1* to retry or disconnect.
IdSel1	I	PCI_1 Initialization Device Select	If PCI_1 is enabled, this pin indicates a chip select for PCI_1 during PCI configuration read and write transactions.
DevSel1*/ Ack64*	I/O	PCI_1 Device Select	If PCI_1 is enabled, the target of the current access asserts this pin. When PCI_1 is bus master, it expects the target to assert DevSel1* within five bus cycles, confirming the access. If the target does not assert DevSel1* within the required bus cycles, the GT-64130 aborts the cycle. As a target, when the GT-64130 PCI device1 recognizes its transaction, it asserts DevSel1* in a medium speed (two cycles after the assertion of Frame1*).
		PCI_0 (64-bit) Acknowledge 64	If PCI_0 is configured for 64-bit, this signal functions as Ack64*. When actively driven by PCI target, it indicates the target is willing to accept 64-bit wide data. Ack64* has the same timing as Devsel0*.
Req1*	O	PCI_1 Bus Request	Asserted by the GT-64130 to indicate to the PCI bus arbiter that it requires use of the PCI bus.
Gnt1*	I	PCI_1 Bus Grant	Indicates to the GT-64130 that access to the PCI bus is granted.
PErr1*	I/O	PCI_1 Parity Error	Asserted when a data parity error is detected. This pin features a sustained tri-state output.
SErr1*	O	PCI_1 System Error	Asserted when a serious system error (not necessarily a PCI error) is detected. The GT-64130 asserts the SErr1* for two cycles after the failing address. This pin features an open drain output.

Table 1: Pin Assignments (Continued)

Pin Name	I/O	Full Name	Description
SDRAM & Devices			
DWr*	O	SDRAM Write	LOW when the GT-64130 writes to the SDRAM.
DAdr[0]/ BAdr[0]	I/O	SDRAM Address 0/ Burst Address 0	In an access to a SDRAM bank, this pin functions as a SDRAM address bit. In write and read accesses from devices that are 8-bit wide, this pin functions as byte address 0 in the packing process of data into 64-bits. In accesses to a word wide (32-bit) device, this bit functions as address 0 in a burst access. Not used for 16/64 bit devices. This pin is sampled on RESET to configure the GT-64130 prior to boot-up. See Section 11. ”Reset Configuration” on page 139 for more information.
DAdr[1]/ BAdr[1]	I/O	SDRAM Address [1] / Burst Address [1]	In SDRAM accesses, this pin functions as an address bit. In read accesses to devices that are 8-, or 16-bit wide, BAdr[2:1] functions as a half word address in the packing process of data into 64 bits. In accesses to a 32-bit bank, BAdr[2:1] functions as part of the (two MSB) burst address bits of an address into an eight word line or when packing/unpacking a 64-bit access. In accesses to a 64-bit bank, BAdr[2:1] functions as the two burst address bits of a four double word line. This pin is sampled on RESET to configure the GT-64130 prior to boot-up. See Section 11. ”Reset Configuration” on page 139 for more information.
DAdr[2]/ BAdr[2]	I/O	SDRAM Address [2] / Burst Address [2]	In SDRAM accesses, this pin functions as an address bit. In read access to devices that are 8- or 16-bit wide, BAdr[2:1] function as a half word address in the packing process of data into 64-bits. In accesses to a 32-bit bank, BAdr[2:1] functions as part of the two MSB, burst address bits of an address into an eight word line or when packing/unpacking a 64-bit access. In accesses to a 64-bit bank, BAdr[2:1] function as the two burst address bits of a four double word line. This pin is sampled on RESET to configure the GT-64130 prior to boot-up. See Section 11. ”Reset Configuration” on page 139 for more information.
DAdr[7:3]/ Wr[4:0]*	I/O	SDRAM Address [7:3] / Byte Write [4:0]	In SDRAM accesses these pins function as SDRAM address. In device writes, they function as byte write enable indications to the bank bytes [4:0]. This pin is sampled on RESET to configure the GT-64130 prior to boot-up. See Section 11. ”Reset Configuration” on page 139 for more information.

Table 1: Pin Assignments (Continued)

Pin Name	I/O	Full Name	Description
SDRAM & Devices (Continued)			
DAdr[10:8]/ Wr[7:5]*	I/O	SDRAM Address [10:8] / Byte Write [7:5]	In SDRAM accesses, these pins function as SDRAM address. In device writes, they function as byte write enable indications to the bank bytes [7:5]. This pin is sampled on RESET to configure the GT-64130 prior to boot-up. See Section 11. ”Reset Configuration” on page 139 for more information.
BankSel[0]	I/O	SDRAM Bank Select [0]	In SDRAM accesses, this pin functions as bank select bit[0]. This pin is sampled on RESET to configure the GT-64130 prior to boot-up. See Section 11. ”Reset Configuration” on page 139 for more information.
SRAS*	O	SDRAM Row Address Select	Asserts to indicate an active ROW address is on the DAdr lines.
SCAS*	O	SDRAM Column Address Select	Asserts to indicate an active column address is on the DAdr lines.
SCS[3:0]*	O	SDRAM Chip Selects	SDRAM chip selects for up to four banks.
SDQM[7:0]*	O	SDRAM Byte Enables	For AD bus to Write up to 64-bits of data to an SDRAM bank.
AD[63:42]	I/O	Address/Data[63:42]	In SDRAM accesses, these pins function as part of the data to be read/written from/to the SDRAMs. In Device accesses, these pins function as part of the data to be read/written from/to the Device.
AD[41]/ DevRW*	I/O	Address/Data [41] / Device Read-Write	In the data phase, it is data bit 41. In the address phase, it indicates if an access to a device is a read (1) or a write (0). Latching is done via ALE.
AD[40]/ BootCS*	I/O	Address/Data [40]/ Boot Chip Select	In the data phase, it is data bit 40. In the address phase, it is the boot device chip select. Latching is done via ALE.
AD[39:36]/ CS[3:0]*	I/O	Address/Data [39:36] / Chip Select [3:0]	In the data phase, these pins function as data bits [39:36]. In the address phase, Device Chip Selects are valid (and should be latched). The Chip Selects need to be qualified with the CSTiming* signal. Latching is done via ALE.
AD[35:32]/ DMAAck[3:0]*	I/O	Address/Data [35:32] / DMA Acknowledge[3:0]	In the data phase, these pins function as data bits [35:32]. In the address phase, DMA Acknowledges are valid (and should be latched). They need to be qualified with the CSTiming* signal. Latching is done via ALE.
AD[31:0]	I/O	Address/Data[31:0]	Multiplexed address and data bus to the SDRAM (data only) and the devices (address and data).

Table 1: Pin Assignments (Continued)

Pin Name	I/O	Full Name	Description
SDRAM & Devices (Continued)			
ADP[3:0]/ EOT[3:0]/DWr*	I/O	AD Bus ECC [3:0]	In SDRAM accesses, these bits serve as ECC bits [3:0], generated by the GT-64130 for writes and read from the ECC bank.
		End of DMA Transfer [3:0]	In DMA transfers, EOT[3:0] serves as End Of Transfer indications for the 4 DMA channels. ADP[3] can be configured to function as DWr* on RESET. See Section 11. ”Reset Configuration” on page 139 for more information.
ADP[4]/Bank-Sel[1]	I/O	AD Bus ECC [4]	In SDRAM accesses these bits serves as ECC bit [4], generated by GT-64130 for writes and read from ECC bank.
		Bank Select [1]	In the SDRAM address phase, serves as bank select bit[1] (64/128Mbit SDRAM).
ADP[5]/ DAdr[11]	I/O	AD Bus ECC [5]	In SDRAM accesses, these bit serves as ECC bit[5], generated by the GT-64130 for writes and read from the ECC bank.
		SDRAM Address [11]	In the SDRAM address phase, serves as the upper order DRAM address bit.
ADP[7:6]/ SRAS*, SCAS*	I/O	AD Bus ECC [7:6]	In SDRAM accesses, these bits serve as ECC bits [7:6], generated by the GT-64130 for writes and read from the ECC bank. ADP[7:6] can be configure to function as SRAS* and SCAS* on RESET. See Section 11. ”Reset Configuration” on page 139 for more information.
		SRAS*, SCAS*	ADP[7:6] can be configure to function as SRAS* and SCAS* on RESET. See Section 11. ”Reset Configuration” on page 139 for more information.
CSTiming*	I/O	Chip Select Timing	Active for the number of cycles that the device currently being accessed was programmed to in the respective device control register. Used to qualify the CS[3:0]*, BootCS and the DMAAck[3:0]* signals. This pin is sampled on RESET to configure the GT-64130 prior to boot-up. See the Reset Configuration Section for more info. See Section 11. ”Reset Configuration” on page 139 for more information.
ALE	O	Address Latch Enable	Used to latch the Address, BootCS*, CS[3:0]*, DevRW* and DMAAck[3:0]* pins from the AD bus.

Table 1: Pin Assignments (Continued)

Pin Name	I/O	Full Name	Description
SDRAM & Devices (Continued)			
Ready*/EOT[1]	I	Ready	A cycle extender. When inactive during a device access, an access will extend until Ready* is asserted.
		End of Transfer [1]	Ready* can be programmed to function as EOT[1]*. See Section 5.1.2.3 ”Duplicating DMA End of Transfer Pins” on page 65. This pin is sampled on RESET to configure the GT-64130 prior to boot-up. See Section 11. ”Reset Configuration” on page 139 for more information.
BypsOE*/ MGNT*/DWr*	O	Bypass Output Enable	Controls the output enable to for bypass latches/buffers/ switches. The bypass can be used when a 64-bit read transaction is executed from the CPU and will be transferred directly.
		DRAM Write	BypsOE*/MGNT* can be programmed to function as DWr*. See Section 5.1.2 ”Duplicating Signals” on page 64.
		UMA Grant/Memory (AD) bus Grant	Asserted in response to MREQ*, in case UMA was activated at RESET.
DMA			
DMAReq[3]*/ SCAS*/ EOT[0]*/ TREQ*	I/O	DMA Request[3]	DMA request by external devices to channel 3. **NOTE:** This pin is sampled on RESET to configure the GT-64130 prior to boot-up. See Section 11. ”Reset Configuration” on page 139 for more information.
		SDRAM CAS/End of Transfer[0]	DMAReq[3]* can be programmed to function as SCAS* or as EOT[0]*.
		Total Request*	For UMA operation, it can be programed to indicate there is a pending internal request in DRAM and Device interface that requires the GT-64130’s ownership of AD bus. See Section 5.7.6 ”Total Request” on page 80.
DMAReq[2]*/ DAdr[11]	I/O	DMA Request[2]	DMA requests by external devices to channel 2.
		DRAM Address[11]	DMAReq[2]* can be programmed to function as DAdr[11]. See Section 5.1.2 ”Duplicating Signals” on page 64.
DMAReq[1]*/ BankSel[1]	I/O	DMA Request[1]	DMA requests by external devices to channel 1.
		Bank Select[1]	DMAReq[1]* can be programmed to function as BankSel[1]. See Section 5.1.2 ”Duplicating Signals” on page 64. This pin is sampled on RESET to configure the GT-64130 prior to boot-up. See Section 11. ”Reset Configuration” on page 139 for more information.

Table 1: Pin Assignments (Continued)

Pin Name	I/O	Full Name	Description
SDRAM & Devices (Continued)			
DMAReq[0]*/ MREQ*/ SRAS*	I/O	DMA Request [0]	DMA request indication by an external device.
		Memory Bus Request	Memory bus (AD) request by a device to support UMA.
		SDRAM RAS	This pin can be programmed to function as SRAS*. See Section 5.1.2 ”Duplicating Signals” on page 64. This pin is sampled on RESET to configure the GT-64130 prior to boot-up. See Section 11. ”Reset Configuration” on page 139 for more information.
JTAG Interface			
JTCLK	I	JTAG Clock	Clock for test logic. JTMS and JTDI are received on the rising edge. JTDO is driven from the falling edge. This signal determines the shifting rate. This input pin must be pulled LOW through a 4.7 KOhm resistor if it is not used.
JTMS	I	JTAG Mode Select	A broadcast signal which controls test logic operation. This input pin must be pulled HIGH through a 4.7 KOhm resistor if it is not used.
JTDO	O	JTAG Data Out	Serial data output. Tri-state changes on negative change of JTCLK. This output pin must be left UNCONNECTED if it is not used.
JTDI	I	JTAG Data In	Serial data input. This input pin must be pulled HIGH through a 4.7 KOhm resistor if it is not used.

2.2 603e/860 Pins Multiplex Table

Table 2: 603e/860 Pins Multiplex

603e	860
TS*	TS*
TT[1]	RD/WR*
TT[3]	----
TBST*	BURST*
TSIZ[1-2]	TSIZ[0-1]
TSIZ[0]	Must be pulled down.
TA*	TA*
QsAEn	----

Table 2: 603e/860 Pins Multiplex (Continued)

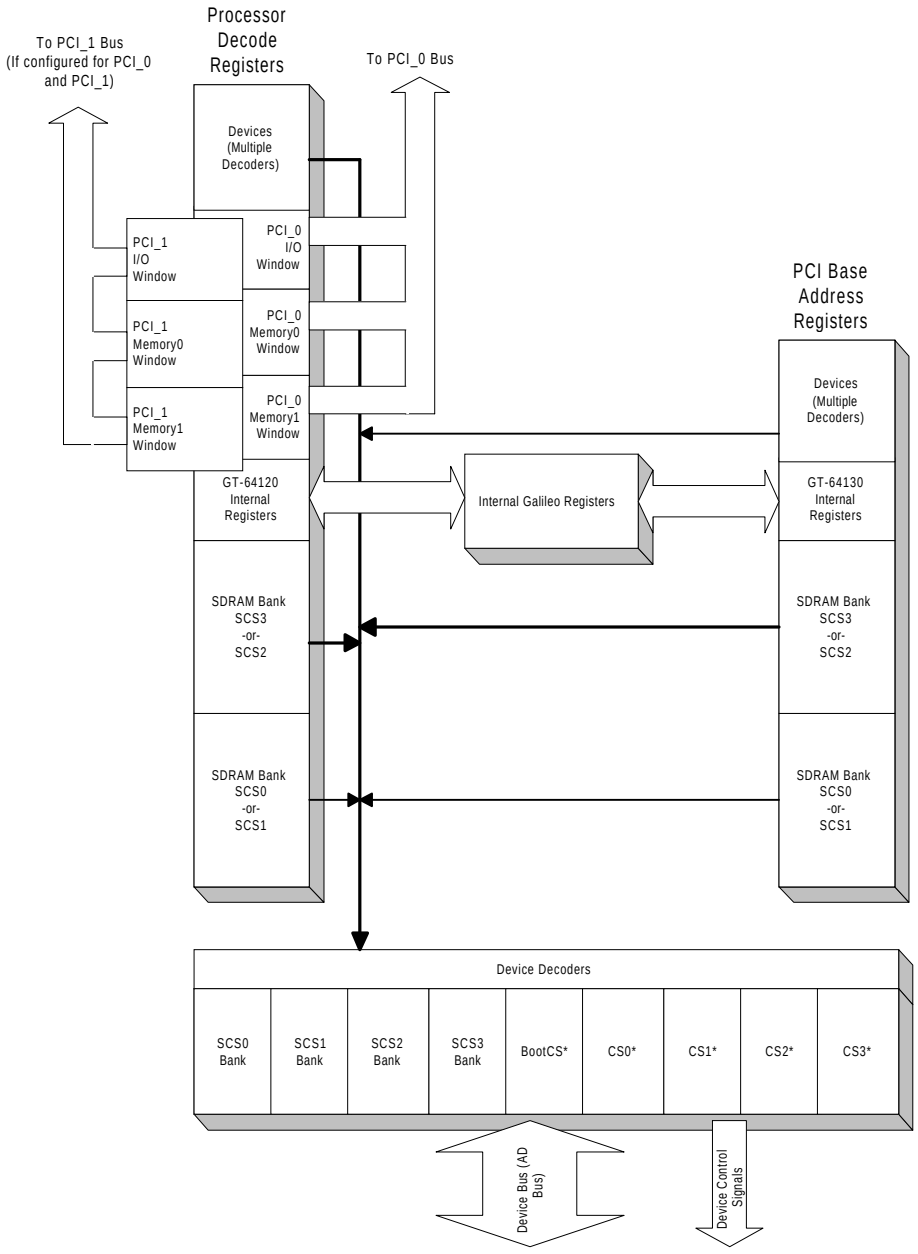
603e	860
QsAEn2	----
A/DL[0-31]	A[0-31]
DH[0-31]	D[0-31]
DP[0-3]	DP[0-3]
DP[4-7]	----
AACK*	----
BG*	BG*
DBG*	----
L2BR*	----
L2BG*	----
L2DBG*	----
ARTRY*	----
L2HIT*	----
Interrupt*	Interrupt*

3. ADDRESS SPACE DECODING

The GT-64130 has a fully programmable address map.

Two address spaces exist: the CPU address space and the PCI address space (see Figure 2.) Both address maps use a two-stage decoding process. First, major device regions are decoded. Second, individual devices are sub-decoded.

Figure 2: Two Stage Address Decoding- Conceptual View



3.1 Two Stage Decoding Process

The system resources are divided into the following groups:

- SCS[1:0]*
- SCS[3:2]*
- CS[2:0]*
- CS[3]* & BootCS*
- Internal Registers
- PCI_0 I/O
- PCI_0 Memory0/1
- PCI_1 I/O
- PCI_1 Memory0/1

NOTE: PCI_1 I/0 and PCI_1 Memory0/1 only exist if the GT-64130 is configured for both PCI_0 and PCI_1.

Each group can have a minimum of 2 Mbytes and a maximum of 256 Mbytes of address space. The individual devices in the device groups (e.g. SCS[0]*) are further sub decoded to 1 Mbyte resolution.

Table 3 shows the CPU decode and device sub-decode associations. The same processes are shown in Table 4, for PCI-0, and Table 5, for PCI_1.

Table 3: CPU and Device Decoder Mappings

CPU Decoder	Associated Device Sub-Decoders
SCS[1:0]*	SCS[0]* SCS[1]*
SCS[3:2]*	SCS[2]* SCS[3]*
CS[2:0]*	CS0* CS1* CS2*
BootCS*/CS3*	BootCS* CS3*
PCI_0 I/O	None Accesses decoded for PCI_0 I/O are bridged to PCI_0 I/O transfers.
PCI_0 Memory 0/1	None Accesses decoded for PCI_0 Memory 0/1 are bridged to PCI Memory transfers.
PCI_1 I/O	None Accesses decoded for PCI_1 I/O are bridged to PCI_1 I/O transfers.

Table 3: CPU and Device Decoder Mappings (Continued)

CPU Decoder	Associated Device Sub-Decoders
PCI_1 Memory 0/1	None Accesses decoded for PCI_1 Memory 0/1 are bridged to PCI_1 Memory transfers.
Internal	None Decodes to GT-64130 internal registers.

Table 4: PCI_0 Base Address Register and Device Decoder Mappings

PCI Base Address Register (BAR) Decoder[1]	Associated Device Sub-Decoders
SCS[1:0]* - BAR 0 at 0x10	SCS0* SCS1*
SCS[3:2]* - BAR 1 at 0x14	SCS2* SCS3*
CS[2:0]* - BAR 2 at 0x18	CS0* CS1* CS2*
BootCS*/CS3* - BAR 3 at 0x1C	BootCS* CS3*
Internal Registers (Memory) - BAR 4 at 0x20	None Decodes PCI_0 memory accesses to the GT-64130 internal registers.
Internal Registers (I/O) - BAR 5 at 0x24	None Decodes PCI_0 I/O accesses to the GT-64130 internal registers.
Expansion ROM - BAR at 0x30	None Decodes directly to CS3*.

1. This mapping also applies to the swap BARs located in PCI function 1, if enabled.

Table 5: PCI_1 Base Address Register and Device Decoder Mappings

PCI Base Address Register (BAR) Decoder[1]	Associated Device Sub-Decoders
SCS[1:0]* - BAR 0 at 0x90	SCS0* SCS1*
SCS[3:2]* - BAR 1 at 0x94	SCS2* SCS3*
CS[2:0]* - BAR 2 at 0x98	CS0* CS1* CS2*
BootCS*/CS3* - BAR 3 at 0x9C	BootCS* Cs3*
Internal Registers (Memory) - BAR 4 at 0xa0	None Decodes PCI_1 memory accesses to the GT-64130 internal registers.
Internal Registers (I/O) - BAR 5 at 0xa4	None Decodes PCI_1 I/O accesses to the GT-64130 internal registers.

1. This mapping also applies to the swap BARs located in PCI function 1, if enabled.

3.1.1 CPU Side Decoding Process

NOTE: When referring to an address in this section, the high number is the most significant bit and the lowest number is the least significant bit. For example, bit [31:0] means that the most significant bit is 31 and the least significant bit is 0.

This is opposite of the PowerPC convention in which the lowest bit number is the most significant bit and the highest bit number is the least significant bit.

Decoding on the CPU side starts with comparing the CPU address with the values in the various CPU Low and High decoder registers. For example, the SCS[1:0]* CPU High and Low decoder registers sets the address range in which the SCS0* and SCS1* signals are active (i.e., where DRAM banks 0 and 1 are located.) The comparison works as follows:

1. Bits 31:28 of the CPU address are compared against bits 10:7 in the various CPU Low decode registers. These values much match exactly. This effectively sets a 256 Mbyte “page” for the resource group.
2. Bits 27:21of the CPU address are then compared against bits 6:0 in the various CPU Low decode registers. The value of CPU address bits must be greater than or equal to the Low decode value. This sets the lower boundary for the region.
3. Next, bits 27:21of the CPU address are compared against the High decode registers. The value of CPU address bits must be less than or equal to this value. This sets the upper bound for the region.
4. If all of the above are true, the resource group is selected and a subdecode is performed to determine the

specific resource.

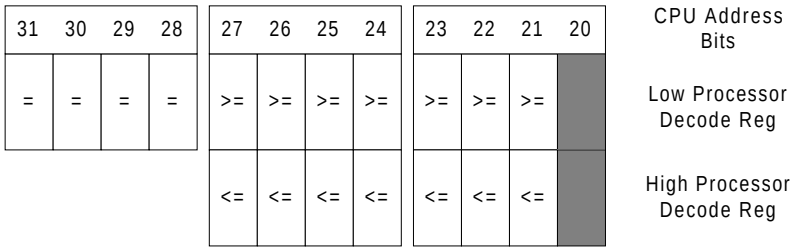
Once a CPU resource group has been decoded, it must be subdecoded to determine which physical device must be accessed within that group. This decoding is controlled by the device Low and High decode registers. The comparison works as follows:

1. Bits 27:20 of CPU address is compared against the relevant device Low decode registers. The value of CPU address bits must be greater than or equal to the Low decode value. This sets the lower boundary for the sub-decode region.
2. Bits 27:20 of CPU address are then compared against the relevant device High decode registers. The value of CPU address bits must be less than or equal to this value. This sets the upper bound for the sub-decode region.
3. If all of the above are true, the specific device is selected and an access to that device is performed.

Examples of the CPU-side decode process are shown in Figure 3 and Figure 4.

Figure 3: CPU-Side Resource Group Decode Function and Example

If CPU address is between the Low and the High decode addresses, then the access is passed to the Device Unit for sub-decode.



Example: Set up an address decode region that starts at 0x4000.0000 and is 64Mbytes in length (0x4000.0000 to 0x43FF.FFFF):

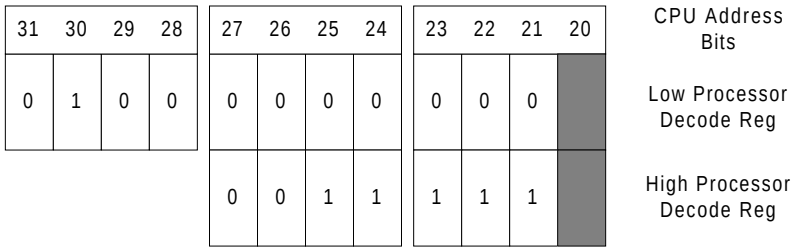
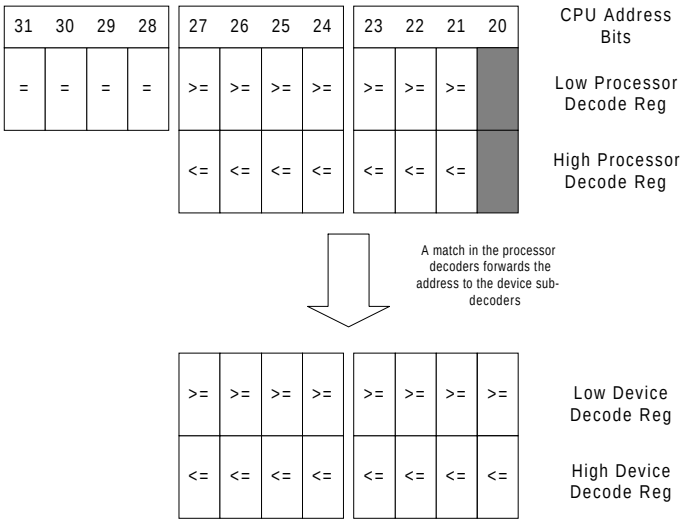
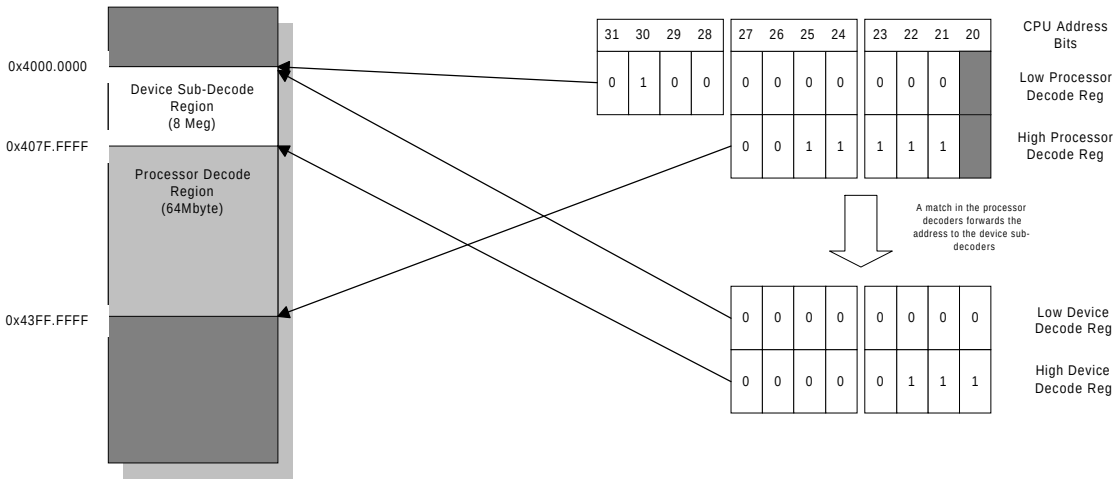


Figure 4: Device Sub-Decode Function and Example



Example: Using the previous CPU decode example (0x4000.0000 to 0x43FF.FFFF), place a device sub-decode in the first 8 Meg:

0x4000.0000 to 0x407F.FFFF

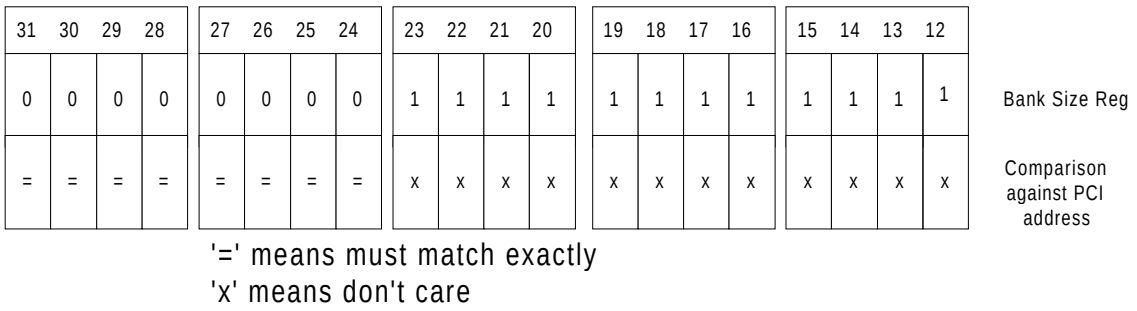


3.2 PCI Side Decoding Process

Decoding on the PCI side starts with the PCI address being compared with the values in the various Base Address Registers. For example, the SCS[1:0]* Base Address Register sets the PCI base address range in which the SCS0* and SCS1* signals are active (i.e., where DRAM banks 0 and 1 are located in PCI space.)

The bank size registers for each Base Address Register sets the size of the "window" in PCI space for each Base Address Register. The bank size sets which address bits are significant for the comparison between the active PCI address and the values in the Base Address Registers (see Figure 5).

Figure 5: Bank Size Register Function Example (16Meg Decode)



The comparison works as follows:

1. Bits 31:N of the PCI address are compared against bits 31:N in the various Base Address Registers (BAR). These values much match exactly. The value of 'N' is set by the least significant bit with a 0 in the Bank Size Registers (i.e., 'N' would be equal to 24 in the example shown in Figure 4, above.)
2. If all of the above is true, the resource group is selected and a subdecode is performed to determine the specific resource.

Once a resource group has been decoded by a BAR, it must be subdecoded to determine which physical device should be accessed within that group. This decoding is controlled by the Device Low and High decode registers.

NOTE: These registers are the same ones used for CPU-side decoding. This means that the PCI and CPU memory maps are coupled at the device decoders. Address bits 27:20 (the bits compared by the Device decoders) for any given device overlap in both the PCI and CPU maps.

The sub-decoding comparison works as follows:

1. Bits 27:20 of the PCI address are compared against bits the relevant device Low decode registers. The value of the PCI address bits must be greater than or equal to the Low decode value. This sets the lower boundary for the sub-decode region.
2. Next, bits 27:20 of the PCI address are compared against the relevant device High decode registers. The value of the PCI address bits must be less than or equal to this value. This sets the upper bound for the sub-decode region.
3. If all of the above are true, the specific device is selected and an access to that device is performed.

NOTE: The coupling of the CPU, PCI, and device memory maps requires special attention for designers of PC Plug and Play adapters.

3.3 Disabling the Device Decoders

Disable the CPU interface address decoders by setting the LOW decoder value higher than the HIGH decoder-value.

Disable the device sub-decoder by setting the LOW decoder value higher than the HIGH decoder value.

Disable the PCI address decoders by setting the BAR's corresponding bit in Base Address registers' Enable to 1.

3.4 DMA Unit Address Decoding

The DMA controller uses the address mapping of the CPU interface when accessing the device/DRAM bus. The DMA Unit can access the PCI bus independent of the CPU-side PCI bridge decoders on the GT-64130 devices only, see Section 3.9 "DMA PCI Override" on page 41.

3.5 Address Space Decoding Errors

When the CPU tries to access an address from the A/DL that is not supported, the GT-64130 latches the address into the Bus Error register (offset 0x70) and set CPUOut bit in the interrupt Cause register, interrupt is asserted if not masked.

This feature is useful during software debugging operations. Errant code can cause fetches from unsupported addresses.

When address matches one of CPU interface address spaces but misses the associated subdecoders, the GT-64130 latches the address into the Address Decode Error register (offset 0x470) and set MemOut bit in the interrupt Cause register, interrupt is asserted if not masked.

When a PCI access hits in a Base Address Register then misses in the associated subdecoders, the result is random data returned on a read. Write data is discarded. The MemOut bit in the interrupt Cause register is also set. Accesses that miss all of the GT-64130 BARs result in no response at all from the GT-64130.

When a DMA accesses an unmapped address, DMAOut bit in the interrupt Cause register is set.

NOTE: Address space decoders must never be programmed to overlap. Unpredictable behavior will result.

3.6 Default Memory Map

Table 6 shows the valid default CPU memory map following RESET.

Table 7 and Table 8 shows the default PCI map and BAR sizing information.

Table 6: CPU and Device Decoder Default Address Mapping

CPU Decode Range and Size	Resource Group	Device Decode Range and Size	Device Selected
0x0 to 0x00FF.FFFF 16 Megabytes	SCS[1:0]*	0x0 to 0x007F.FFFF 8 Megabytes	SCS0*
		0x0080.0000 to 0x00FF.FFFF 8 Megabytes	SCS1*
0x0100.0000 to 0x01FF.FFFF 16 Megabytes	SCS[3:2]*	0x0100.0000 to 0x017F.FFFF 8 Megabytes	SCS2*
		0x0180.0000 to 0x01FF.FFFF 8 Megabytes	SCS3*
0x1400.0000 to 0x1400.0FFF 4 Kbytes	Internal Registers	No subdecode Access bridged directly to the GT-64130 internal registers.	Internal Registers
0x1000.0000 to 0x11FF.FFFF 32 Megabytes	PCI0 I/0	No subdecode Access bridged directly to PCI I/O space.	PCI0
0x1200.0000 to 0x13FF.FFFF 32 Megabytes	PCI0 Mem0	No subdecode Access bridged directly to PCI memory space	PCI0
0x1C00.0000 to 0x1E1F.FFFF ~32 Megabytes[1]	CS[2:0]*	0x1C00.0000 to 0x1C7F.FFFF 8 Megabytes	CS0*
		0x1C80.0000 to 0x1CFF.FFFF 8 Megabytes	CS1*
		0x1D00.0000 to 0x1DFF.FFFF 16 Megabytes	CS2*
0xFF00.0000 to 0xFFFF.FFFF 16 Megabytes	CS[3]* and BootCS*	0xFF00.0000 to 0xFFBF.FFFF 12 Megabyte	CS3*
		0xFFC0.0000 to 0xFFFF.FFFF 4 Megabytes	BootCS*

Table 6: CPU and Device Decoder Default Address Mapping (Continued)

CPU Decode Range and Size	Resource Group	Device Decode Range and Size	Device Selected
0xF200.0000 to 0xF3FF.FFFF 32 Megabytes	PCI0 Mem1	No subdecode Access bridged directly to PCI memory space.	PCI0
0x2000.0000 to 0x21FF.FFFF 32 Megabytes	PCI1 I/0	No subdecode Access bridged directly to PCI I/O space	PCI1
0x2200.0000 to 0x23FF.FFFF 32 Megabytes	PCI1 Mem0	No subdecode Access bridged directly to PCI memory space.	PCI1
0x2400.0000 to 0x25FF.FFFF 32 Megabytes	PCI1 Mem1	No subdecode Access bridged directly to PCI memory space.	PCI1

1. By default, OX1E00.0000 to OX1E1F.FFFF is allocated to CS[2:0] but is not accessible.The device decoders are not pro-grammed to respond to a hit from this region.

Table 7: PCI Function 0 and Device Decoder Default Address Mapping

PCI Function 0 Decode Range and Size	Resource Group	Device Decode Range and Size	Device Selected
0x0 to 0x00FF.FFFF 16 Megabytes in Memory Space	SCS[1:0]*	0x0 to 0x007F.FFFF 8 Megabytes	SCS0*
		0x0080.0000 to 0x00FF.FFFF 8 Megabytes	SCS1*
0x0100.0000 to 0x01FF.FFFF 16 Megabytes in Memory Space	SCS[3:2]*	0x0100.0000 to 0x017F.FFFF 8 Megabytes	SCS2*
		0x0180.0000 to 0x01FF.FFFF 8 Megabytes	SCS3*
0x1400.0000 to 0x1400.0FFF 4 Kbytes in Memory Space	Internal Registers	No subdecode	Internal Registers
0x1400.0000 to 0x1400.0FFF 4 Kbytes in I/O Space	Internal Registers	No subdecode	Internal Registers

Table 7: PCI Function 0 and Device Decoder Default Address Mapping (Continued)

PCI Function 0 Decode Range and Size	Resource Group	Device Decode Range and Size	Device Selected
0x1C00.0000 to 0x1DFF.FFFF 32 Megabytes in Memory Space	CS[2:0]*	0x1C00.0000 to 0x1C7F.FFFF 8 Megabytes	CS0*
		0x1C80.0000 to 0x1CFF.FFFF 8 Megabytes	CS1*
		0x1D00.0000 to 0x1DFF.FFFF 16 Megabytes	CS2*
0xFF00.0000 to 0xFFFF.FFFF 16 Megabytes in Memory Space	CS[3]* and BootCS*	0xFF00.0000 to 0xFFBF.FFFF 12 Megabyte	CS3*
		0xFFC0.0000 to 0xFFFF.FFFF 4 Megabytes	BootCS*
0xFF00.000 to 0xFFFF.FFFF 16 Megabytes (uses CS[3]* and BootCS* size register)	PCI Expansion ROM	No subdecode. This decoder is used only during PC BIOS initialization.	CS3*

Table 8: PCI Function 1 (Byte Order Swap) and Device Decoder Default Address Mapping

PCI Function 0 Decode Range and Size	Resource Group	Device Decode Range and Size	Device Selected
0x0 to 0x00FF.FFFF 16 Megabytes in Memory Space	SCS[1:0]*	0x0 to 0x007F.FFFF 8 Megabytes	SCS0*
		0x0080.0000 to 0x00FF.FFFF 8 Megabytes	SCS1*
0x0100.0000 to 0x01FF.FFFF 16 Megabytes in Memory Space	SCS[3:2]*	0x0100.0000 to 0x017F.FFFF 8 Megabytes	SCS2*
		0x0180.0000 to 0x01FF.FFFF 8 Megabytes	SCS3*
0xFF00.0000 to 0xFFFF.FFFF 16 Megabytes in Memory Space	CS[3]* and BootCS*	0xFF00.0000 to 0xFFBF.FFFF 12 Megabyte	CS3*
		0xFFC0.0000 to 0xFFFF.FFFF 4 Megabytes	BootCS*

NOTE: The default BootCS address space matches PowerPC boot address (0xfff00100). When working with PowerQUICC and the boot device resides on the GT-64130's BootCS, the PowerQUICC needs to be configured at reset to boot address of 0xfff00100. IP bit of PowerQUICC reset configuration determines whether the boot address is 0xfff00100 or 0x00000100.

3.7 CPU and PCI Address Remapping

The GT-64130 supports address remapping from CPU side and from PCI side.

3.7.1 CPU Address Remapping

The CPU can address the following resources:

- the SDRAM banks (SCS[1:0]*, SCS[3:2]*)
- the devices (CS[2:0]*, CS[3]* & BootCS*)
- PCI_0 IO
- PCI_0 Memory0/1
- PCI_1 IO
- PCI_1 Memory0/1

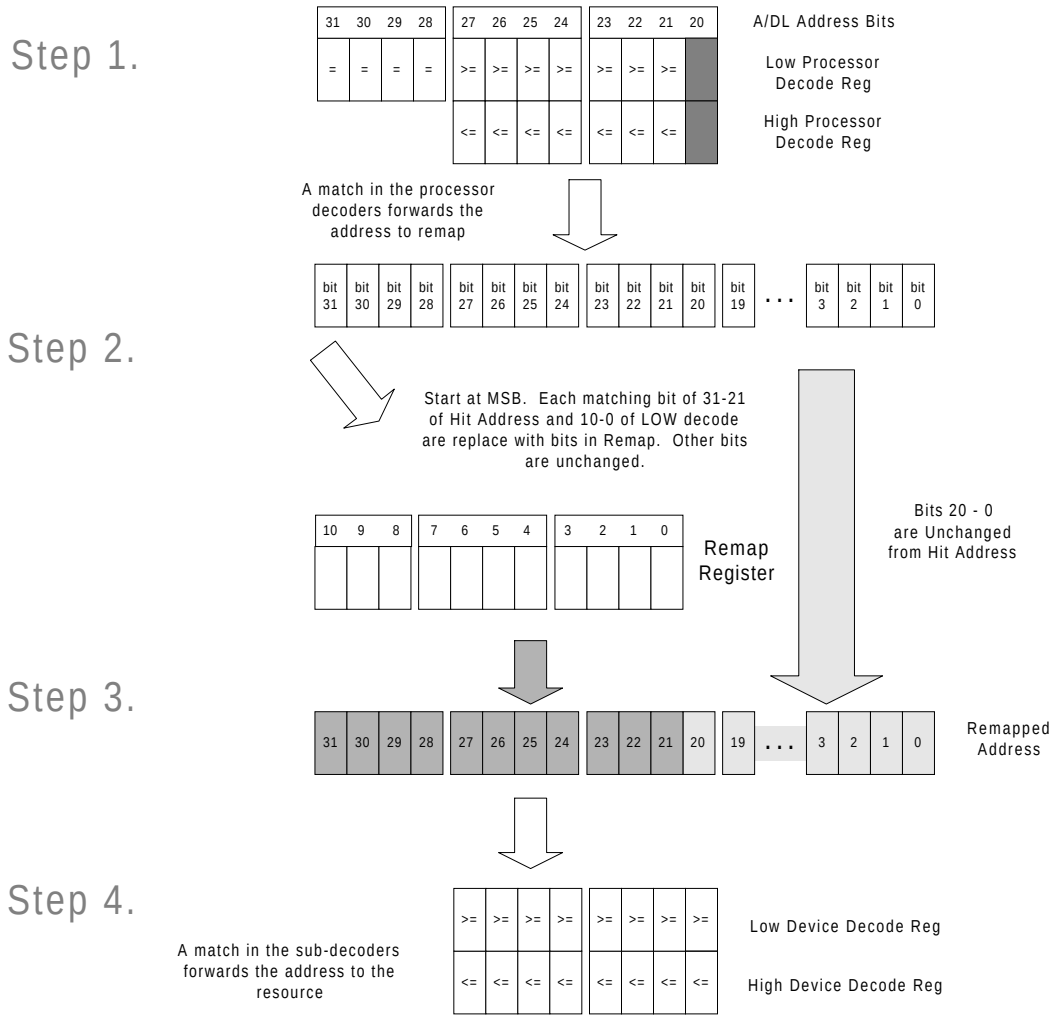
PCI_1 IO and PCI_1 Memory0/1 are only addressable if the device is configured for both PCI_0 and PCI_1 on RESET. Each resource has a Remap Register associated with it and are listed in the Register Section as part of Processor Address Space. See Section 19.4 "CPU Address Decode" on page 171. The offsets for these registers are 0x0d0–0x100. Each MAP register is 11 bits wide.

An address presented on the A/DL bus by the CPU is decoded with the following steps:

1. CPU address bits [31:21] are checked for a hit in the CPU decoders.
2. Assuming there is a hit in the CPU decoders, bits 20:0 are left unchanged. Bits[31:21] are remapped as follows:
   i) Going from the MSB to LSB of the HIT address bits [31:21], any bit found matching to its respective bit in the LOW decode register's bits [10:0] will cause the according bit in the remap register to REPLACE the original address bit.
   ii) Upon first mismatch, all remaining LSBs of address bits[31:21] are unchanged.
3. Address bits [27:20] of the *remapped address* are checked to be a hit in the Device decoders.
4. Assuming there is a hit in the Device decoders, the HIT address will be transferred to the resource.

See Figure 6 outlining this address remapping procedure.

Figure 6: CPU Address Remapping To Resources



3.7.2 Writing to Decode Registers

When a LOW decode register is written to, the same value is written to the associated remap register, 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 LOW Decode register bits [10:0]. 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.

3.7.3 PCI Address Remapping

The PCI slave interface can remap addresses of PCI transactions to memory using PCI remap registers.

There are seven registers for PCI_0 and seven registers for PCI_,1 if implemented. These registers correspond to the following PCI Base Address Registers:

- SCS[1:0]*
- SCS[3:2]*
- CS[2:0]*
- CS[3]* & BootCS*
- Swapped SCS[1:0]*
- Swapped SCS[3:2]*
- Swapped CS[3]* & BootCS*

These registers are listed in the Register Section as part of PCI Internal Registers. See Section 19.15 ”PCI Internal” on page 200. The offsets for these registers are 0xc48 - 0xc64. Each MAP register is 32-bits wide. Each MAP register is 32-bits wide, where bits [12:0] are Read Only.

When an address is presented on the PAD lines, the address decoder in the PCI slave compares the PCI address to its base/size registers. If there is a hit in one of the seven Base Address registers listed above, the address will undergo remapping according to the right remap register in the non-masked address bits (by size register). An example of this is summarized in Table 9.

Table 9: PCI Address Remapping Example

PCI address	0x1D98.7654
SCS[1:0]* BAR	0x1F00.0000
SCS[1:0]* Size	0x03FF.FFFF
SCS[1:0]* Remap Register	0x3F00.0000
Remapped PCI Address Presented to SDRAM	0x3D98.7654

Notice that 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.

3.7.4 Writing to 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 equals 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.

3.8 CPU PCI Override

In default, the CPU interface supports 512Mbyte PCI memory address space (256Mbyte on PCI_0 Mem0, 256Mbyte on PCI_0 Mem1). If configured to both PCI_0 and PCI_1, it supports 512Mbyte also on PCI_1. The CPU PCI override feature enables larger PCI memory address space.

The CPU configuration register includes four PCI override bits. These four bits comprise two bits for PCI_0 and two bits for PCI_1. Each bit pair controls whether PCI window is 2Gbyte, 1Gbyte, the complement of all other address windows, or the default.

When PCI override bits are set to 01, if bits[31:30] of CPU address matches bits [10:9] of PCI Mem0 Low decode address register, the transaction is directed to PCI Mem0. This effectively sets a 1Gbyte window to PCI. If Bits[31:30] do not match bits [10:9] of PCI Mem0 Low decode address register, the address is compared against all other address decode registers.

When PCI override bits are set to 10, if bit[31] of CPU address matches bit [10] of PCI Mem0 Low decode address register, the transaction is directed to PCI Mem0. This effectively sets a 2Gbyte window to PCI. If bit[31] does not match bit [10] of PCI Mem0 Low decode address register, the address is compared against all other address decode registers.

When PCI override bits are set to 11, the address is first compared against all address decode registers. If there is no match in any of them, the transaction is directed to PCI Mem0. This effectively sets a PCI window size of 4Gbyte *minus* the sizes of all devices. However, if this mode is chosen, there is no indication for bad CPU address (CPUOut interrupt will never be asserted). Setting override bits to 11 is not allowed in Multi-GT configuration. See Section 4.13 ”Multiple GT-64130 Support” on page 52 for more details.

NOTE: When PCI override is enabled, there is no address remapping from CPU to PCI.

3.9 DMA PCI Override

In default, the DMA controller uses CPU interface address decoding as explained before. However, the DMA controller supports direct access to PCI bus, bypassing this address decoding.

In each of the four DMA channel control registers, there are six PCI override bits - 2-bits per source address, 2-bits per destination address, and 2- bits per next record address. Each bit pair controls whether the address should be directed to PCI_0 memory space, to PCI_1 memory space, or run through the CPU address decoding. See Section 19.12 ”DMA Channel Control” on page 193 for more details.

4. CPU INTERFACE DESCRIPTION

The GT-64130 CPU bus interface allows the CPU to gain access to the GT-64130's internal registers, PCI interface and the memory/device bus. The A/D bus supports accesses from one to 32 bytes in length.

The A/D bus on the GT-64130 is a slave-only interface; the GT-64130 will never master the A/D bus.

4.1 CPU Interface Signals

The CPU interface incorporates the following signals:

Table 10: CPU Interface Signals

Signals	Description
A/D[0-31]	Multiplexed Address/Data-Low.
DH[0-31]	Data-High bus.
DP[0-7]	An 8-bit bus containing odd parity for the A/D bus. DP is only valid on data read cycles.
TS*	Indicates that the CPU is driving valid address and attributes.
TBST*	Transaction attribute. TBST* indicates burst transfer.
TSIZ[0-2]	Transaction attribute. TSIZ indicates number of bytes for transfer.
TT[1]	Transaction attribute. TT[1] indicates read/write.
TT[3]	Transaction attribute. TT[3] indicates address only transaction.
AACK*	Indicates that the GT-64130 sampled address and attributes and CPU may float address bus.
TA*	Indicates that the GT-64130 is driving valid data on the bus on a read transaction or that the GT-64130 sampled data driven by CPU on a write transaction.
BG*	Indicates that the CPU may drive a new transaction on the bus.
DBG*	Indicates that the CPU may drive data on the bus if a write transaction.
QsAEn/ QsAEn2	The GT-64130 quick switch control signals.
L2BR*	L2 cache request for bus mastership (for cache line write-back).
L2BG*	Indicates that L2 may drive a new transaction (write-back) on the bus.

Table 10: CPU Interface Signals (Continued)

Signals	Description
L2DBG*	Indicates that L2 may drive data on the bus.
L2HIT*	Indicates a hit in L2 cache tag RAM. The GT-64130 ignores the transaction if L2HIT* is asserted.
ARTRY*	Address retry. In multi-CPU systems, indicates that a transaction initiated by one CPU, hit a modified line in another CPU L1 cache. The first CPU needs to restart the transaction after the 2nd CPU writes back the cache line to memory.The GT-64130 ignores the transaction if ARTRY* is asserted.
Interrupt*	An “OR” of all the internal interrupt sources on the GT-64130.

4.2 PowerPC Address/Data Buses Multiplex

The PowerPC bus specification defines separate address and data buses.

A single PowerPC CPU can support an address pipeline depth of two transactions (three in the case of MPC604e). In instances of address pipelining, the CPU starts a new address tenure while a previous data tenure is in progress.

The GT-64130 CPU interface is based on a 64-bit multiplexed address/data bus. The GT-64130 supports two ways of implementing this multiplex. The first configuration is implemented with external Quickswitches on CPU A[0-31] and DL[0-31] where the GT-64130 controls which QuickSwitch is open (address or data). This configuration can be implemented with external QuickSwitch or any other 3-state logic. In this configuration, the CPU may strart a new address tenure while a previous data tenure is still in progress. However, the GT-64130 samples the new address only after the data tenure is completed.

In the second method there is no need for external QuickSwitch. CPU’s A[0-31] and DL[0-31] are shorted together and the GT-64130 controls CPU’s bus arbitration with BG*, DBG*, and AACK* signals. The GT-64130 is configured to one of the methods by DMAReq[3]* and Dadr[9] pins sampled at reset. See Section 11. “Reset Configuration” on page 139 for more information.

The GT-64130 supports L2 cache in the system. For the case of write-back L2 cache, it has three more arbitration signals:

- L2BR*
- L2BG*
- L2DBG*

These arbitration signals enables the GT-64130 to control L2 bus arbitration for write-back transactions.

The GT-64130 also supports a multi-CPU system. In such a system, bus arbitration is controlled by an external arbiter. If a non QuickSwitch implementation is required, this external arbiter must be designed to support the short between A[0-31] and DL[0-31].

For a detailed description of bus multiplex, see Section 4.4 “QuickSwitch Configuration” on page 46.

4.3 A/DL,DH,DP Buses and Transaction Attributes

The GT-64130 CPU bus protocol is completely compatible with the 64-bit 60X bus protocol used by the MPC603e, MPC604e, and MPC740/750 processors.

On Transfer Start, the CPU drives address on A[0-31] and transaction attributes on TBST*, TSIZ, and TT. It keeps this data until the GT-64130 asserts AACK*. It floats these signals to the next cycle. In case of a write transaction, the CPU drives data bus with valid data. In case of a read transaction, the CPU samples data driven on the bus with each TA* asserted by the GT-64130. With each valid read data, the GT-64130 drives odd parity on DP bus.

PowerPC 60X bus spec defines Transfer Type with TT[0-4]. However, most of the information carried on these signals is irrelevant for the GT-64130 work. The GT-64130 decodes only TT[1] and TT[3]. TT[1] is identifying read/write transaction (read = 1). TT[3] is used for identifying address-only transaction (address only = 0).

TBST* and TSIZ defines the number of bytes for transfer as shown in Table 11.

Table 11: Transfer Size Summary

TBST*	TSIZ[0-2]	Transfer Size
0	X	32 bytes burst
1	000	8 bytes
1	001	1 byte
1	010	2 bytes
1	011	3 bytes
1	100	4 bytes
1	101	5 bytes
1	110	6 bytes
1	111	7 bytes

On burst transactions (TBST* asserted), the CPU drives double-word aligned address (A[29-31] = 0).

On burst write, the address is cache-line aligned (A[27-31] = 0).

On burst read, burst order is determined by the first word address as shown in Table 12.

Table 12: Burst Read Ordering

Data Transfer	Start Address			
	A[27-28]=00	A[27-28]=01	A[27-28]=10	A[27-28]=11
1st data beat	DW0	DW1	DW2	DW3
2nd data beat	DW1	DW2	DW3	DW0
3rd data beat	DW2	DW3	DW0	DW1
4th data beat	DW3	DW0	DW1	DW2

On non-burst transactions, the CPU might access 1, 2, 3, 4, or 8 bytes.

Table 13: Aligned Data Transfers

Transfer Size	TSIZ[0-2]	A[29-31]	Data Bus Byte Lanes							
			0	1	2	3	4	5	6	7
Byte	001	000	A	-	-	-	-	-	-	-
	001	001	-	A	-	-	-	-	-	-
	001	010	-	-	A	-	-	-	-	-
	001	011	-	-	-	A	-	-	-	-
	001	100	-	-	-	-	A	-	-	-
	001	101	-	-	-	-	-	A	-	-
	001	110	-	-	-	-	-	-	A	-
	001	111	-	-	-	-	-	-	-	A
Half word	010	000	A	A	-	-	-	-	-	-
	010	010	-	-	A	A	-	-	-	-
	010	100	-	-	-	-	A	A	-	-
	010	110	-	-	-	-	-	-	A	A
Word	100	000	A	A	A	A	-	-	-	-
	100	100	-	-	-	-	A	A	A	A
Double word	000	000	A	A	A	A	A	A	A	A

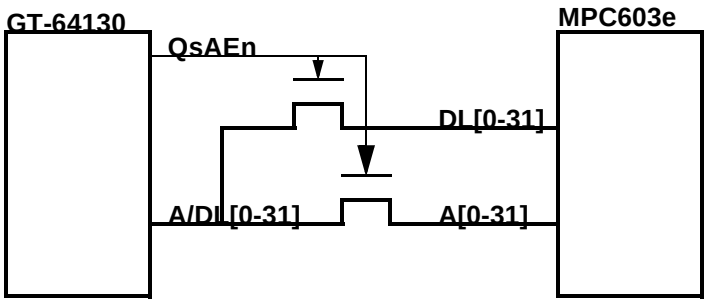
Table 14: Misaligned Data Transfers

Transfer Size	TSIZ[0-2]	A[29-31]	Data Bus Byte Lanes							
			0	1	2	3	4	5	6	7
Two bytes	010	001	-	A	A	-	-	-	-	-
	010	101	-	-	-	-	-	A	A	-
Three bytes	011	000	A	A	A	-	-	-	-	-
	011	001	-	A	A	A	-	-	-	-
	011	100	-	-	-	-	A	A	A	-
	011	101	-	-	-	-	-	A	A	A

4.4 QuickSwitch Configuration

QuickSwitch configuration requires an external 64->32 quick switch mux in order to select between CPU address and data as shown in Figure 7.

Figure 7: A/DL[0-31] Mux Using QuickSwitch



On address phase, the GT-64130 asserts QsAEn, deasserts QsAEn until the end of data phase, and switches back to address select.

For load balancing, the GT-64130 also drives a QsAEn2 signal which is a duplication of QsAEn.

Figure 8 shows two consecutive write transactions. The first one is a burst write of 32-bytes. The second is a single beat write. The CPU starts the transaction with TS* assertion for one cycle. During the same cycle, it asserts ABB* (not used by GT-64130) and drives address and attributes. It keeps driving address until the GT-64130 asserts AACK*. The next cycle it floats address bus. On the cycle after TS*, the CPU starts driving data bus. The GT-64130 signals that it received the data by asserting TA*. In case of single data transfer, the CPU will float data bus the next cycle after TA* assertion. On burst write, CPU will drive the data bus until the fourth TA*, with each TA* assertion it will drive new data on the bus.

The GT-64130 uses AACK* to control address pipeline. Since the GT-64130 has a multiplexed address/data bus, it can't accept a new transaction address before the completion of previous transaction data tenures. Although the CPU is allowed to drive a new transaction address before completion of previous transactions, the GT-64130 forces the CPU to keep driving the second transaction address until the first transaction's completion, by not asserting AACK*. The GT-64130 also holds AACK* assertion if it's internal address buffers are full and it cannot accept a new transaction address.

In general, the GT-64130 accepts write data every cycle. However, if it's internal write data buffer is full, it holds data receiving by deasserting TA*.

Figure 8: CPU Write in QuickSwitch Configuration

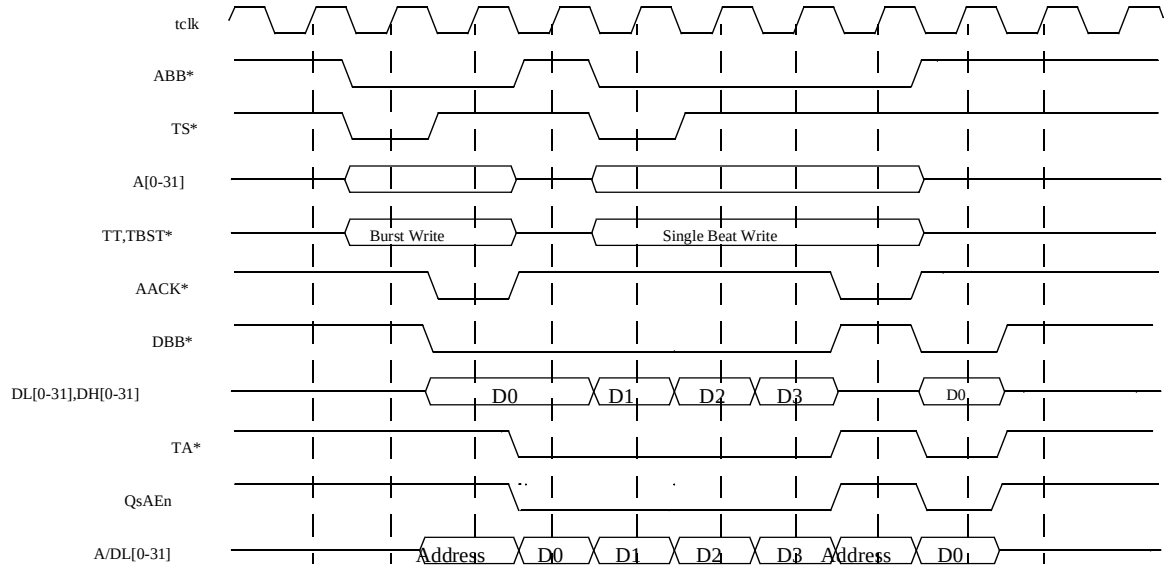
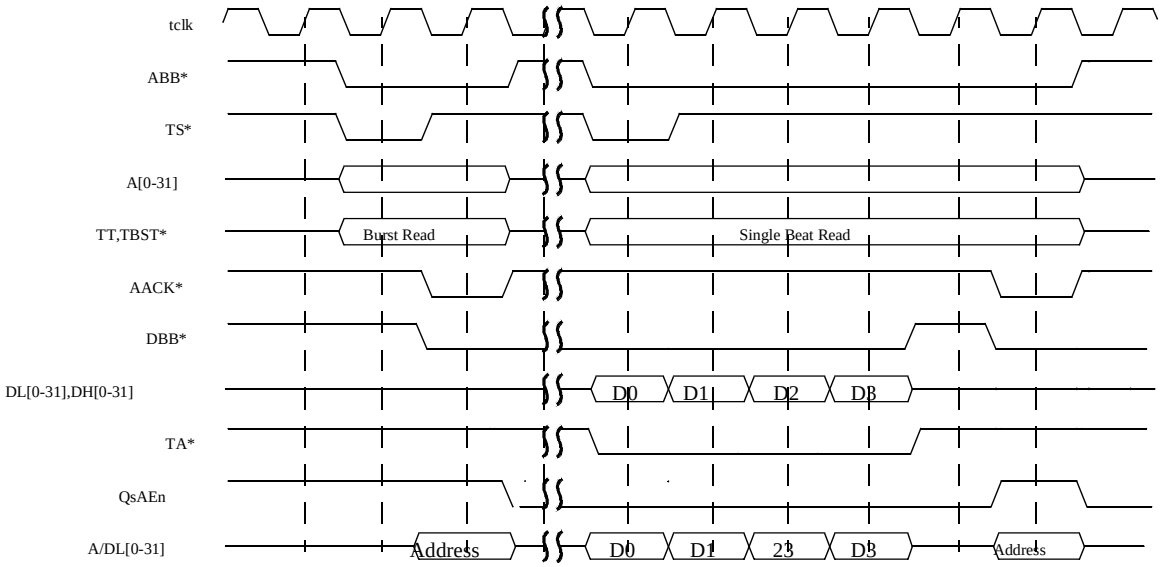


Figure 9 show two consecutive read transactions. The first one is burst read of 32 bytes. The second is a single beat read. Address phase is the same as in the case of write transaction. The next cycle after TS* assertion, the CPU asserts DBB* to indicate it is waiting for read data. As soon as the GT-64130 receives data from the requested source (SDRAM,PCI ...), it signals that it drives valid data on the bus by asserting TA*. GT-64130 will drive with each valid data also odd data parity on DP[0-7] bus.

On read transaction, GT-64130 uses AACK* to insert a dead cycle on A/DL bus between end of read transaction and start of write transaction.

Figure 9: CPU Read in QuickSwitch Configuration



4.5 Glueless Configuration

In a glueless configuration, no external hardware is required to select between CPU address and data. CPU A[0-31] and DL[0-31] are shorted together, and the GT-64130 prevents bus contention by controlling buses driving through BG*, DBG*, and AACK* signals as shown in Figure 10.

Figure 10: A/DL[0-31] Mux Without External Hardware

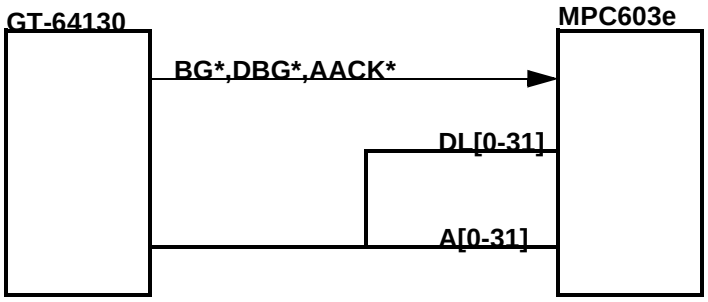


Figure 11 shows two consecutive write transactions. These transactions are a burst write transfer of 32-bytes followed by a single data transfer.

In the burst write transfer, the GT-64130 asserts BG* and waits for transaction start. The CPU starts the transaction by asserting TS* for one cycle. During the same cycle, it asserts ABB* (not used by the GT-64130) and

drives address and attributes. The GT-64130 deasserts BG* the next cycle and keeps it deasserted until the transaction end. The CPU keeps driving address until the GT-64130 asserts AACK*.

During the next cycle, the GT-64130 asserts DBG* to enable the CPU driving the data bus. The CPU start driving data bus in the next cycle after DBG*. The GT-64130 signals that it received the data by asserting TA*. In case of single data transfer, the CPU will float data bus the next cycle after TA* assertion. On burst write, the CPU will drive the data bus until the 4th TA*. With each TA* assertion, it will drive new data on the bus. The next cycle after last TA*, the GT-64130 asserts BG* to enable a new CPU transaction start.

In this configuration, the GT-64130 controls A/DL bus mux using BG*, DBG*, and AACK*. This configuration requires insertion of a dead cycle between every address/data switch. This is a performance penalty that must be considered when choosing between the QuickSwitch configuration and the glueless configuration.

Figure 11: CPU Write in Glueless Configuration

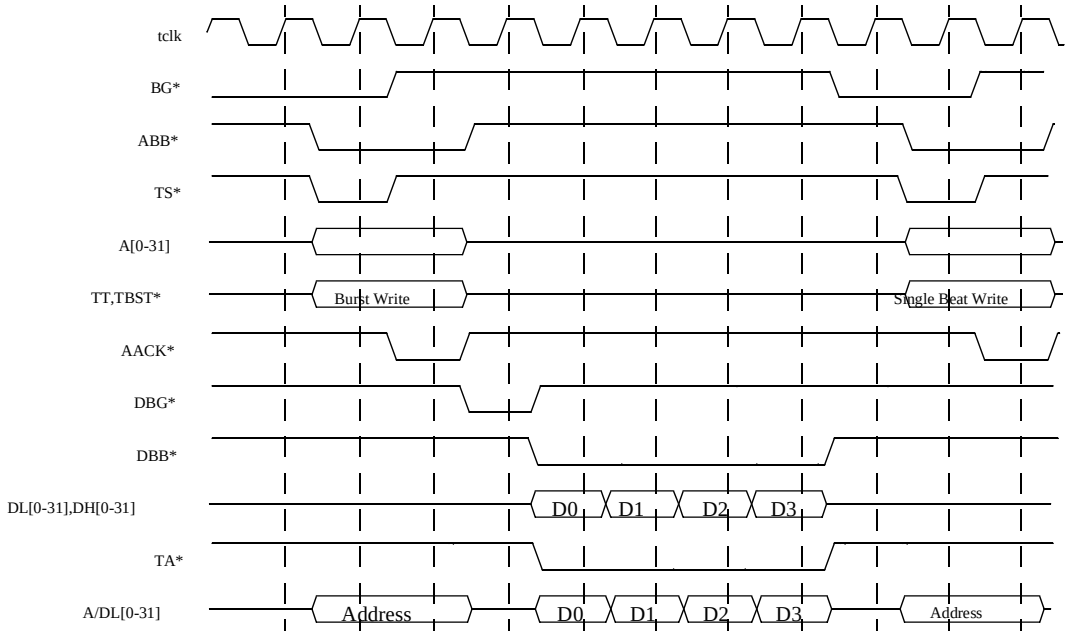
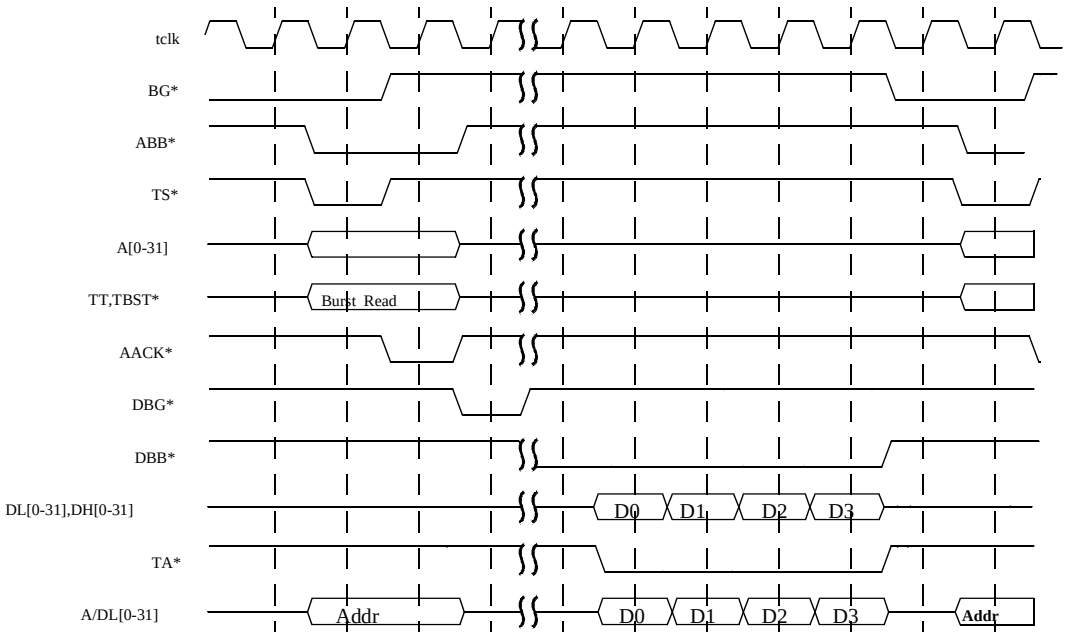


Figure 12 shows two consecutive read transactions. The first transaction is a single data transfer and it is followed by a burst read of 32 bytes.

The address phase is the same as in a write transaction.

During the cycle after the GT-64130 assert DBG*, the CPU asserts DBB* to indicate it is waiting for read data. As soon as the GT-64130 receives data from the requested source (SDRAM,PCI ...), it signals that it drives valid data on the bus by asserting TA*. The GT-64130 will drive with each valid data also odd data parity on DP[0-7] bus. During the next cycle after the last TA*, the GT-64130 asserts BG* to enable a new CPU transaction start.

Figure 12: CPU Read in Glueless Configuration



4.6 CPU Interface Write Buffer

The GT-64130's CPU interface includes a write posting queue that absorbs CPU writes at zero wait-states. The write posting queue has four address entries and eight 64-bit data entries.

The GT-64130 signals if there is "room" in the CPU write posting queue by asserting AACK* or TA*. If all four address entries are full, the GT-64130 will not assert AACK* in response to a new transaction start. If all data entries are full, the GT-64130 will not assert TA*.

4.7 CPU Interface Read Buffer

The GT-64130's CPU interface includes a read buffer that absorbs data received from memory or PCI. This buffer is used for the unpack process in case of 32-bit bus CPU (603e in 32-bit mode or PowerQUICC) and for burst read order alignment in case of 64-bit bus.

Since the GT-64130's internal data path is 64-bit wide, the CPU interface needs to unpack 64-bit received data before driving it on 32-bit wide data bus. This buffer is also used for driving burst read order properly (linear wrap around for 32-byte cache line in case of 603e, and 16-byte cache line in case of PowerQUICC).

When working in 64-bit bus configuration, only burst read data passes through the read buffer. The read buffer may be bypassed by setting BlockRdBuffer bit in CPU Configuration register to 1.

In order to get the minimum latency on a CPU bus read from 64-bit SDRAM or a device, this bit should be set to 1. However, in this case, SDRAM burst length should be limited to 4 which limits PCI and DMA bursts to SDRAM to 4x64-bit. If BlockRdBuffer bit is cleared, data will pass through the read buffer. In this case, if transaction address is aligned to a cache line (32 byte), latency increases by one clock in comparison to read buffer bypass. However, in this case, SDRAM may be configured to burst of 8, gaining higher DMA and PCI performance.

4.8 CPU Interface Endianess

The GT-64130 provides the capability to swap the endianess data transferred to or from the internal registers and to or from the PCI interface. Data written to or from the memory controller is NEVER swapped.

The CPU endianess is programmed on RESET by sampling the Interrupt* pin. See Section 11. “Reset Configuration” on page 139 for more information. The setting of this pin also programs the Endianess bit in the CPU Configuration register. When accessing the internal registers, the endianess of the data will be determined by the setting of the Interrupt* pin. If set to BIG endian, data is swapped.

The setting of MByteSwap bit and MWordSwap bit in the PCI Internal Command Register determines how data transactions from the CPU to and from PCI are handled. Both MByteSwap and CPU Endianess bits are set to the same value as the pin strapping of the Interrupt* (resulting PCI interface working in little endian mode). These bits can be re-programmed after RESET. For more information, see Section 14. “Big and Little Endian” on page 146.

4.9 Parity Support

The GT-64130 drives ODD parity on DP[0-7] signal with each data transfer of a read transaction. The GT-64130 always drives correct ODD parity, even if the parity received from PCI or ECC received from memory were wrong.

The GT-64130 ignores parity driven by the CPU on write transactions. It generates parity to PCI and ECC to SDRAM or Devices internally.

4.10 Burst Order

The GT-64130 supports linear wrap-around burst order used by PowerPC CPUs. With burst transactions, CPUs can access the PCI, SDRAM, and devices interface.

4.11 Address Only Transaction

The PowerPC bus spec defines address-only transactions. In such a transaction, there is only an address tenure. There is no data tenure.

Address-only transactions are generally directed to caches in the system. However, if for some reason such a transaction is targeted to the GT-64130, it completes the transaction properly by asserting AACK* but the transaction itself is ignored. In other words, the GT-64130 does not generate any internal event (no register is changed, no memory or PCI transaction is generated).

4.12 L2 Cache Support

In the presence of L2 cache in the system, the target of any CPU transaction can be the GT-64130 or L2 cache. Since only one target should drive AACK* and TA* signals, they are now in a sustained tri-state output requiring a 4.7 KOhm pull-up resistor. AACK*/TA* outputs of the GT-64130 are connected to AACK*/TA* of L2 cache controller, respectively. After the GT-64130 asserts AACK* low for one cycle, it will drive it High for another cycle and then tri-state it. The same is true for driving last TA*.

The GT-64130 samples L2HIT* signal. The GT-64130 L2HIT* sample window is programmable through bit[15] of the CPU interface configuration register. It is either the next clock after TS* assertion or two clocks.

In case of hit in L2 cache, the GT-64130 ignores the transaction. Due to L2HIT* sample, the GT-64130 delays AACK* assertion. The earliest AACK* assertion is two cycles after TS* assertion.

The GT-64130 supports both write-back and write-through cache. With write-back cache, the write-back cache becomes a bus master when it needs to write-back a cache line to main memory. The GT-64130 supports bus arbitration for single CPU plus L2 write-back cache. In a glueless configuration, the GT-64130 controls L2BG* and L2DBG* signals to enable the short between A[0-31] and DL[0-31] busses.

4.13 Multiple GT-64130 Support

Up to four GT-64130 devices can be connected to the CPU System Interface without the need for any glue logic. This capability increases the address space and adds significant flexibility for system design.

Multiple GT-64130s are enabled by sampling the value of DAdr[10] on RESET. See Section 11. “Reset Configuration” on page 139 for more information. If this pin is sampled HIGH, multiple GT-64130s are enabled. This pin must be tied LOW if there is only one GT-64130. The value of DAdr[10] sampled on RESET will be bit 18, MultiGT, of the CPU Configuration Register. This bit is programmable after RESET.

If multiple GT-64130s are enabled, the values sampled on Ready* and CSTiming* determine the ID of the particular GT-64130 as shown in Table 15. This sampled values also program the MultiGTAct bits [1:0] in the Multi-GTID Register, 0x120. These bits are programmable after RESET.

Table 15: Pin Strapping the GT-64130 ID

Pin	Configuration Function
Ready*, CSTiming	Multi-GT-64130 Address ID
00-	GT-64130 responds to A/DL[5-6]='00'
01-	GT-64130 responds to A/DL[5-6]='01'
10-	GT-64130 responds to A/DL[5-6]='10'
11-	GT-64130 responds to A/DL[5-6]='11'
	NOTE: NOTE: Boot GT-64130 should be programmed to '11'

4.13.1 Hardware Connections

The boot GT-64130 monitors the CPU transaction regardless of which of the four GT-64130 devices is the target. Bus arbitration and QuickSwitch control signals are driven only by the Boot GT-64130 device. A system must use only the boot GT-64130's QsAEn, BG*, DBG*, L2BG*, and L2DBG* output pins in a multi GT-64130 configuration.

However, in a multi-GT-64130 configuration, AACK* and TA* signals function as sustained tri-state outputs requiring 4.7 KOhm pull-up resistors. All TA* outputs from the GT-64130 devices must be tied together to drive the CPU TA* input.

All AACK* outputs from the GT-64130 devices must be tied together to drive the CPU AACK* input. AACK* and TA* are only driven by the target GT-64130. After asserting TA* LOW for one cycle, it will drive it HIGH for another cycle and then tri-state it. This also applies for AACK*.

In case of a bad CPU address that misses all address windows in all of the GT-64130 devices, no device will assert AACK* and the system will hang. By setting the bit NoMatchCntEn in CPU Configuration Register to 1, the boot GT-64130 will respond after a timeout period, defined in NoMatchCnt field of the CPU Configuration Register, to complete the transaction.

4.13.2 Multi-GT Mode Enabled

When the Multi-GT mode bit is SET, the CPU Interface address decoding reduces to:

1. If A/DL[5-6] == ID AND it's a WRITE, the access is directed to the CPU Interface registers' internal space with A[20-31] defining the specific register offset.
2. If A/DL[5-6] == ID AND it's a READ AND A/DL[4] == 0, the access is directed to the CPU Interface registers' internal space with A[20-31] defining the specific register offset.
3. If A/DL[5-6] == ID AND it's a READ AND A/DL[4] == 1, the access is directed to BootCS*.

NOTE: Since 0xFFF0.0100 (PowerPC boot address) implies A/DL[5-6] == 3, the GT-64130 holding the boot device must be strapped to ID = 3.

4. When the Multi-GT mode bit is CLEARED, the CPU Interface resumes normal address decoding.

NOTE: As long as Multi-GT mode bit is SET, there is no access to PCI, SDRAM and Devices, and DMA internal registers. The only access is to the CPU interface internal registers and to boot ROM.

4.13.3 Initializing a Multiple GT-64130 System

The following is the recommended step-by-step procedure to initialize a system with two GT-64130s attached to the same CPU.

NOTE: Assume that the two GT-64130s are called GT-1 and GT-2, respectively. Both devices have DAdr[10] pulled to VCC (enabling Multi-GT mode). GT-1 has Ready* and CSTiming* tied to 11 (boot GT-64130) and GT-2 has Ready* and CSTiming* tied to 00. GT-1 has the bootrom.

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 A[0-31] is 0xFFF00100 where A[4-6] = 111 and it's a read cycle. Registers on GT-1 are accessible via addresses A[5,6]=11and A[20-31]=offset. Registers on GT-2 are accessible via address A[5,6]=00 and A[20-31]=offset.
2. Access GT-1's BootROM and reconfigure GT-1's CPU Interface address space registers. This reconfigures ALSO, the Internal space address decode register. Later (once 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 0xFFCx.xxxx (i.e. 0xFFBx.xxxx). This causes GT-2 to ignore accesses to 0xFFCx.xxxx once taken out of Multi-GT mode. Also, confirm that the address mapping of the register and memory space in the GT-64130 and on their interfaces is unique. In other words, the four PCI address ranges, two SDRAM ranges, I/O space, and internal GT-64130 register spaces of both system controllers must be different.
4. Clear GT-2 Multi-GT mode bit.
5. Clear GT-1 Multi-GT mode bit.

Now both GT-64130s resume NORMAL operation with USUAL address decoding.

4.13.4 Multi-GT Restrictions

Due to System Interface loading, maximum operating frequency will decrease as the number of GT-64130 increases.

4.14 Multi CPU Support

In a multi CPU system, bus arbitration cannot be achieved by the GT-64130 . An external arbiter is needed.

With an external arbiter, it can work both in QuickSwitch or glueless configurations. However, if a glueless configuration is required, the external arbiter must be designed in such a way that it supports the short between A[0-31] and DL[0-31] - DBG* should be asserted only after address bus is floated. BG* must only be asserted data bus is floated. Also, confirm that there is a dead cycle on every switch between address and data tenures.

It is advised to use the GT-64130 BG* and DBG* outputs when implementing an external arbiter. The arbiter is used to redirect these signals to one of the CPUs.

The GT-64130 also supports cache coherency between CPUs. If one CPU initiates a transaction directed to one of GT-64130 memory spaces, and there is a hit in a MODIFIED line in another CPU L1 cache, the snooped CPU asserts ARTRY*. This assertion indicates that the transaction must be terminated and that the initiator CPU must restart it after the snooped CPU write-backs the line to memory.

When the GT-64130 samples ARTRY*, it ignores the transaction. The ARTRY* sample window as defined in PowerPC bus spec is between TWO cycles after TS* and ONE cycle after AACK* assertion. Due to this sample window, the GT-64130 adds one cycle latency.

4.15 AACK* Delay

60X bus spec defines that AACK* may be asserted as early as the next cycle after TS* assertion. However, a 603e CPU requires in some clock configurations that AACK* is asserted as early as two cycles after TS*.

The GT-64130 supports both options. In default, it asserts AACK* two cycles after TS*. However, if AACK* Delay bit in CPU Configuration Register is set to 1, AACK* asserts one cycle after TS*. It is recommended to set this bit to 1 in order to get the best performance unless the CPU is a 603e with clock ratio of 1:1 or 1:1.5. See the PowerPC spec for more details.

4.16 32-bit Bus Support

603e CPU can be configured to 32-bit wide bus. The GT-64130 must be configured to this mode by DMAReq[0]* and Dadr[9] pins sampled at reset. See Section 11. “Reset Configuration” on page 139 for more information.

In 32-bit mode, there is no need for external mux between address and data bus. DL[0-31] and DP[4-7] signals are not connected, so the GT-64130’s A/DL bus only functions as an address bus.

All transaction attributes are the same as in 64-bit mode. The difference in a 32-bit mode is that the data transfer is one, two, or eight beats. The number of beats depends on the data size. If data is up to 4 bytes, it will be a one beat transaction. If it is eight bytes, it will be a two beat transaction. If it is a cache line (32-bytes), it will be a burst of 8 beats.

NOTE: Two beat burst transactions do not assert TBST*.

Burst order and data alignment in 32-bit mode is similar to the 64-bit mode as shown in Table 16, Table 17, and Table 18.

Table 16: 32-bit Mode Burst Read Ordering

Data Transfer	Start Address			
	A[27-28]=00	A[27-28]=01	A[27-28]=10	A[27-28]=11
1st data beat	DW0-U	DW1-U	DW2-U	DW3-U
2nd data beat	DW0-L	DW1-L	DW2-L	DW3-L
3rd data beat	DW1-U	DW2-U	DW3-U	DW0-U
4th data beat	DW1-L	DW2-L	DW3-L	DW0-L
5th data beat	DW2-U	DW3-U	DW0-U	DW1-U
6th data beat	DW2-L	DW3-L	DW0-L	DW1-L

Table 16: 32-bit Mode Burst Read Ordering (Continued)

Data Transfer	Start Address			
	A[27-28]=00	A[27-28]=01	A[27-28]=10	A[27-28]=11
7th data beat	DW3-U	DW0-U	DW1-U	DW2-U
8th data beat	DW3-L	DW0-L	DW1-L	DW2-L

Table 17: 32-bit Mode Aligned Data Transfers

Transfer Size	TSIZ[0-2]	A[29-31]	Data Bus Byte Lanes			
			0	1	2	3
Byte	001	000	A	-	-	-
	001	001	-	A	-	-
	001	010	-	-	A	-
	001	011	-	-	-	A
	001	100	A	-	-	-
	001	101	-	A	-	-
	001	110	-	-	A	-
	001	111	-	-	-	A
Half word	010	000	A	A	-	-
	010	010	-	-	A	A
	010	100	A	A	-	-
	010	110	-	-	A	A
Word	100	000	A	A	A	A
	100	100	A	A	A	A
Double word 2nd beat	000	000	A	A	A	A
	000	000	A	A	A	A

Table 18: 32-bit Mode Misaligned Data Transfers

Transfer Size	TSIZ[0-2]	A[29-31]	Data Bus Byte Lanes			
			0	1	2	3
Two bytes	010	001	-	A	A	-
	010	101	-	A	A	-
Three bytes	011	000	A	A	A	-
	011	001	-	A	A	A
	011	100	A	A	A	-
	011	101	-	A	A	A

The GT-64130 CPU interface supports fully supports 32-bit mode. Internally, it performs pack/unpack of data to 64-bit.

4.17 PowerQUICC Support

The GT-64130 fully supports the PowerQUICC (MPC860) CPU. It can be configured to this mode by the DMAReq[0]* and Dadr[9] pins sampled at reset, see Section 11. “Reset Configuration” on page 139.

The PowerQUICC bus interface is similar to the 603e in 32-bit mode. Still there are some differences between the two buses:

- Bus arbitration. Not like the PowerPC that has separate address and data bus arbitration signals, the PowerQUICC has only BR* and BG* signals. When it’s granted, it is the bus master of both address and data busses. More over, it has no AACK* signal. It drives valid address and attributes starting TS* till end of transaction.
- PowerQUICC does not support cache coherency.
- PowerQUICC has an internal memory controller that supports access to DRAM or device through the PowerQUICC bus. This means that a transaction might be targeted to PowerQUICC’s own memory, rather than to the GT-64130. More over, PowerQUICC memory controller memory spaces default values after reset overlap the GT-64130 address spaces (especially boot ROM space).
- PowerQUICC cache line is 16-bytes length. This results in bursts reads and writes of four words, rather than eight words as in 603e 32-bit mode.

4.17.1 Bus Arbitration

In PowerPC configuration, the GT-64130 generated bus arbitration signals for the CPU and L2 cache. This is especially essential when using glueless configuration, in which bus arbitration is the mechanism to mux between address and data.

Since in PowerQUICC configuration there is no need for this external mux and there is no L2 cache support, CPU is parked on the bus. The GT-64130 always assert BG*.

In the presence of another bus master (in addition to the PowerQUICC) where bus arbitration is required, use the PowerQUICC internal bus arbiter. In this case, the GT-64130 BG* output is left unconnected.

4.17.2 PowerQUICC Memory Controller

It is recommended to use the GT-64130 memory controller rather than the PowerQUICC memory controller. In this case, the PowerQUICC memory controller must be disabled after reset through PowerQUICC reset configuration sampling (BDIS bit of configuration word).

If the PowerQUICC memory controller is needed, the GT-64130 should be configured to multi-GT mode. The Address ID of the GT-64130 depends whether the boot ROM is controlled by the PowerQUICC or the GT-64130. If the boot ROM is controlled by the PowerQUICC, the Address ID should be different than 11. This is to ensure that the GT-64130 does not respond to access to boot ROM. If the boot ROM resides on the GT-64130's memory bus, the Address ID must be 11. See Section 4.13 "Multiple GT-64130 Support" on page 52 for more details on multi-GT mode. In this case, the PowerQUICC must wake up with BDIS bit of configuration word set.

4.17.3 PowerQUICC Transactions

On Transfer Start, the CPU drives address on A[0-31] and transaction attributes on BURST*,TSIZ,RD/WR*. It keeps this data for the whole transaction.

In case of write transaction, the CPU drives data bus with valid data.

In case of read transaction, the CPU samples data driven on the bus with each TA* asserted by the GT-64130. With each valid read data, the GT-64130 drives odd parity on DP bus.

BURST* and TSIZ defines the number of bytes to be transferred as shown in Table 19.

Table 19: PowerQUICC Transfer Size Summary

BURST*	TSIZ[0-1]	Transfer Size
0	00	16 bytes burst
1	00	4 bytes
1	01	1 byte
1	10	2 bytes
1	11	reserved

On burst transactions (TBST* asserted), the CPU always drives word aligned address (A[30-31] = 0). On burst write, the address is always cache-line aligned (A[28-31] = 0). On burst read, burst order linear wrap around cache line boundary (16 bytes).

Figure 13 and Figure 14 shows typical read and write transactions.

Figure 13: PowerQUICC Read Transaction

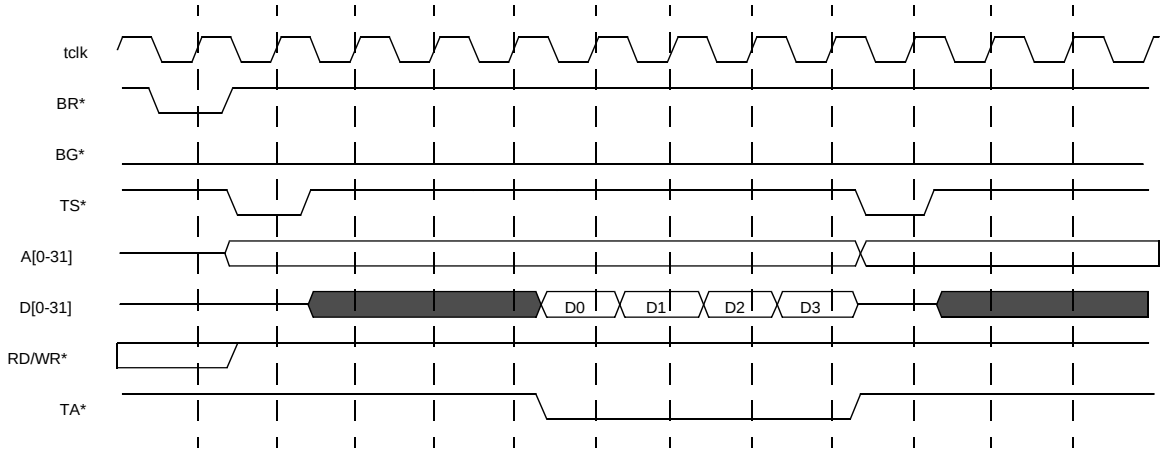
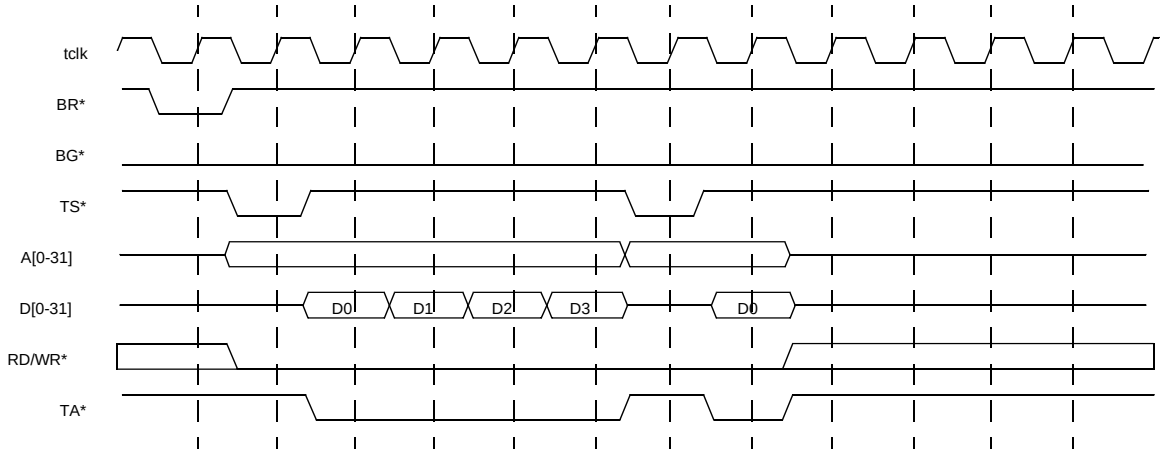


Figure 14: PowerQUICC Write Transactions



4.18 PowerQUICC II Support

The GT-64130 fully supports the PowerQUICC II (MPC8260) CPU.

For the GT-64130 to interface with the PowerQUICC II (MPC8260) bus interface, the PowerQUICC II must be configured to run in the 60x bus compatible mode.

The GT-64130 must be configured to run 64-bit 60x bus with QuickSwitch via the reset configuration. It can be configured to this mode by the DMAReq[0]* and Dadr[9] pins sampled at reset, see Section 11. “Reset Configuration” on page 139.

NOTE: For more information about support for the PowerQUICC II (MPC8260) CPU, request the relevant application note from Galileo Technology.

4.19 CPU Interface Restrictions

1. The GT-64130 does not support eciwx and ecowx instructions. These instructions are optional in PowerPC architecture. If one wishes to use these instructions for access to some other target on the bus, the GT-64130 must be configured to Multi-GT mode. Since the transaction address is not mapped in any of the GT-64130’s address windows, the GT-64130 will not respond and will wait for the target to complete the transaction.
2. The GT-64130 does not support access of more than 4 bytes to internal space.
3. In PowerQUICC configuration with the PowerQUICC memory controller enabled, the GT-64130 must be configured to Multi-GT mode. During system initialization, Multi-GT ID distinguishes between access to the GT-64130 and access to the PowerQUICC’s own memory. After initialization, the GT-64130 and PowerQUICC memory controller will have different address windows.
4. The GT-64130 only supports odd parity. If working with PowerQUICC, the GT-64130 must be configured to odd parity (bit OPAR in PowerQUICC SIUMCR register).
5. If L2 cache is present and a transaction targeted to L2 is pipelined after a transaction targeted to the GT-64130, the L2 cache controller should not assert AACK* until the last TA* driven by the GT-64130. The L2 cache controller may assert AACK* the next cycle after last TA*. It can also assert a new TA* in the same cycle.
6. If BlockRdBuffer bit in CPU Configuration register is set to 1 (bypass read buffer), 64-bit SDRAM must be configured to burst length of 4. Configure PCI and DMA accordingly.
7. If BlockRdBuffer bit in CPU Configuration register is set to 0 (burst read passes through read buffer), read bypass is not supported.

5. MEMORY CONTROLLER

The GT-64130's Memory Controller consists of an integrated SDRAM Controller and a Device Controller.

The Device Contro1.1ller uses the 64-bit muxed AD bus for both address and data transfers.

The SDRAM Controller has a 14-bit address bus (DAdr[11:0], BankSel[1:0]) and shares the 64-bit address/data (AD) bus for data transfers.

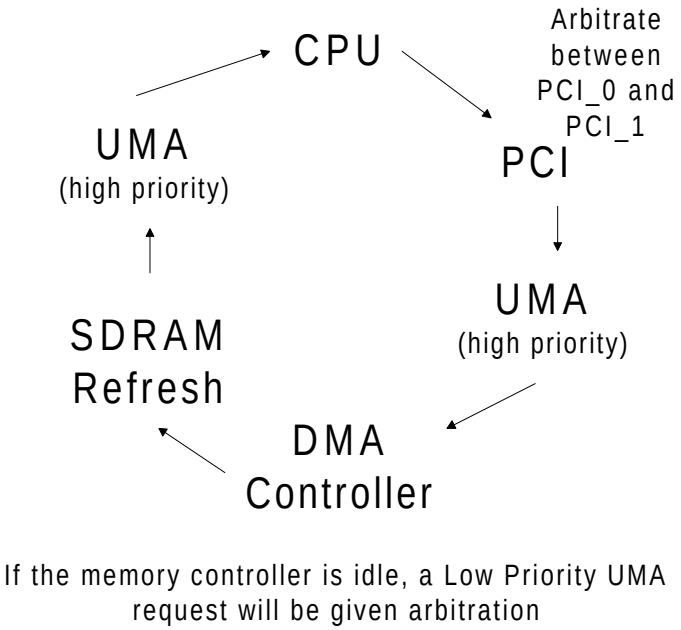
All memory and I/O devices in the GT-64130 system are connected to the AD bus (the A/DL,DH bus is used primarily as a point-to-point connection between the CPU and the GT-64130.)

The memory controller will only MASTER read and write transactions to SDRAM or devices, as instructed from the CPU, DMA controller, or a PCI device on the PCI interface. The GT-64130's Memory Controller can support both 32 or 64-bit SDRAM as well as 8-, 16-, 32-, and 64-bit devices.

NOTE: A device may NOT master transactions via the GT-64130's memory controller.

The GT-64130 implements a round robin arbitration scheme for requests of the memory controller as shown in Figure 15.

Figure 15: Memory Controller Arbitration



5.1 SDRAM Controller

The SDRAM controller supports up to four banks of SDRAMs.

The SDRAM configuration register (0x448) contains configuration information which is valid for all banks. Various access parameters are programmable on a per bank basis as each bank has its own parameters register (0x44c - 0x458).

The supported address depth of the SDRAM varies for each bank separately, depending on whether 16 Mbit, 64 Mbit or 128 Mbit SDRAMs are used (see Section 5.2 “Connecting the Address Bus to the SDRAM” on page 69 for more information). Up to 256 Mbytes can be addressed by each SCS.

5.1.1 SDRAM Configuration Register (0x448)

The SDRAM Configuration Register contains parameters which are used for ALL of the SDRAM banks that are used with the GT-64130.

5.1.1.1 Refresh Rates

The GT-64130 implements standard SCAS before SRAS refreshing.

Refresh rates for all banks are programmable to occur at different frequencies according to the RefIntCnt, a 14-bit value SDRAM configuration register. For example, the default value of RefIntCnt is 0x200. If TClk is 50 MHz, than a refresh sequence will occur every 10us. This is derived from 50MHz (=20ns) * 0x200 (512d) = 10.24us. Every instance that the refresh counter in the GT-64130 reaches its terminal count, a refresh request is sent to the Memory Controller. This request enters the arbiter, and once the AD bus is idle, and the last SDRAM or Device transaction has finished, the refresh cycle will begin.

NOTE: If a UMA transaction is being serviced, the external SDRAM master is responsible for refreshing the SDRAM, see Section 5.7 “Unified Memory Architecture (UMA) Support” on page 76.

5.1.1.2 Non-staggered and staggered Refresh

Non-staggered or staggered refresh for all banks are programmable according to StagRef in the 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 16.

If the SDRAM Controller is programmed to performed staggered refresh (default), SCS[3:0]* will not simultaneously assert LOW together with SRAS* following the low-going SCAS*. Rather, SCS[0]* will first go LOW for 1 cycle, followed by SCS[1]* on the next TClk, and so on. After the last SCS[3]* has asserted LOW for 1 cycle, SCAS* and SRAS* will go HIGH again. Staggered Refresh is useful for load balancing, shown in Figure 17.

Figure 16: Non-Staggered Refresh Waveform

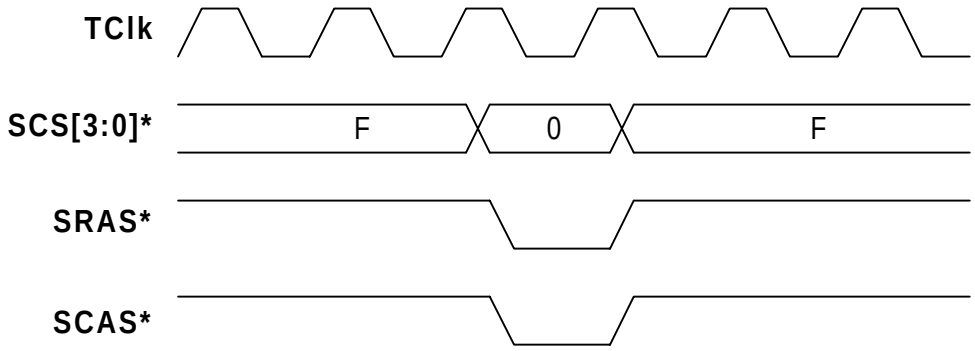
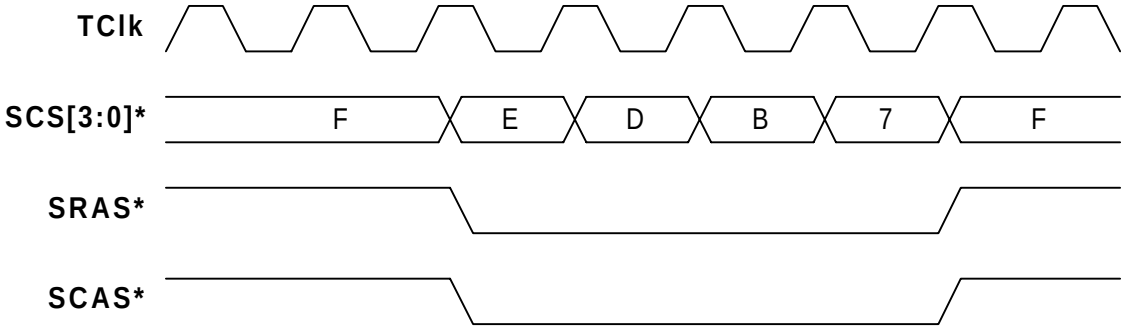


Figure 17: Staggered Refresh Waveform



5.1.1.3 Read Modify Write Enable/ECC

The GT-64130 supports Error Checking and Correction of 64-bit wide SDRAMs. It does not support Error Checking and Correction of 32-bit SDRAMs.

If ECC is enabled, ECC checking is done via the ADP[7:0] on SDRAM reads. If there is an ECC error on the read, an interrupt will be asserted.

For 64-bit SDRAMs, if ECC is enabled, ECC is generated and written to the ADP[7:0] lines on 64-bit writes during the same cycle that the data is written.

Bit 15 enables or disables read modify write protocol to SDRAM. If ECC is enabled in any of the DRAM banks, this bit must be set to 1, to enable read modify write.

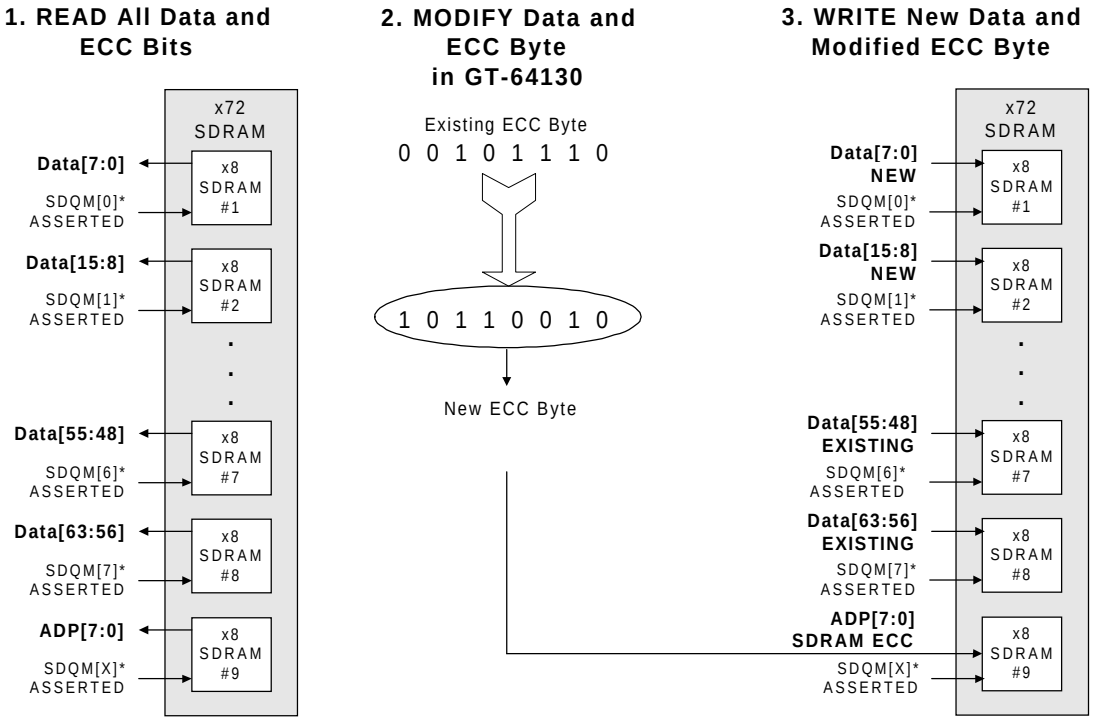
ECC checking and generation requires a 72-bit SIMM to store the ECC information. In order to generate the ECC on partial writes (writes smaller than 64-bits), the current ECC bits must first be read and then modified during the partial write. The protocol for the read modify write transaction is as follows:

1. Read the existing data and ECC information. On this read, all SDQM* lines are asserted (LOW). This means that the BE (byte enable) for the ECC byte can be connected to ANY of the SDQM[7:0]* outputs. The ECC data is read on the ADP[7:0] inputs.

2. Modify the ECC information based on the data that is to be written. The modification of the ECC byte is done in the GT-64130.
3. Write the new data and new ECC byte.

Figure 18 illustrates the procedure that the GT-64130 uses to generate ECC in a partial write to SDRAM.

Figure 18: Read Modify Write Transaction by the SDRAM Controller



5.1.2 Duplicating Signals

Some systems require the use of duplicate signals due to loading requirements. The following sections outlines which signals can be duplicated by setting the appropriate bit in the SDRAM Configuration Register.

5.1.2.1 Duplicating SDRAM Control Lines

SRAS*, SCAS*, and DWr* are the control lines for SDRAM.

These following signals can be duplicated on different pins for loading considerations.

Table 20: Duplicating Signals

By setting bit19 to 1 the signals. . .	are duplicated on pins. .
SRAS*	DMAReq0*/MREQ*
SCAS*	DMAReq3*
DWr*	BypsOE*/MGNT

NOTE: These pins are no longer usable as DMAReq0*/MREQ*, DMAReq3*, and MGNT*/BypOE* when bit 19 is set to 1.

Just for clarification, regardless of the pin strapping of DAdr[7] sampled on RESET (UMA enable), when bit 19 is set to one, the duplication of SRAS*, SCAS*, and DWr* on the DMAReq0*, DMAReq3* and BypsOE* pins takes priority over setting DMAReq[0]* to MREQ* and BypsOE* to MGNT.

5.1.2.2 Duplicating DAdr[11] and BankSel[1] on DMAReq[2:1]

By setting bit 20 to 1, DAdr[11] and BanSel[1] signals are duplicated on DMAReq[2]* and DMAReq[1]*, respectively. These pins are no longer usable as DMAReq2* and DMAReq1* when bit 20 is set to 1.

NOTE: If ECC is implemented in the system, ADP[5:4] cannot be used as DAdr[11] and BankSel[1]. Therefore, in order to use this signals, bit 20 must be programmed to 1 and use DMAReq[2:1]* as DAdr[11] and BankSel[1].

5.1.2.3 Duplicating DMA End of Transfer Pins

The DMA controllers may use End of Transfer pins to give an external device the ability to terminate a current DMA transfer. See Section 8.2.15 “End Of Transfer Enable, EOTE, bit 18” on page 126 for more information about this feature.

Bit 21 controls the function of DMAReq3*.If bit 21 is set to 1, DMAReq3* functions as End of Transfer for channel 0, EOT0*.

Likewise, if bit 22 is set to 0, Ready* functions as Ready. If bit 22 is set to 1, Ready* functions as End of Transfer for channel 1, EOT1*.

Table 21: DMAReq*, Ready* and BypsOE* Functionality

Primary Signal Name	Secondary Signal Name (Programmed on RESET)	Bit 19 = 1	Bit 20 = 1	Bit 21 = 1	Bit 22 = 1
DMAReq[0]*	MREQ*	SRAS*			
DMAReq[1]*			Bank-Sel[1]		
DMAReq[2]*			DAdr[11]		
DMAReq[3]*		SCAS*		EOT[0]*	
Ready*					EOT[1]*
BypsOE*	MGNT*	DWr*			

5.1.3 SDRAM Operation Mode Register (0x474)

The SDRAM Operation Mode Register is a 3-bit register and is used to execute commands other than standard memory reads and writes to the SDRAM. These operations include:

- Normal SDRAM Mode (0x0)
- NOP Commands (0x1)
- Precharge All Banks (0x2)
- Writing to the SDRAM Mode Register (0x3)
- Force a Refresh Cycle (0x4)

In order to execute one of the above commands on the SDRAM, the following procedure occurs:

1. SDRAM Operation Mode Register should be written the corresponding value.
2. This write should be followed by a dummy word (32-bit) write to the corresponding SDRAM.
3. To complete the command, the SDRAM Operation Mode Register should be written 0x0 to place it back into Normal SDRAM Mode.

5.1.3.1 Normal SDRAM Mode

The SDRAM Operation Mode Register must be written 0x0 to enable normal reading and writing to the SDRAM.

5.1.3.2 NOP Commands

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

5.1.3.3 Precharge Both Bank

The Precharge Bank command is used to deactivate the open row in a particular bank or the open row in both banks. Once 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.

5.1.3.4 Writing to the SDRAM's Mode Register

Each SDRAM has its own Mode Register.

The Mode Register is used to define the specific mode of operation for the SDRAM. This definition includes the selection of a burst length, SCAS latency, burst type, operating mode, etc. See your SDRAM data sheet for more information about this register.

Typically, the Mode Register of each SDRAM is initialized on boot-up of the system and is kept static. The GT-64130 has the flexibility to allow the CPU or a PCI Master to update the SDRAM's Mode Register at any time during operation.

The parameters that the GT-64130 can change are the CAS latency and the burst length. In order to change these parameters in the SDRAM's Mode Register, the corresponding SDRAM Bank Parameters Register (0x44c - 0x458) must be updated with the correct values. Then, the SDRAM Operation Mode Register must be written to 0x3 indicating a Write Command to the SDRAM Mode Register. This write must be followed by a dummy word (32-bit) write to the corresponding SDRAM whose Mode Register must be updated. To complete this command, the SDRAM Operation Mode Register must be written 0x0 to place it back into Normal SDRAM Mode.

The GT-64130 uses the following procedure to automatically initialize the SDRAM on boot up. This default initialization can be overwritten by the protocol described above.

1. SRAS* and DWr* are asserted with DAdr[10] HIGH and SCS[3:0] = 0000. This indicating a Precharge to all SDRAM Banks.
2. SRAS* and SCAS* are asserted with SCS[3:0] = 0000, indicating a CBR (CAS before RAS) refresh to all SDRAM Banks. This occurs twice in a row.
3. SRAS*, SCAS*, and DWr* are asserted four 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 activating each of the four chip selects (SCS[3:0]) individually.[1]

NOTE: The GT-64130 automatically initializes the SDRAM on boot up to Sub-block burst ordering. It must be changed to linear ordering before the first PPC CPU cache line read. It is done by programing SDRAM Burst Mode register to 0x9, and then follow the above procedure.

5.1.3.5 Force Refresh

The Force Refresh Command is used to execute a refresh cycle on the particular bank that is accessed.

5.1.4 SDRAM Address Decode Register (0x47c)

The Address Decode Register is a three bit register which determines how bits of an address presented on the A/DL or PCI bus are translated to row and column address bits on DAdr[10:0] and BankSel[0] (16 Mbit) or DAdr[11:0] and BankSel[1:0] (64/128 Mbit). This flexibility allows the designer to choose the address decode setting which gives the software a better chance of interleaving, thus enhancing overall system performance.

The row and column address translation is different for 16 Mbit or 64 Mbit and 128 Mbit SDRAMs as well as 32-bit and 64-bit SDRAM banks. Therefore, the address decoding will depend on the setting of AddrDecode at 0x47c. See Table 22 through Table 25 for the SRAS* and SCAS* address translation from the A/DL interface and PCI.

Table 22: CPU/PCI Address Decoding for 32-bit SDRAM, 16 Mbit

AddrDecode, 0x47c	CPU/PCI Address Bits used for SRAS* on BankSel[0],DAdr[10:0]	CPU/PCI AddressBits used for SCAS* on BankSel[0],DAdr[10:0]
000	4, 21-11	4, “0”, 23-22, 10-5, 3-2
001	5, 21-11	5, “0”, 23-22, 10-6, 4-2
010	11, 21-12, 10	11, “0”, 23-22, 9-2
011	12, 21-13, 11-10	12, “0”, 23-22, 9-2
100	20, 21, 19-10	20, “0”, 23-22, 9-2
101	21, 20-10	21, “0”, 23-22, 9-2

1. The GT-64130 user must program the SDRAM's mode register to burst in the linear wrap around order that supports the PowerPC CPU burst order. The other modes that are programmed following boot up are the default values of the SDRAM control and parameters register.

Table 22: CPU/PCI Address Decoding for 32-bit SDRAM, 16 Mbit (Continued)

AddrDecode, 0x47c	CPU/PCI Address Bits used for SRAS* on BankSel[0],DAdr[10:0]	CPU/PCI AddressBits used for SCAS* on BankSel[0],DAdr[10:0]
110	22, 21-11	22, “0”, 23, 10-2 (only for x4 & x8)
111	23, 21-11	23, “0”, 22, 10-2 (only for x4)

Table 23: CPU/PCI Address Decoding for 32-bit SDRAM, 64 Mbit

AddrDecode, 0x47c	CPU/PCI Address Bits used for SRAS* on BankSel[0], BankSel[1], DAdr[11:0]	CPU/PCI AddressBits used for SCAS* on BankSel[0], BankSel[1], DAdr[11:0]
000	4, 5, 23-12	4 ,5, “00”, 25-24, 11-6, 3-2
001	5, 6, 23-12	5, 6, “00”, 25-24, 11-7, 4-2
010	11, 12, 23-13, 10	11, 12, “00”, 25-24, 9-2
011	12, 13, 23-14, 11-10	12, 13, “00”, 25-24, 9-2
100	20, 21, 23-22, 19-10	20, 21, “00”, 25-24, 9-2
101	22, 23, 21-10	22, 23, “00”, 25-24, 9-2
110	23, 24, 21-10	23, 24, “00”, 25, 22, 9-2 (only for x4 & x8)
111	24, 25, 21-10	24, 25, “00”, 26, 22, 9-2 (Only for x4)

Table 24: CPU/PCI Address Decoding for 64-bit SDRAM, 16 Mbit

AddrDecode, 0x47c	CPU/PCI Address Bits used for SRAS* on BankSel[0],DAdr[10:0]	CPU/PCI Address Bits used for SCAS* on BankSel[0],DAdr[10:0]
000	5, 22-12	5, “0”, 24-23, 11-6, 4-3
001	6, 22-12	6, “0”, 24-23, 11-7, 5-3
010 (UMA)	11, 22-12	11, “0”, 24-23, 10-3
011	13, 22-14, 12-11	13, “0”, 24-23, 10-3
100	21, 22, 20-11	21, “0”, 24-23, 10-3
101	22, 21-11	22, “0”, 24-23, 10-3
110	23, 22-12	23, “0”, 24, 11-3 (only for x4 & x8)
111	24, 22-12	24, “0”, 23, 11-3 (only for x4)

Table 25: CPU/PCI Address Decoding for 64-bit SDRAM, 64/128 Mbit

AddrDecode, 0x47c	CPU/PCI Address Bits used for SRAS* on BankSel[0], BankSel[1], DAdr[11:0]	CPU/PCI AddressBits used for SCAS* on BankSel[0], BankSel[1], DAdr[11:0]
000	5, 6, 24-13	5, 6, 27, “0”, 26-25, 12-7, 4-3
001	6, 7, 24-13	6, 7, 27, “0”, 26-25, 12-8, 5-3
010	11, 12, 24-13	11, 12, 27, “0”, 26-25, 10-3
011	13, 14, 24-15, 12-11	13, 14, 27, “0”, 26-25, 10-3
100	21, 22, 24-23, 20-11	21, 22, 27, “0”, 26-25, 10-3
101	23, 24, 22-11	23, 24, 27, “00”, 26-25, 10-3
110	24, 25, 22-11	24, 25, 27, “00”, 26, 23, 10-3 (Only for x4 & x8)
111	25, 26, 22-11	25, 26, 27, “0”, 24-23, 10-3 (Only for x4)

5.2 Connecting the Address Bus to the SDRAM

The SDRAM controller has its own address bus and the way it connects to SDRAM depends on the type being used, 16 Mbit, 64 Mbit, or 128 Mbit SDRAMs.

5.2.1 16 MBit SDRAMs

For 16 Mbit SDRAMs, DAdr[10:0] and BankSel[0] are outputs of the GT-64130 and must be directly connected to address bits 10-0 and Bank Select of the actual SDRAM.

NOTE: DAdr[11] and BankSel[1] are NOT used when connecting to 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. BankSel[0] is held constant from the SRAS cycle.

With 16 MBit SDRAMs, the GT-64130 supports a maximum of 4M addresses, 12 address bits for SRAS and 10 address bits for SCAS.

5.2.2 64/128 Mbit SDRAMs

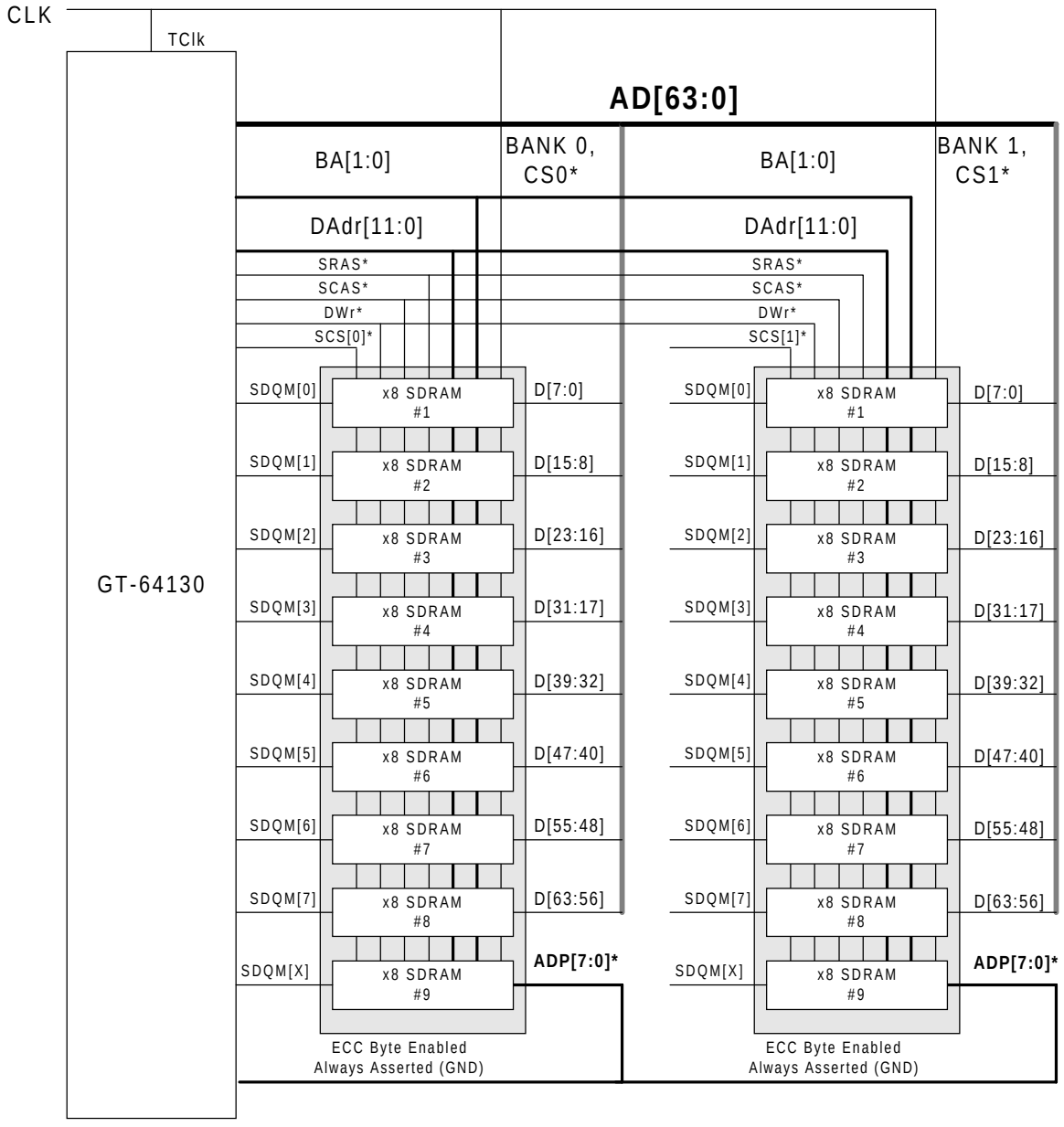
For 64/128 MBit SDRAMs, DAdr[11:0] and BankSel[1:0] are outputs of the GT-64130 and must be directly connected to address bits 11-0 and Bank Select of the actual SDRAM.

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. BankSel is held constant from the SRAS cycle.

With 64-MBit SDRAMs, the GT-64130 supports a maximum of 16M addresses, 14 address bits for SRAS and 10 address bits for SCAS (DAdr[11] is ignored). With 128-MBit SDRAMs, the GT-64130 supports a maximum of 32M addresses, 14 address bits for SRAS and 11 address bits for SCAS.

5.3 64/128 Mbit/64-bit SDRAM Connection to Memory Bus Using x8 Devices

Figure 19: 64/128 Mbit/64-bit SDRAM Connection to Memory Bus Using x8 Devices



5.4 Programmable SDRAM Parameters

The SDRAM controller of the GT-64130 supports a wide range of SDRAMs with different access times and each bank can be programmed independently by the SDRAM Bank[3:0] Parameter registers (0x44c-0x458). These parameters include the number of clock cycles (based on TClk) between active SRAS* and SCAS*.

NOTE: To update the SCAS* Latency or the Burst Length, follow the procedure outlined in Section 5.1.3.4 "Writing to the SDRAM's Mode Register" on page 66 to update the SDRAM's Mode Register.

5.4.0.1 SCAS Latency*

SCAS* Latency is the number of TClks from the assertion of SCAS* to the sampling of the first read data. This parameter can be programmed to be either 2 or 3 TClks. Selecting this parameter will depend on TClk frequency and the speed grade of the SDRAM. Check your SDRAM data sheet for the optimal setting.

5.4.0.2 Flow-through

Bit 2 specifies the number of times that the data is sampled by the GT-64130 on SDRAM reads when bypass is not enabled (bit 9 = 0). This option is included for future designs which will run at faster clock frequencies.

NOTE: As of January 1999, Flow-through mode should always be enabled (bit 2 = 1). If ECC is used, this mode must be disabled.

5.4.0.3 SRAS Precharge*

Bit 3 specifies the SRAS precharge time. This parameter specifies the number of TClks following a precharge cycle that a new SRAS* transaction may occur.

Setting this bit to 0 means the SRAS to precharge time will be 2 TClk cycle.

Setting this bit to 1 means the SRAS to precharge time will be 3 TClk cycles.

5.4.0.4 64-bit Interleaving

Bit 5 specifies how many banks are supported for interleaving if the bank is set for 64/128 Mbit SDRAM. If the bank is NOT set for 64/128 Mbit SDRAM (bit 11 = 0), the setting of this bit is irrelevant.

Setting this bit to 0 means the GT-64130 will interleave between two banks.

Setting this bit to 1 means the GT-64130 will interleave between four banks.

5.4.0.5 Bank Width

Bit 6 specifies the data width of the particular bank.

Setting this bit to 0 means the bank width is 32 bits.

Setting this bit to 1 means the bank width is 64 bits.

5.4.0.6 Bank Location

Bit 7 specifies the location of the bank if the bank is set for 32-bit SDRAM. If the bank is NOT set for 32-bit SDRAM (bit 6 = 1), the setting of this bit is irrelevant.

Setting this bit to 0 means the 32-bit SDRAM is controlled on the even bank, AD[31:0].

Setting this bit to 1 means the 32-bit SDRAM is controlled on the odd bank, AD[63:32].

5.4.0.7 ECC Support

Bit 8 enables or disables ECC on a 64-bit wide SDRAM bank.

If set to 1 ECC is enabled, indicating a 72-bit SIMM.

5.4.0.8 64-bit Bypass Mode for CPU Reads

Bit 9 enables or disables 64-bit SDRAM Read bypass.

NOTE: This option is ONLY for SDRAM banks configured as 64-bit (the option is not available for devices).

There is an optional bypass mode for CPU reads where a clock cycle of latency can be eliminated when the CPU executes read cycles from 64-bit SDRAM. Using the bypass mode requires the addition of bus switches to enable the flow of data to the CPU directly. Instead of passing response data from the SDRAM to the GT-64130 prior to presenting it to the CPU, data can flow directly from the SDRAM to the CPU via bus switches. This reduces the latency from TS* to TA* by one TClk cycle.

Setting this bit to 0 means bypass is disabled. All reads from 64-bit SDRAM flow through the GT-64130 before being presented to the CPU on the DL,DH lines.

Setting this bit to 1 means the BypsOE* output signal is active on any reads from a 64-bit SDRAM (64-/32-/16-/8- bit reads) and controls the bus switches which directly connect the CPU's DL,DH lines to the SDRAM data lines.

NOTE: The bypass is used for partial reads, as well. Reads from 64-bit SDRAM are no longer sampled by the GT-64130 prior to presenting the data to the CPU.

64-bit bypass are only be enabled if the bank is set for 64-bit (bit 6 = 1).

No writes will ever be transferred via the bypass switches.

5.4.0.9 SRAS* to SCAS*

Bit 10 specifies the number of TClks that the GT-64130 inserts between the assertion of SRAS* with a valid row address to the assertion of SCAS* with a valid column address.

Setting this bit to 0 means the SRAS to SCAS latency is 2 Tclk cycles (default).

Setting this bit to 1means the SRAS to SCAS latency will be 3 TClk cycles.

5.4.0.10 16/64/128 MBit SDRAM Configuration

Bit 11 specifies when the particular bank supports 16 or 64/128 MBit SDRAMs.

Setting this bit to 0 means the bank supports 16 MBit SDRAMs.

Setting this bit to 1 means the bank supports 64/128 Mbit SDRAMs.

5.4.0.11 Burst Length

Bit 13 specifies the data burst length supported for the particular SDRAM bank.

Setting this bit to 0 means the bank supports 8 data bursts.

Setting this bit to 1 means the banks supports 4 data bursts.

NOTE: The data can be either 32 or 64 bit, depending on the setting of bit 6.

5.5 SDRAM Performance

SDRAM performance varies on the CPU and PCI interface, depending on the setting of certain variables.

5.5.1 CPU Access to SDRAM

SDRAM performance on the CPU interface is based on the latency between the CPU's assertion of TS* to the GT-64130's assertion of TA* returning the first data on a burst read (cache line read). Performance is different if the 64-bit Bypass feature is enabled.

Table 26 summarizes the latency between TS* and TA* on SDRAM reads. After the first data is read, the remaining data is returned with zero wait states. For example, if bypass is enabled, and the CPU executes a cache line read from memory, data is returned with a 1-1-1 performance when bypass is enabled.

Table 26: CPU SDRAM Performance on Reads

SDRAM device	Number of TClks between TS* to TA*
Bypass Enabled[1]	9
Bypass Not Enabled	10

1. See Section 5.4.0.8 for more information about the bypass feature.

On CPU writes to SDRAM, the data cycles follow the address cycle with zero wait states. Further, the next data of a burst can also be written on the next clock cycle (zero wait states).

5.5.2 PCI Read Performance from SDRAM

The following sections outlines SDRAM memory performance.

These figures depend on a number of variables including the PClk/TClk ratio, as well as the sync. mode configured for the device. The numbers in Table 29are based on the fastest SDRAM settings.

Performance is rated on three events:

Table 27: PCI Read Performance Events

Event	Description
Event A: PCI Read request from SDRAM.	This event is based on the number of clocks between Frame* being asserted by a PCI master to the assertion of SRAS* by the SDRAM controller.
Event B: Data from SDRAM controller to PCI.	This event is based on the number of clocks between the first data placed on the AD bus until TRdy* is asserted on the PCI bus by the GT-64130.
Event C: Latency till first data.	This event is based on the number of clocks between a PCI master asserting Frame* to the assertion of TRdy* by the GT-64130.

There are five different sync. modes that the GT-64130 can be placed in depending on the PClk/TClk ratio. These sync. modes are:

Table 28: GT-64130 Sync. Modes

Sync. Mode	Description
0, x00	No assumptions on TClk/PClk ratio.
1, 001	PClk frequency is greater than or equal to 1/2 TClk frequency.
2, 01x	PClk frequency is synchronized[1] to TClk frequency and greater than or equal to 1/2 TClk frequency.
5, 101	PClk frequency is greater than or equal to 1/3 TClk frequency but smaller than 1/2 TClk frequency.
6, 11x	PClk frequency is synchronized to TClk frequency and greater than or equal to 1/3 TClk frequency but smaller than 1/2 TClk frequency.

1. If TClk and PClk are synchronized, see Table 281 on page 257.

Table 29: SDRAM Performance Summary PCI Read Accesses

TClk/PClk Frequency (MHz)	Sync. Mode	Event	# of TClk	# of PClk
66/33	0	A B C	11 9 24	6 5 12
	1	A B C	11 9 24	6 5 12
	2	A B C	9 5 18	5 3 9
75/33	0	A B C	11 10 24	5 5 11
	5	A B C	11 10 24	5 5 11

Table 29: SDRAM Performance Summary PCI Read Accesses (Continued)

TClk/PClk Frequency (MHz)	Sync. Mode	Event	# of TClk	# of PClk
66/50	0	A B C	8 7 19	6 5 14
	1	A B C	9 7 20	7 6 15
75/50	0	A B C	9 8 21	6 6 14
	1	A B C	9 8 21	6 6 14
75/66	0	A B C	8 6 18	7 6 16
	1	A B C	8 6 18	7 5 16
75/25	0	A B C	14 12 30	5 4 10
	5	A B C	14 12 30	5 4 10
	6 or 7	A B C	12 8 24	4 3 8

5.6 SDRAM Interleaving

The GT-64130 supports two bank interleaving with 16 Mbit SDRAM and two or four bank interleaving with 64 Mbit or 128 Mbit SDRAMs.

The SDRAM Address Control Register (0x47c) determines what address bits are used for SRAS and SCAS cycles. This allows flexibility for different software applications to select an address decoding scheme which may give data accesses a better probability of interleaving.

Interleaving provides higher system performance by hiding SRAS to SCAS cycles and precharge time of a pending transaction during the data cycles of a current transaction. This reduces the number of wait states before data is read from or written to SDRAM, thus increasing bandwidth.

At the end of every SDRAM memory transaction, the GT-64130 precharges the bank.

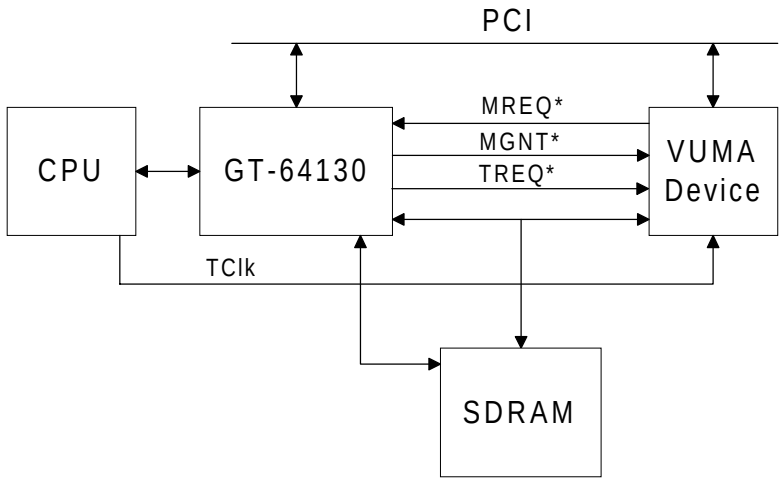
5.7 Unified Memory Architecture (UMA) Support

The GT-64130 supports UMA.

UMA allows an external master device to share the same physical SDRAM memory that is controlled by the GT-64130. It works according to the VESA Unified Memory Architecture (VUMA) specification[1].

A VUMA device refers to any type of controller that needs to share the same physical system memory and have direct access to it as shown in Figure 20.

Figure 20: VUMA Device and GT-64130 sharing SDRAM



5.7.1 UMA Hardware Support

UMA is enabled by a pin strapping option.

If DAdr[7] is sampled LOW on RESET, DMAReq0* is programmed to function as MREQ*. The BypsOE* pin will function as MGNT* as long as bit 9 is 0 for all of the SDRAM Bank Parameters registers (bypass disabled).

NOTE: The Bypass feature and UMA cannot be used simultaneously.

MREQ* is an input into the GT-64130 and must be an output for the VUMA device. The VUMA device uses this signal to make a request to the GT-64130 to access the shared SDRAM. MREQ* must be driven by the VUMA device on a rising edge of TClk. MREQ* is sampled on a rising edge of TClk by the GT-64130.

1. More information about the VESA Unified Memory Architecture can be found at http://www.vesa.org

MGNT* is an output of the GT-64130 and must be an input to the VUMA device. The GT-64130 uses this signal to inform the VUMA device that it can now access the shared SDRAM. MGNT* is driven by the GT-64130 on a rising edge of TClk. MGNT* must be sampled by the VUMA on a rising edge of TClk.

The AC Timing parameters are shown Table 30.

Table 30: UMA AC Timing Parameters

Signals	Description	Min.	Max.	Unit	Loading
MREQ*	Setup	3		ns	
MREQ*	Hold	1		ns	
MGNT*	Output Delay From TClk rising	2	10	ns	30 pF

5.7.2 SDRAM Pins

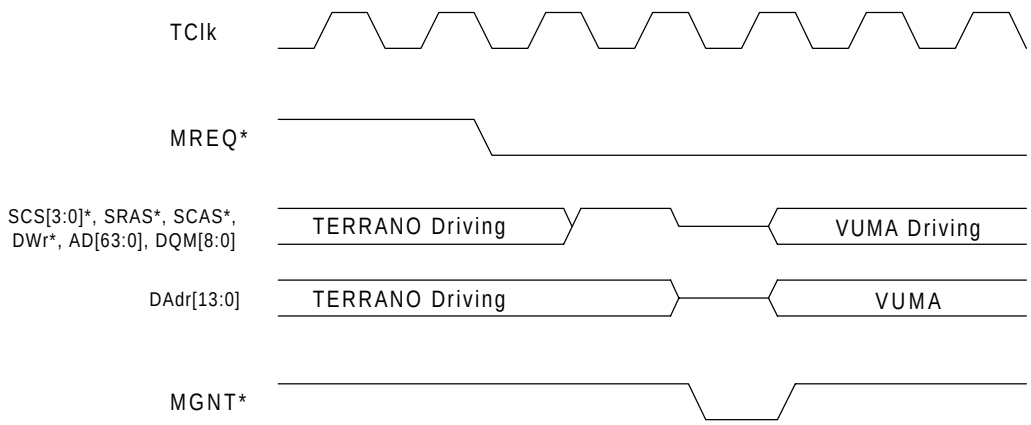
Once MGNT* is asserted by the GT-64130 and the VUMA are granted access to:

- SDRAM
- SCS[3:0]*
- SRAS*
- SCAS*
- DWr*
- AD[63:0]
- DQM[8:0]
- DAdr[11:0]
- BankSel[1:0]

These signals are held in sustained tri-state until the GT-64130 regains access to SDRAM. During this period, the VUMA device must drive these signals to access SDRAM.

When the GT-64130 and the VUMA 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). The SDRAM address lines do not need to be driven HIGH before floating the bus. Figure 21 shows a sample waveform.

Figure 21: Handing the Bus Over



5.7.3 Address Decoding

The GT-64130 complies with the standard for CPU address to SDRAM Row and Column addressing. See Section 5.1.4 “SDRAM Address Decode Register (0x47c)” on page 67 for information about how the CPU address translation is performed.

This register (0x47c) must be programmed with the binary value of 010 to properly use the UMA feature when using 16/128 Mbit/64-bit SDRAM.

NOTE: As of December 1997, there is no standard for UMA SDRAM addressing when using 16 Mbit/32-bit SDRAM or 64/128 MBit SDRAMs (32 or 64 bit). If you are using any of these configurations, the UMA device must use the same address decoding scheme as the GT-64130 (set by register 0x47c).

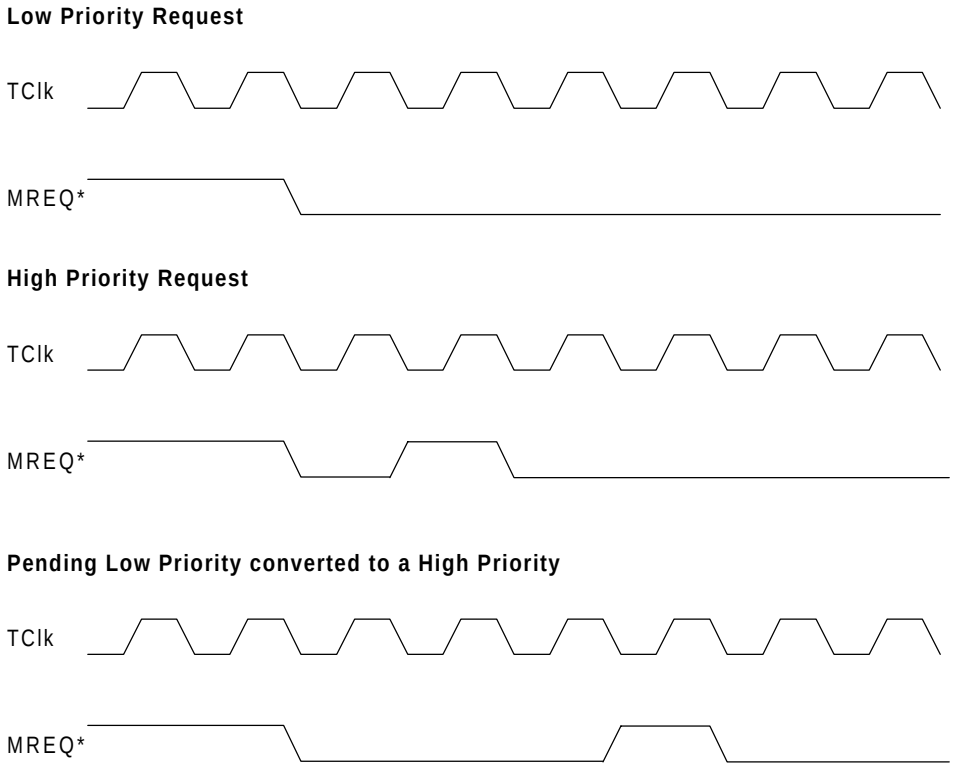
5.7.4 Arbitration

As shown in Figure 20, the VUMA device arbitrates with the GT-64130 for access to SDRAM through MREQ* and MGNT* which should be synchronous to TClk.

The GT-64130 is the default owner of the SDRAM and access to this memory is only allowed to the VUMA upon demand. The GT-64130 has the right to take the VUMA device off of the bus by de-asserting MGNT*.

VUMA devices may request access to SDRAM with either a low or high priority. These priorities are conveyed to the core logic through the MREQ* signal.

Figure 22: MREQ* Requests from the VUMA Device[1]



The following rules must be followed when the VUMA device makes a request for access to the SDRAM:

1. Once MREQ* is asserted by the VUMA for a low priority request, it must keep it asserted until the VUMA device is given access to SDRAM via MGNT*. The only reason to change the MREQ* pin's status is to raise a high priority request or raise the priority of an already pending low priority request.
   - If MGNT* is sampled asserted, the VUMA device must not deassert MREQ*. Instead, the VUMA device has ownership of SDRAM and must assert MREQ* until it has completed its transaction.
   - If MGNT* is sampled de-asserted, the VUMA device de-asserts MREQ* for one clock and asserts it again regardless of the status of MGNT*. After reassertion, the VUMA device must keep MREQ* asserted until the GT-64130 gives the VUMA access to SDRAM via MGNT*.
2. The VUMA only asserts the MREQ* for the purpose of accessing SDRAM and must stay asserted until MGNT* is sampled asserted except to raise the priority request. No speculative requests or request abortion is allowed.
3. Once the VUMA samples MGNT* as asserted, it gains and retains access to SDRAM until MREQ* is de-asserted.
4. The GT-64130 always asserts MGNT* for one clock cycle only. It immediately requests back ownership of the bus.

1. Any other transitions asserted other than those shown in this figure will keep the state machine in the current state.

5. The VUMA can retain ownership of SDRAM indefinitely. The standard calls for the VUMA device to keep ownership for no longer than 60 TClks before it must release the bus. This is not a requirement for the GT-64130 and it will wait until the VUMA device releases the bus by de-asserting MREQ*.
6. When the VUMA device has ownership of the bus, it has full responsibility to execute refresh cycles on the SDRAM.
7. Once the VUMA de-asserts MREQ* to transfer ownership back to the GT-64130 either on its own or because of a preemption requires it, MREQ* must be de-asserted for at least two TClks before asserting it again to raise a request.

5.7.5 Latencies, Low and High Priority

If a VUMA places a low priority request for access to SDRAM, there is no set time specified by the GT-64130 to assert MGNT*. Once there are no pending transactions to the memory controller, MGNT* is asserted.

If a VUMA places a high priority request for an access to SDRAM, the GT-64130 has a maximum of 35 TClks before it will assert MGNT*.

5.7.6 Total Request

The GT-64130 immediately requests back ownership of the bus after MGNT* assertion.

This might cause some difficulty to implement a fair arbitration mechanism on the SDRAM bus. If bit 4 of SDRAM Bank2 Parameters register is set to 1, DMAReq[3]*/SCAS* pin functions as a total request pin. It indicates whether there is a real internal request inside the GT-64130 that requires SDRAM bus ownership.

5.7.7 Disable Refresh

In some applications, the GT-64130 will be most of the time OFF the SDRAM bus. The bus master has the full responsibility to execute refresh cycles on the SDRAM.

In such applications, the user may want to disable refresh cycles of the GT-64130 (make memory controller arbitration cycle shorter). If bit 4 of SDRAM Bank1 Parameters register is set to 1, refresh is disabled.

5.8 Device Controller

The device controller supports up to five banks of devices. Various access parameters can be programmed on a per bank basis as each bank has its own parameters register (0x45c - 0x46c).

The supported memory space of each device bank can vary for each bank separately up to 256 Mbytes. The width of each bank may be 8-, 16-, 32- or 64-bits.

The maximum device address space is 512Mbytes for five banks. The five individual chip selects are typically broken up into four individual device banks plus one chip select for a boot device (non-volatile memory). Each device bank can have unique programmable timing parameters to accommodate different device types (e.g., Flash, ROM, I/O Controllers). The devices share the local AD bus with the SDRAM. But unlike SDRAM, the devices use the AD bus as a multiplexed address and data bus.

In the address phase, the device controller puts an address on the AD bus with a corresponding Chip Select asserted. ALE indicates the AD bus is output as address with a valid CS*. ALE is used to latch the address and

the CS* in an external 373 device. ALE is HIGH by default, making the latch transparent. ALE goes LOW a half clock cycle before CSTiming* is asserted for the particular read or write transaction. At the completion of the transaction, ALE goes HIGH again on the same rising TClk that CSTiming* is de-asserted.

CS* must then be qualified (OR-tied) with CSTiming*. A read or write cycle is indicated by DevRW*. The CSTiming* signal will be valid for the programmable number of cycles of the specific CS* that is active. Turn-Off, AccToFirst and AccToNext can be set in registers 0x45c - 0x46c for each bank's read timing parameters. ALEtoWr, WrActive and WrHigh can be set for each bank's write timing parameters. There are certain restrictions to setting these timing parameters. See Section 5.11 "Memory Controller Restrictions" on page 92 before configuring these bits.

NOTE: Some of these parameters can be extended by the Ready* pin, see Section 5.8.10 "Ready* Support" on page 85.

5.8.1 TurnOff, bits [2:0]

TurnOff is the number of TClk cycles that the GT-64130 will not drive the memory bus after a read from a device. This prevents contentions on the memory bus after a read cycle for a slow device.

This parameter is measured from the number of cycles between the deassertion of DevOE* and a new AD bus cycle. DevOE* is an externally extracted signal that is the logical OR between CSTiming* and inverted DevRW*.

5.8.2 AccToFirst, bits [6:3]

AccToFirst defines the number of cycles in a read access from the assertion of CS* (first rising TClk where CS* is asserted LOW) to the cycle that the first data will be sampled by the GT-64130. This parameter can be extended by the Ready* pin.

5.8.3 AccToNext, bits [10:7]

AccToNext defines the number of cycles in a read access from the cycle that the first data was latched to the cycle to the next data will be latched (in burst accesses).

This parameter can also be thought of as the delay between the rising edge of TClk which data is latched to the rising edge of TClk where the next data is latched in a burst cycle.

This parameter can be extended by the Ready* pin.

5.8.4 ALEtoWr, bits[13:11]

There are eight byte write signals, Wr[7:0]*.

ALEtoWr can also be thought as the delay between the rising edge of TClk which drives ALE LOW to the assertion of Wr*, or for the first write pulse.

NOTE: The Wr[7:0]* signals are asserted and de-asserted off of the FALLING edge of TClk.

5.8.5 WrActive, bits[16:14]

WrActive is the number of TClks that Wr* are active (asserted).

This parameter is measured from the first rising edge of TClk where Wr* is asserted LOW to the last rising edge of TClk where Wr* is LOW for that particular write pulse.

This parameter can be extended by the Ready* pin.

NOTE: The Wr[7:0]* signals are asserted and de-asserted off of the FALLING edge of TClk.

5.8.6 WrHigh, bits[19:17]

WrHigh is the number of TClks that Wr* is inactive between burst writes.

This parameter is measured from the first rising edge of TClk, where Wr* is de-asserted HIGH, to the last rising edge of TClk where Wr* is HIGH. On the next rising edge of TClk, Wr* will be asserted LOW for the next write pulse.

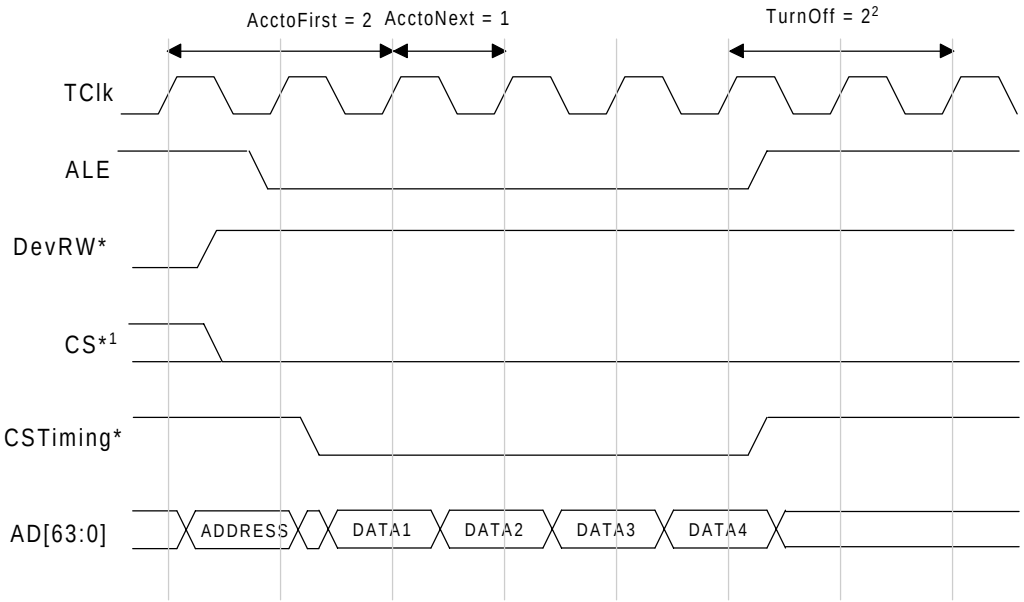
NOTE: The Wr[7:0]* signals are asserted and de-asserted off of the FALLING edge of TClk. The previous data will remain on the AD bus during WrHigh.

5.8.7 Device Bank Width and Location

Bit 21:20 of the Device Bank[3:0] Parameter registers (0x45c-0x46c), DevWidth specifies whether the data width of the particular device bank is 8-, 16-, 32- (default except for BootCS*), or 64-bit.

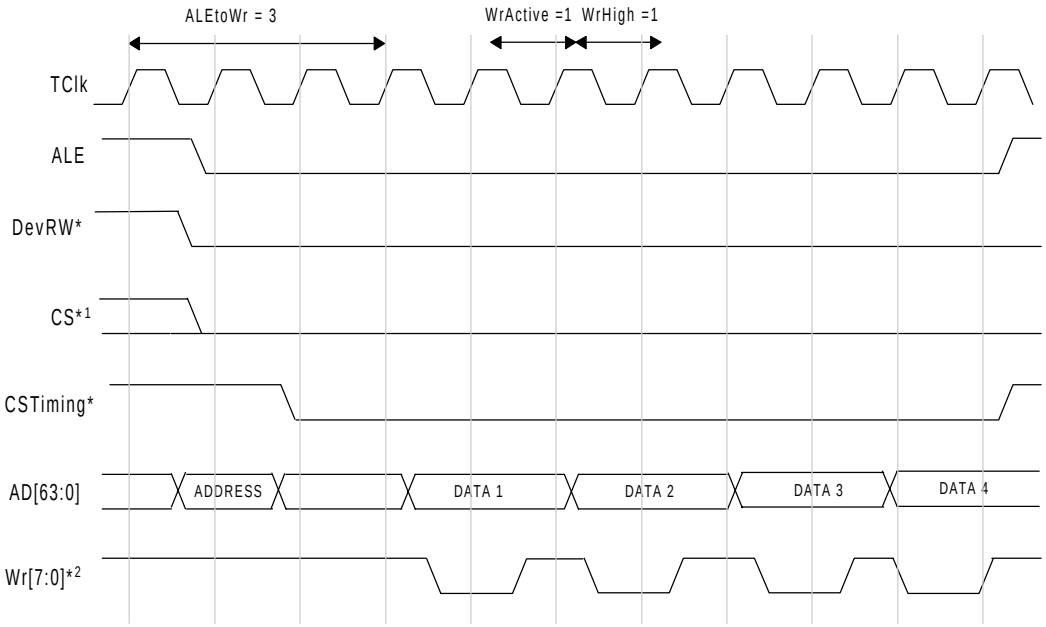
If the bank is set for 8-, 16-, or 32-bit operation, it can either reside on the even half (31:0) or odd half (63:32) by setting bit 23 (DevLoc). Selecting the even or the odd half allows for load balancing. In case of a 64-bit device, DevLoc has no meaning.

Figure 23: Waveform Showing Device Read Parameters



Notes:
1. CS* is driven off the same rising TClk* as ALE. Throughout consecutive transactions to the same device, CS* will remain asserted. This is why CS* must ALWAYS be qualified with CSTiming*.
2. GT-64130 may start a new AD cycle after TurnOff.

Figure 24: Waveform Showing Device Write Parameters



Notes:
1. CS* is driven off the same rising TClk* as ALE. Throughout consecutive transactions to the same device, CS* will remain asserted. This is why CS* must ALWAYS be qualified with CSTiming*.
2. Wr[7:0]* are asserted and deasserted from the falling edge of TClk.

5.8.8 Burst Writes

The device controller supports up to eight word burst accesses. The burst address is supported by a 3-bit wide address bus (BAdr[2:0]) that is different from the latched address on the multiplexed AD bus.

NOTE: BAdr[2:0] are the same pins as the least significant SDRAM address lines, DAdr[2:0]. See Section 20. “GT-64130 Pinout Table, 388 pin BGA” on page 240 for more information.

5.8.9 Packing and Unpacking Data and Burst Support

The AD bus supports the packing of data into a 64-bit double word, in reads from devices that are 8-, 16-, or 32-bits wide.

Devices that are 8-bits or 16-bits wide only are supported by partial reads (up to 64-bits). The controller supports CPU writes of 1- to 8- bytes to 8-bit or 16-bit wide devices.

Eight and 16-bit devices must not be mapped to cacheable regions. The reason is that the PowerPC has a 32-bytes cache line size. This would equate to a burst of 32 8-bit accesses or 16 16-bit accesses.

The GT-64130 supports cached accesses to 32- and 64-bit device spaces. It supports DMA/PCI writes of 1- to 8-bytes to 8-bit or 16-bit wide devices.

Bursts to 64-bit wide devices must be limited to 4 if BlockRdBuffer bit in CPU Configuration register is set to 1. See Section 4.7 “CPU Interface Read Buffer” on page 50 for more details. In this case, only BAdr[1:0] is used as the device address LSB.

If bursts are not limited, all three BAdr[2:0] bits are used as device address LSB. See Section 12.2 “Devices” on page 142 for more details.

5.8.10 Ready* Support

The Ready* pin is sampled on three occasions:

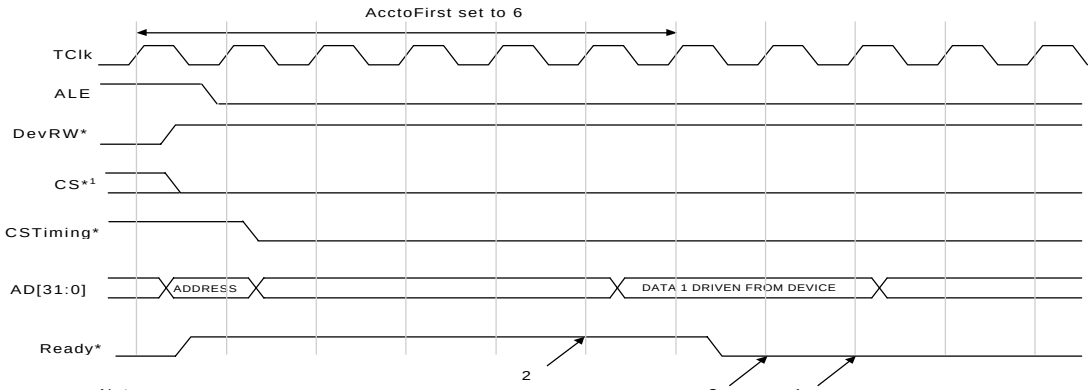
- one clock before the data is sampled to the GT-64130 during AccToFirst (see Figure 25)
- one clock before the data is sampled to the GT-64130 during AccToNext (see Figure 26) phases of read cycles
- on the last rising edge of the WrActive (see below) phase during a write cycle.

During all other phases, Ready* is not sampled by the GT-64130.

If Ready* is not asserted during these clocks, the WrActive, AccToFirst, or AccToNext phases are extended until Ready* is asserted again. The transaction may be indefinitely held off until Ready* is asserted.

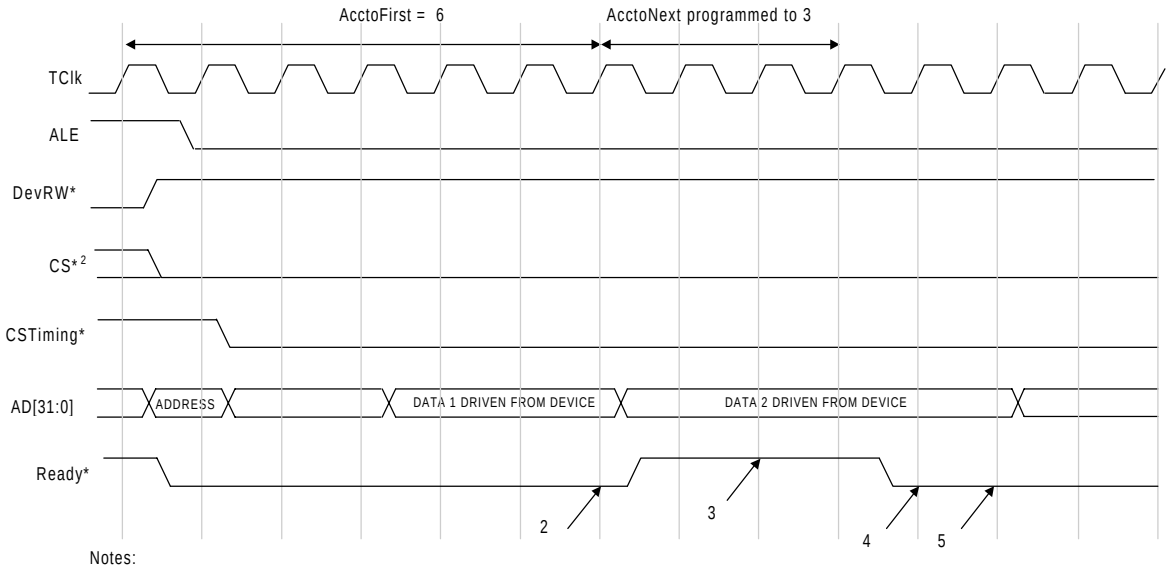
NOTE: While Ready* is de-asserted, SDRAM refresh requests will not be serviced. Therefore, it is important make sure Ready* is not de-asserted indefinitely.

Figure 25: Ready* Extending AccToFirst on Read Cycle



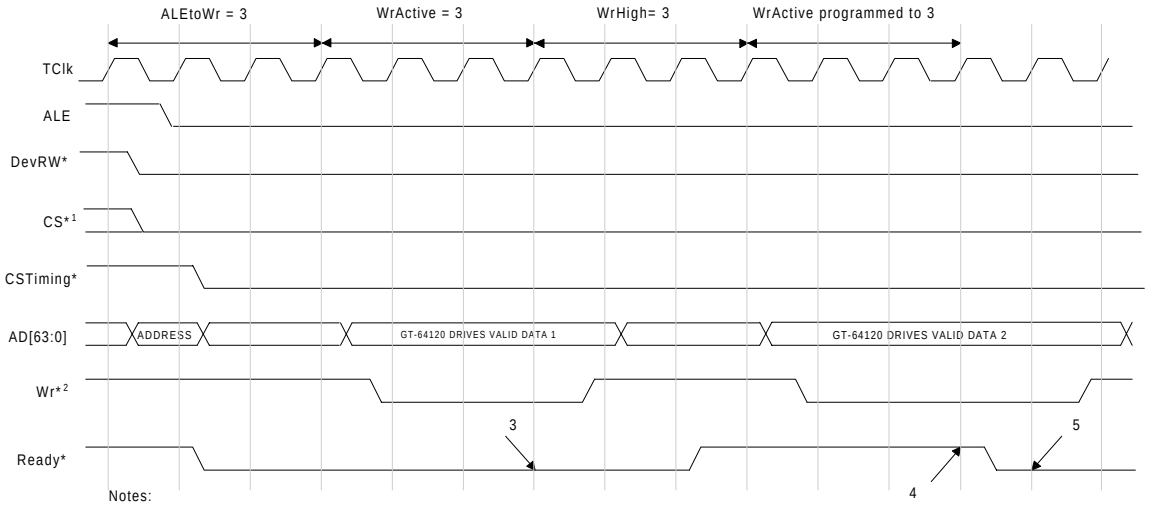
Notes:
1. CS* is driven off the same rising TClk* as ALE. Throughout consecutive transactions to the same device, CS* will remain asserted. This is why CS* must ALWAYS be qualified with CSTiming*.
2. Ready* is sampled as deasserted one clock before data should be sampled according to AcctoFirst. AD[31:0] is NOT sampled by the GT-64130 on the following rising TClk.
3. Ready* is asserted on some following rising TClk.
4. AD[31:0] (DATA 1) is sampled on the following rising TClk of the rising TClk that Ready* was asserted. Effectively, AccToFirst is 8 in this example (instead of the programmed 6).

Figure 26: Ready* Extending AccToNext on Read Cycle



Notes:
1. CS* is driven off the same rising TClk* as ALE. Throughout consecutive transactions to the same device, CS* will remain asserted. This is why CS* must ALWAYS be qualified with CSTiming*.
2. Ready* is sampled as asserted one clock before data should be sampled according to AcctoFirst. DATA 1 is sampled on AD[31:0].
3. Ready* is sampled as deasserted one clock before data should be sampled according to AcctoNext. DATA 2 is NOT sampled.
4. Ready* is asserted on some following rising TClk.
5. AD[31:0] (DATA 2) is sampled on the following rising TClk of the rising TClk that Ready* was asserted. Effectively, AccToNext is 5 (instead of the programmed 3).

Figure 27: Extending WrActive Parameter on Write Cycle



Notes:
1. CS* is driven off the same rising TClk* as ALE. Throughout consecutive transactions to the same device, CS* will remain asserted. This is why CS* must ALWAYS be qualified with CSTiming*.
2. Wr[7:0]* are asserted and deasserted from the falling edge of TClk.
3. Ready* is asserted on last rising TClk of WrActive, therefore, the GT-64130 assumes the device is written to correctly and continues to the next write cycle.
4. Ready* is deasserted on the last rising TClk of WrActive, therefore, the GT-64130 does NOT continue to the next write cycle effectly extending the WrActive parameter. Wr* remains asserted.
5. Ready* is asserted on the next rising TClk, therefore, the GT-64130 assumes the device has been written to correctly and continues to the next cycle (end of write transaction). Effectively, WrActive is 4 (instead of the programmed 3). For clarification, if there was another word to burst, WrHigh would start counting from the rising TClk denoted by 5, not 4.

5.8.11 Parity Support for Devices

There is no dedicated logic in the GT-64130 to support devices parity. If Devices parity checking is required, it can be implemented externally using 511 devices and some logic for interrupt generation.

Still the GT-64130 generates ODD parity on CPU DP lines during CPU read transaction from devices.

5.9 Error Checking and Correcting (ECC)

The GT-64130 ECC logic supports detection and correction of 1-bit data error, detection of 2-bits error, and 3- or 4-bits if reside in the same nibble.

5.9.1 ECC Calculation

Each of the 64 data bits and eight check bits has a unique 8-bit ECC check code, as shown in Table 31. For example, data bit 12 has the check value of 01100001and check bit 5 has the check value of 00100000.

Table 31: ECC Code Matrix

Check Bit	Data Bit	ECC Code Bits								Number of 1s in
		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
	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	3
	52	0	0	0	1	1	1	1	1	5
5		0	0	1	0	0	0	0	0	5
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

Table 31: ECC Code Matrix (Continued)

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

Table 31: ECC Code Matrix (Continued)

Check Bit	Data Bit	ECC Code Bits								Number of 1s in
		7	6	5	4	3	2	1	0	
	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
	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-64130 calculates ECC by taking the even parity of ECC check codes of all data bits that are logic one. For example, if the 6-bit data is 0x45. The binary equivalent is 01000101. From Table 31, 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.

On a write transaction to a 64-bit wide SDRAM with ECC enabled, the GT-64130 calculates the new ECC and writes it to the ECC bank along with the data that is written to the data bank. Since ECC calculation is based on a 64-bit data width, if the write transaction is smaller than 64-bit, the GT-64130 will run read a modify write sequence.

On a read transaction from a 64-bit wide SDRAM with ECC enabled, the GT-64130 reads a 64-bit data and an 8-bit ECC, calculates ECC based on the 64-bit data and compares against the received ECC. The result of this comparison (bitwise XOR between received ECC and calculated ECC) is called syndrome. If the syndrome is 00000000, both the received data and ECC are correct. If the syndrome is any other value, it is assumed 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. If the received by the ECC is 01010101, the resulting syndrome is

00001000. Table 31 shows that this syndrome corresponds to check bit 3. The GT-64130 will not report nor correct this type of error.

If the syndrome contains three or five 1s, it indicates that there is at least one data bit error. For example, if the received data is 0x45, the calculated ECC is 01011101. If the received ECC is 00011110, the resulting syndrome is 01000011. This syndrome includes three 1s and it corresponds to data bit 2 as shown in Table 31. The GT-64130 corrects the data by inverting data bit 2 (the corrected data is 0x41).

If the result syndrome contains two 1s, it indicates that there is a double-bit error. If it contains four ones, it indicates a 4-bit error located in four consecutive bits of a nibble. If it contains three or five ones, but it does not corresponds to any data bit, it indicates a triple-bit error within a nibble. These types of errors cannot be corrected. The GT-64130 reports an error but will not change the data.

NOTE: ECC is not supported in bypass mode. Data must flow through the GT-64130 in order to be checked and corrected.

5.9.2 ECC Error Report

If the GT-64130 identifies an ECC data bit error, it sets MemErr bit in the Interrupt Cause register, an interrupt is asserted if not masked.

In addition, it stores error information in dedicated registers. It stores the 64-bit data in:

- ECC Upper Data and ECC Lower Data registers.
- ECC byte read from memory in ECC from Mem register.
- Calculated ECC byte in ECC Calculated register.

In ECC Error Report register it reports whether it was a single data bit error (implies it was corrected) or a multiple data bits error.

5.10 Programming the ADP lines for other Functions

If ECC is enabled for any SDRAM bank, the ADP lines will function as pins used for reading and writing the ECC byte.

If ECC is not implemented in the system, the ADP lines can be used for other functions. These functions include duplicating the SDRAM control lines (used for load balancing) and duplicating the ALE signal (used for load balancing).

During RESET, certain pins are sampled (either HIGH or LOW) to select these pin functions. See Section 11. “Reset Configuration” on page 139 for more information.

If ECC is not enabled and the ADP lines are not programmed to function as other pins on RESET, ADP[3:0 functions as EOT[3:0]. See Section 8.2.15 “End Of Transfer Enable, EOTE, bit 18” on page 126 for a description. Further, ADP[4] will be used as BankSel[1] and ADP[5] As DAdr[11]. ADP[7:6] is unused.

Table 32 summarizes the functionality of the ADP lines depending on system configuration.

Table 32: ADP[7:0] Pin Functionality

ECC IN SYSTEM	NO ECC IN SYSTEM		
ADP Pin (ECC Enabled for any SDRAM bank)	**ECC Not Enabled for ANY SDRAM bank**	**DMAReq[1]* sampled HIGH on RESET**	**DMAReq[3]* sampled HIGH on RESET**
ADP[0]	EOT[0]		
ADP[1]	EOT[1]		ALE
ADP[2]	EOT[2]		
ADP[3]	EOT[3]	DWr*	
ADP[4]	BankSel[1]		
ADP[5]	DAdr[11]		
ADP[6]		SCAS*	
ADP[7]		SRAS*	

5.11 Memory Controller Restrictions

1. Unless the boot device is 64-bits wide, the boot must be on the even half (bits 31:0 of AD[63:0] bus).
2. When working with an 8- or 16-bit configured bank, a read/write operation cannot exceed 64-bits (8 bytes). Since PCI reads are always prefetchable, PCI access to a 8- or 16-bit wide device is not allowed (unless FRAME* is asserted for a single PClk cycle).
3. When an erroneous address is issued or a burst operation is performed to an 8- or 16-bit device, the GT-64130 forces an interrupt (unless masked). If a sequence of address misses occurs, there will be no other interrupt prior to resetting the appropriate bit in the cause register and no new address will be registered in the Address Decode Error register (0x470) prior to reading it.
4. When the CPU reads from an address which is decoded in the CPU Interface Unit as being a hit for CS[2:0]* or BootCS* and CS[3]* and decoded as a miss in the SDRAM/Device Interface Unit, the cycle completes only if Ready* is asserted (i.e., driven LOW). Although being a result of improper and inconsistent programming of the address space defining registers, the following workarounds exist:
   - •Ready* must always be asserted (LOW) when CSTiming* is inactive (HIGH).
   - •If the Ready* signal is not needed in the system, the Ready* pin must be constantly driven active (LOW).
5. The minimum parameter for TurnOff, AccToFirst, AccToNext, WrActive and WrHigh is 1. The minimum parameter for ALEToWr is 3.
6. SDRAM must be configured to linear wrap-around to support CPU burst reads.
7. If BlReadBuffer bit in CPU configuration register is set to 1, 64-bit SDRAM must be configured to burst length of 4.
8. When using 64/128 Mbit SDRAM, address decode 0 is not allowed.
9. PCI reads from 8- or 16-bit bank must not be prefetchable.

10. Access to SDRAM during SDRAM initialization time after reset results in unpredictable behavior.
11. When SDRAM CAS latency is 3, RAS precharge time (bit 3 of SDRAM parameter register) must be pro-grammed to 1.
12. In order for memory controller to return data of an internal register read, it must have control over the AD bus.
13. When using aggressive prefetch, the upper address allowed to be read from the PCI is last DRAM address minus 0x8.
14. Table 33 describes the limitation of a 32-bit device in the GT-64130.

Table 33: Limitation of a 32-bit Device in the GT-64130

Terminology	• word: 32-bit • aligned: aligned to a word
Limitation	During a read/write cycle of an odd number of words to a 32-bit device from an aligned address or a read/write cycle to a 32-bit device to an unaligned address, an extra read/write will occur to the complementary word of the doubleword address accessed with byte enables not active. This may result in destructive reads and loss of performance while accessing the device.
Examples	1. CPU writes one word to offset 0 of a 32 bit device . Result: A write to offset 0 with Wr* asserted and a write to offset 4 with Wr* not asserted. 2. CPU writes one word to offset 4 of a 32 bit device . Result: A write to offset 0 with Wr* not asserted and a write to offset 4 with Wr* asserted. 3. DMA writes five words to offset 0 of a 32 bit device. Result: Burst of 6 to the device, with the last Wr* not asserted. **NOTE:** If Ready* is used, the GT-64130 must sample it active an even number of times (i.e., once for address 0 and once for address 4, see example 1).
Workaround	1. If the device is always accessed for single word transfers, connect the AD[4:2], latched, to the device's least significant address pins instead of the BAdr[2:0]. This will make sure there are no destructive reads. 2. Access the device always with an even number of words at double word aligned addresses (e.g. 0x0, 0x8, 0x10...). This means that the DMA must have the DatTrans-Limit field set to at least 8-bytes. This will ensure that there are no destructive reads and no loss of performance.

6. PCI INTERFACES

The GT-64130 can be configured to have two 32-bit PCI buses or one 64-bit PCI bus.

The GT-64130 is compliant with revision 2.1 PCI specifications.

Both 3.3V and 5V operations are supported through the use of universal PCI buffers. Support for the I_2O Specification is also included.

6.1 Reset Configuration

At reset, the GT-64130 can be configured to have one 64-bit PCI interface or two 32-bit PCI interfaces. See Section 11. “Reset Configuration” on page 139 for more information.

Each PCI device can be either a master initiating a PCI bus operation or a target responding to a PCI bus operation.

NOTE: When The GT-64130 is configured as two 32-bit PCI devices, both devices are almost identical in behavior and structure. The only difference is that PCI interface 1 (PCI_1) does not support locked cycles and does not have a separate interrupt signal. In the following text, any description that does not specify a particular PCI interface means that the description applies to both interfaces.

6.2 PCI Master Operation

When the CPU or the internal DMA machine initiates a bus cycle to a PCI device, the GT-64130 needs to decode the target address to determine which address space is being accessed.

If the GT-64130 is configured as one 64-bit PCI device, then PCI_0 registers are used for address comparison and remapping. If the GT-64130 is configured as two 32-bit PCI devices, then PCI_0 and PCI_1 are used for comparison.

Based on the match results, either PCI_0 or PCI_1 will become the master on PCI bus and translate the cycle into the appropriate PCI bus cycle. Supported master PCI cycles are:

- 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

Memory Write & Invalidate and Memory Read Line cycles are carried out when the transaction accessing PCI memory space requests a data transfer equal to the PCI cache line size. In case the transfer initiator is the DMA, the requested address must be cache line aligned. In case of write transaction, Memory Write & Invalidate Enable bit in configuration command register must be set. When the PCI cache line size is set equal to 0, the GT-64130 will never issue Memory Write & Invalidate or Memory Read Line cycles.

Memory Read Multiple is carried out when the transaction accessing PCI memory space requests a data transfer greater than the PCI cache line size.

As a master, the GT-64130 does not issue Dual Address cycles or Lock cycles on the PCI.

The PCI posted write buffer in the GT-64130 permits the CPU to complete CPU-to-PCI memory writes even if the PCI bus is busy. Posted data is written to the PCI device when the PCI bus is available.

6.2.1 PCI Master CPU Address Space Decode and Translation

Local masters access the PCI space through the PCI_0/1 Memory 0, PCI_0/1 Memory 1 and PCI_0/1 I/O decoders in the CPU address space.

The CPU accesses that are claimed by these decoders are translated into the appropriate PCI cycles by the appropriate PCI interface (PCI_0 only in case of 64-bit PCI interface). The address seen on the CPU bus is copied directly to the PCI bus (unless the CPU-to-PCI address remapping capability is enabled.) For example, if an access to 0x1200.0040 is programmed to be bridged as a memory read from PCI, then the active PCI address for this cycle will be 0x1200.0040.

6.2.2 PCI Master CPU Byte Swapping

All accesses to PCI space through the CPU can have the data byte order swapped as the data moves through the GT-64130. Byte swapping is controlled via MByteSwap and MWordSwap bits in the PCI Internal Command register (0xc00.) For more information, see Section 14. “Big and Little Endian” on page 146.

6.2.3 PCI Master FIFOs

If the GT-64130 is configured to have one 64-bit PCI device, its master interface includes a FIFO of eight entries, each 64-bits wide (64-bytes total.)

If it is configured to have two 32-bit PCI devices, then each master interface includes its own FIFO of eight entries, each 64-bits wide. During writes to the PCI interface, one master receives write data from the CPU interface or the DMA unit. When the PCI bus is granted, this master’s FIFO delivers the write data to the target on the PCI bus.

Upon receiving the first 32- or 64-bit data from the CPU interface or DMA unit, the PCI master interface requests the PCI bus (if the GT-64130 is not already parked). Once granted, the appropriate write cycle is started on the PCI bus.

During reads, the PCI master interface FIFO receives read data from the PCI bus and delivers it to the CPU interface or the DMA unit. Upon receiving the first word or doubleword from the PCI target, the data is forwarded to the requesting unit (CPU interface or DMA unit).

The GT-64130 supports linear wrap-around ordering during CPU reads. If the original read request address is not aligned to a cache line boundary, then the first 32-bit word (or 64-bit double-word in case of a 64-bit PCI interface) returned to the requesting unit will be delayed until it is received from the PCI target, since reads across the PCI bus are linear.

The GT-64130 internal architecture allows zero wait-state data transfer over the PCI bus (Irdy* continuously asserted) during both master reads and writes.

6.2.4 PCI Master DMA

The GT-64130's internal DMA engines acts as PCI bus masters while transferring data to/from the PCI bus. The DMA engines will only issue PCI memory space read and write cycles. The type of cycle issued follows the same rules as for the CPU. The DMA engines can transfer data between PCI devices using the on-chip DMA FIFOs for temporary storage.

6.2.5 PCI Master RETRY Counter

RETRY's detected by the PCI master interface are normally handled transparently from the point of view of the CPU or DMA engines.

In some rare circumstances, however, a target device may RETRY the GT-64130 excessively or without stopping. Use the Retry Counter to recover from this condition. Every time the number of RETRYs equals the value in the Retry Counter, the GT-64130 aborts the cycle and sends an interrupt to the CPU. If the cycle was a read, undefined data is returned and the ERROR bit is set in the data command.

The Retry Counter can be disabled by setting the Retry count to zero.

Figure 28: PCI Master FIFOs in Single 64-bit Mode

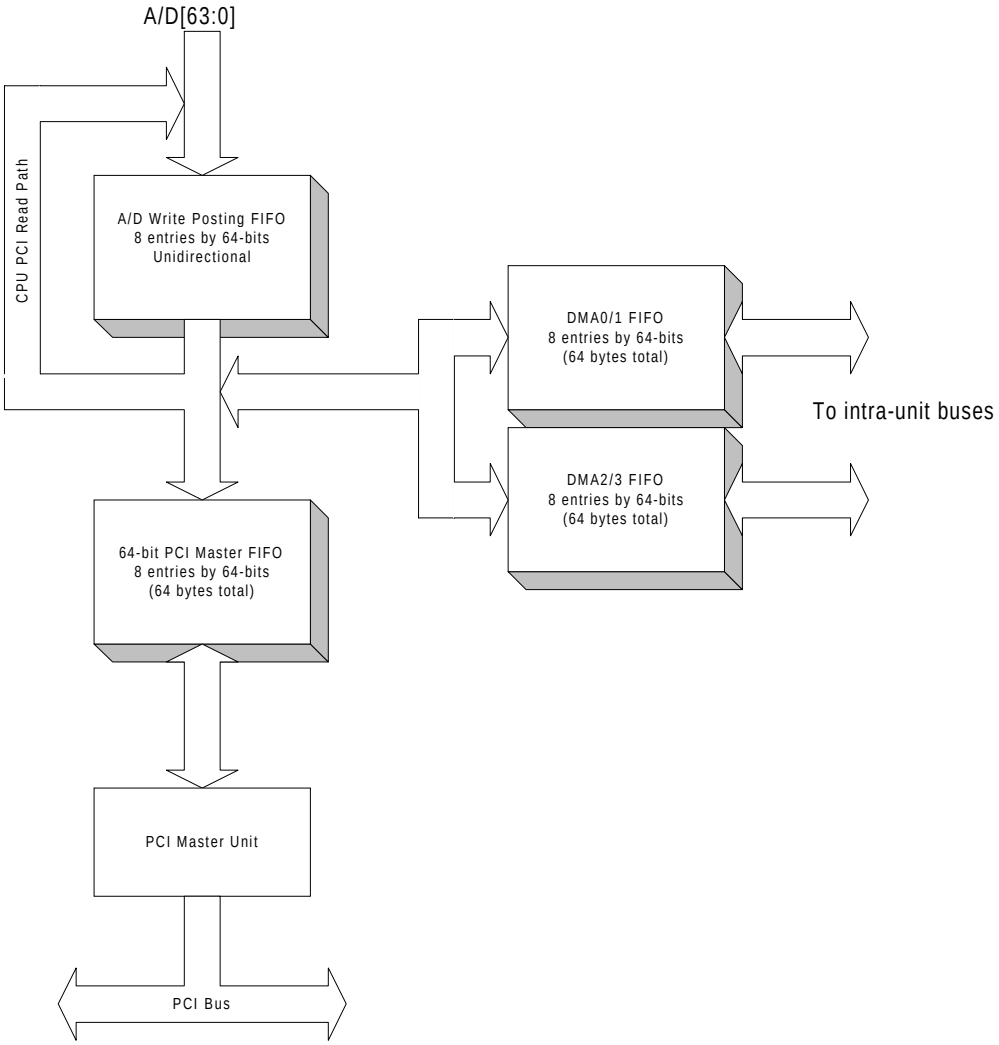
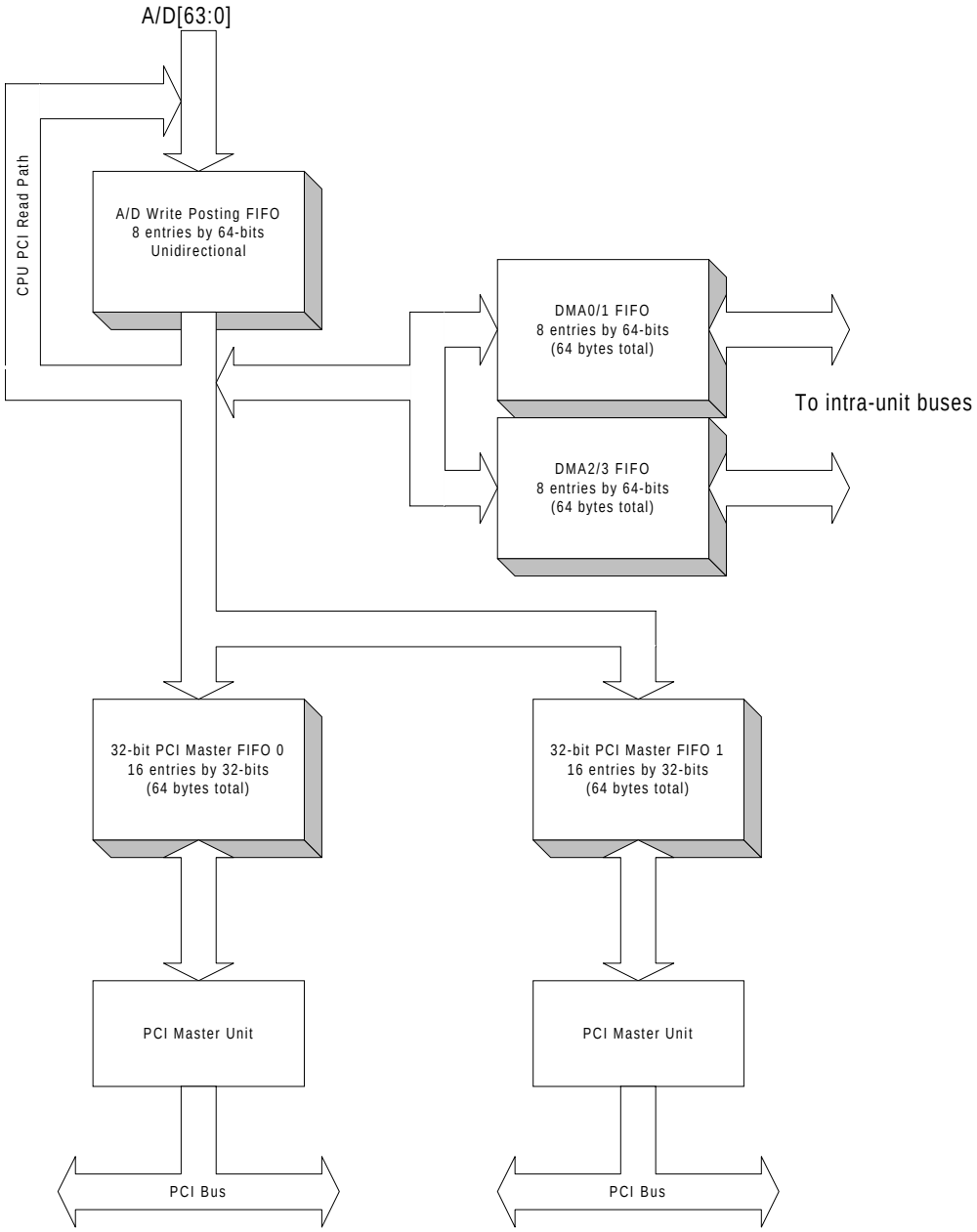


Figure 29: PCI Master FIFOs in Dual 32-bit Mode



6.3 PCI Target Interface

The GT-64130 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
- Locked Read to PCI_0 only
- Locked Write to PCI_0 only

The GT-64130 does not act as a target for Interrupt Acknowledge, Special, and Dual Address cycles. These cycles are ignored.

6.3.1 PCI Target FIFOs

The GT-64130 incorporates dual 64-byte posted write/read prefetch buffers, per PCI interface, to allow full memory (AD) and PCI bus concurrency. The dual FIFOs can operate in a “ping-pong” fashion, each FIFO alternating between filling and draining.

When the GT-64130 is the target of PCI write cycles, data is first written to one of the FIFOs. When the first FIFO fills up (64 bytes), the data is written to the destination from the first FIFO while the second FIFO is filled. This “ping-pong” operation continues as long as data is received from the PCI bus. The GT-64130 will de-assert TRDY for 2 PCI clocks while switching target FIFOs.

Occasionally, the PCI target interface cannot drain the FIFOs (i.e., write to local memory) as fast as data is received. This only happens when access to memory is prevented (possibly by excessive CPU accesses) or when the target memory is particularly slow. In this case, the GT-64130’s PCI target interface will deassert TRDY until one of the FIFOs is empty again and might even issue a DISCONNECT to the PCI bus if reached timeout. See Section 19.15 “PCI Internal” on page 200 for more information.

The target FIFOs are also used to align data bursts that do not start on 64-byte boundaries for more efficient processing by the GT-64130’s memory subsystem. When an incoming burst passes a 64-byte boundary, the target FIFOs switch and the remainder of the burst (now aligned to a 64-byte boundary) fills the new FIFO. When the FIFO switch occurs TRDY will deassert for 2 PCI clocks.

6.3.2 Controlling Burst Length

The GT-64130’s Memory Interface capable of maximum burst of 8 to SDRAM or device, regardless of the device width. This implies that if the device is 64-bits wide, it can burst up to 64-bytes in a single transaction. If the device is 32-bits wide it can burst up to 32-bytes per transaction and so on.

The PCI target is limited to request burst reads and writes from Memory Interface not greater than the bursts supported by this interface. Each PCI target FIFO corresponds to a single Memory Interface transaction. The PCI

target has a “programmable FIFO depth”. MaxBurst register (offset 0xc40) defines per each Base Address Register the FIFO depth. If a bit is set to 1, FIFO depth is 16 (each FIFO entry is 32-bits) it means the maximum burst of 64-bytes on Memory Interface. If a bit is cleared (this is the default after reset), FIFO depth is 8 which means maximum burst of 32 bytes on Memory Interface.

It is the software’s responsibility to program MaxBurst register properly.

NOTE: In order to support PCI bursts to a 32-bit SDRAM or device, MaxBurst must be set to 0. This is also true for a 64-bit SDRAM programed to burst length of 4.

MaxBurst is one of the PCI target interface hooks for performance optimization. Even if the PCI communicates with 64-bit wide memory devices, it is sometimes preferred to keep the bursts shorter to allow CPU or DMA to get more of the memory interface bandwidth.

NOTE: Having the PCI communicate with a 64-bit wide memory device means MaxBurst bits can be programed to 1 allowing the maximum PCI to memory bursts

Figure 30: PCI Target Interface “Ping-Pong” FIFOs

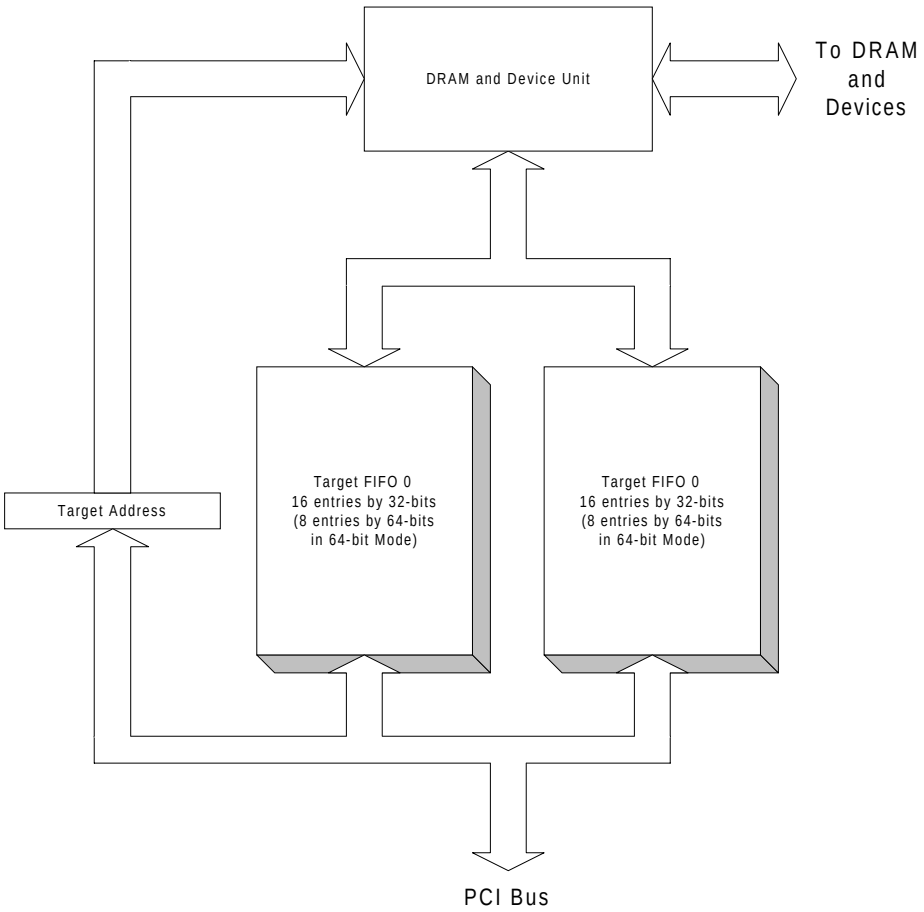
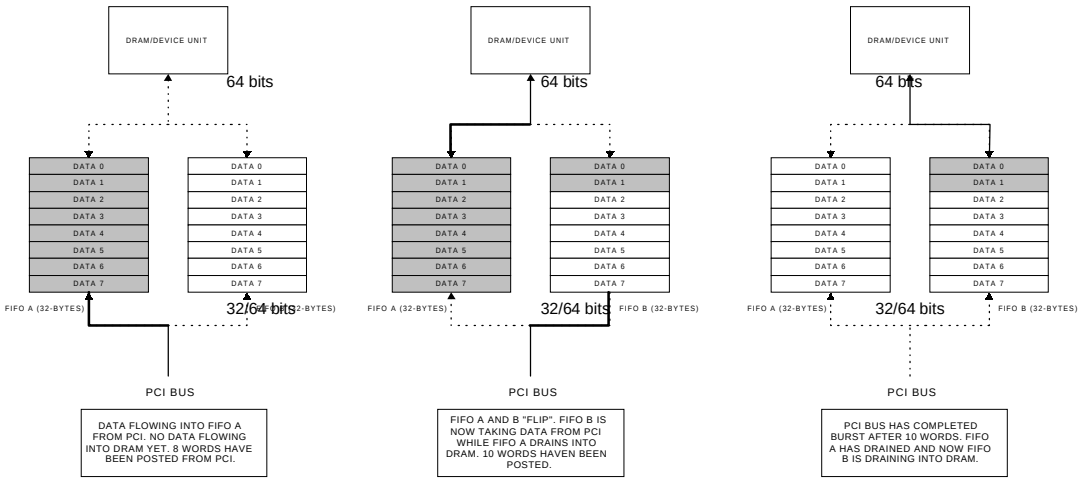


Figure 31: PCI Target Interface FIFOs Operational Example



NOTE: Graphic shows only 8 of the 16 entries in the FIFOs (in 32-bit mode)

6.3.3 PCI Target Read Prefetching

The target FIFOs are also used for read prefetch.

All PCI read commands initiate a prefetch read cycle.

The Device/DRAM Unit fills the target FIFOs as soon as it is determined that the PCI request is longer than a single word (based on how long Frame* is asserted.) The “aggressiveness” of the prefetch is controlled by the type of read command (MR/MRL/MRM) and the state of the bits for each of the read command types in the Prefetch/Max_Burst register.

By default, the only “aggressive” prefetch read is Memory Read Multiple. Both Memory Read and Memory Read Line commands can be marked for aggressive prefetch via the Prefetch/Max_Burst register. With aggressive prefetch, as soon as at least one word is delivered from the FIFO to the PCI bus, the PCI slave will request from the Device/DRAM unit to prefetch into the second FIFO.

NOTE: When using aggressive prefetch, the upper address allowed to be read from the PCI is last DRAM address minus 0x8.

By comparison, a non-aggressive prefetch read cycle data prefetches into only one of the target FIFOs. Data is not fetched into the second FIFO until all data from the first FIFO is delivered to the PCI bus. MR and MRL cycles are by default, non-aggressive prefetch.

NOTE: All three types of read commands will result in prefetch (multiple read cycles) on the Device/DRAM bus unless Frame* is only asserted for a single cycle.

Cycles to internal registers and Configuration cycles are non-postable nor prefetchable. More over, this cycles are always single word. If a PCI master attempts a burst transaction to an internal register, the GT-64130 will disconnect after the first TRDY.

6.3.4 PCI Target Address Space Decode Process

The GT-64130 decodes accesses on the PCI bus for which it may be a target by the values programmed into the Base Address registers (BARs).

There are two sets of BARs for each PCI interface: regular BARs (in PCI Function 0); and, swap BARs (in PCI Function 1). Accesses decoded by the regular BARs are passed without modifying the data to or from the target memory device. Accesses decoded by the swap BARs are passed with to or from the target memory device after converting the endianess of the data (e.g., little-endian to big-endian).

The GT-64130 uses a two stage decode process for accesses through the PCI devices target interface. Once a PCI access is determined to be a "hit" based on the BAR comparison, the address is passed to the Device Unit for sub-decode. For example, PCI_0 Base Address Register 0 in Function 0 (PCI_0 BAR0) decodes non-byte swapped accesses to the SDRAM controlled by either RAS0* or RAS1*. The GT-64130 uses the values programmed into the RAS0 Low and RAS0 High decode registers to determine if the access is to the SDRAM in RAS0 space.

NOTE: The second stage decoders are shared with the CPU.

If the target address of a PCI write transaction "hits" based on the BAR decode, misses in the Device Unit transaction are completed properly on PCI bus but data is not written to memory.

In case of a read transaction, there is no actual read from memory. A read transaction is completed properly on PCI bus but the data driven on the bus is undefined (unless timeout is reached first - RETRY termination). In both cases, MemOut bit in interrupt cause register is set (and interrupt is generated if not masked).

6.3.5 PCI Target Byte Swapping

All PCI accesses to SDRAM or device can have the data byte order swapped as the data moves through the GT-64130. Byte swapping is controlled via the SByteSwap and SWordSwap bits in the PCI Internal Command register (0xc00) and the SSWordSwap bit in the PCI Internal Command register (0xc00), for accesses through swap BARs. For more information, see Section 14. "Big and Little Endian" on page 146.

6.3.6 Tweaking the Performance of the Target Interface

The GT-64130 includes special performance tuning features for the PCI target interface. The PCI_0/1 Timeout0 and Timeout1 registers allow the designer to force the GT-64130 to wait either longer than normal, or shorter than normal, before issuing a RETRY/DISCONNECT.

The Timeout0 value of PCI device0 or device1 sets the number of clocks the device will wait for the first data of an access before issuing a RETRY. The Timeout1 value sets the number of clocks the device will wait between subsequent data phases during an access before issuing a DISCONNECT.

The PCI 2.1 specifications sets the maximum for Timeout0 at 16 clocks and for 8 Timeout1at clocks, respectively. However, in many systems, especially those with long SDRAM latencies, it may be necessary to "bend" these restrictions.[1]

If excessive RETRY/DISCONNECT behavior occurs in the system, try lengthening the Timeout0/1 values. It may be because there is a lot of memory activity due to the CPU or DMA engines and the PCI interface cannot get to the SDRAM within 16 clocks.

1. The PCI specification also states that a master may not enforce target rules. In other words, even if the GT-64130 takes longer than 16 clocks to return the first data, the master must just wait patiently.

The settings in the Prefetch/Max_Burst registers provide another area for performance optimization. Try turning on aggressive prefetch for all read types. This may result in a significant performance improvement, depending on the length and type of read commands.

If, however, the system uses many short reads from PCI, aggressive prefetch will actually waste bandwidth on the Device/DRAM bus on speculative reads that are not used.

6.4 PCI Synchronization Barriers

The GT-64130 considers some cycles to be “synchronization barrier” cycles. In such cycles, the GT-64130 confirms that no posted data remains within the chip at the end of the cycle.

The target “synchronization barrier” cycles are Lock Read (for PCI_0 only) and Configuration Read. If there is no CPUposted data within the GT-64130, the cycle ends normally. If after a retry period there is still posted data, the cycle is retried.

Until the original cycle ends, any other (different address/command) “synchronization barrier” cycles are retried. Lock Read is a “synchronization barrier” cycle which lasts during the entire Lock period (i.e., when the slave of PCI_0 is locked all Configuration Reads are retried.) Also, all cycles addressed to internal registers are retried until Lock ends.

The CPU interface treats I/O Reads to PCI and Configuration Reads as “synchronization barrier” cycles, as well. These reads will be responded to once no posted data remains within the GT-64130.

The GT-64130 provides the CPU with a simpler way to perform synchronization with PCI buffers. Without accessing the PCI bus, the CPU can issue a read command to “PCI_0/1 Sync Barrier Virtual” register to synchronize PCI_0/1 buffers. The response dummy read completes once no posted data remains within the addressed PCI interface.

6.5 PCI Master Configuration

The GT-64130 translates CPU read and write cycles into configuration cycles using PCI configuration mechanism #1 (per the PCI spec). Mechanism #1 defines how to translate the CPU cycles into both PCI configuration cycles on the PCI bus and accesses to the GT-64130’s internal configuration registers.

The GT-64130 includes two registers per PCI device: Configuration Address (at offset 0xcf8 for PCI0 and 0xcf0 for PCI1) and Configuration Data (at offset 0xcfc for PCI0 and 0xcf4 for PC1). The mechanism for accessing configuration registers is to write a value into the Configuration Address register that specifies:

- PCI bus number.
- The device number on that bus.
- The function number within the device.
- The configuration register within that device/function being accessed.

A subsequent read or write to the Configuration Data register (at 0xcfc/0xcf4) causes the GT-64130 to translate that Configuration Address value to the requested cycle on the PCI bus.

If the BusNum field in the Configuration Address register equals 0 but the DevNum field is other than 0, a Type0 access is performed that addresses a device attached to the local PCI bus. If the BusNum field in the Configuration Address register is other than 0, a Type1 access is performed that addresses a device attached to a remote PCI bus.

The GT-64130 performs address stepping for PCI configuration cycles. This allows for the use of the high-order PCI AD signals as IdSel signals through resistive coupling.[1] Table 34 shows DevNum to IdSel mapping.

Table 34: DevNum to IdSel Mapping

DevNum[15:11]	PAD_0[31:11]/PAD_1[31:11]
00001	0 0000 0000 0000 0000 0001
00010	0 0000 0000 0000 0000 0010
00011	0 0000 0000 0000 0000 0100
00100	0 0000 0000 0000 0000 1000
- - -	- - -
10101	1 0000 0000 0000 0000 0000
00000, 10110 - 11111	0 0000 0000 0000 0000 0000

The CPU accesses the GT-64130's internal configuration registers when the fields DevNum and BusNum in the Configuration Address register are equal to 0. The GT-64130 configuration registers are also accessed from the PCI bus when the GT-64130 is a target responding to PCI configuration read and write cycles.

CPU accesses the GT-64130's internal PCI_1 configuration registers through PCI_0 Configuration Address and Data registers (0xcf8,0xcfc). For example, to access GT-64130 PCI_1 Class Code and Rev Id register, the CPU software should write 0x80000088 to PCI_0 Configuration Address register (0xcf8), and then read/write PCI_0 Configuration Data register. The CPU will use PCI_1 Configuration Address register (0xcf0) and Configuration Data register (0xcf4) only in order to generate a configuration read/write transaction on PCI_1 bus.

The CPU interface unit cannot distinguish between an access to the GT-64130 PCI configuration space and an access to an external PCI device configuration space. This is because both are accessed using an access to the GT-64130 internal space (i.e., Configuration Data register). The software engineer must keep this in mind, especially when byte swapping is enabled on the PCI interface. In this case, internal configuration registers and configuration registers in external devices will appear to have a different endianess.

The configuration enable bit (ConfigEn) in the Configuration Address register must be set before the Configuration Data register is read or written. Failure to do this will "lock up" the GT-64130 and require a RESET to recover.

1. "Resistive Coupling" is a fancy way of saying "hook a resistor from ADx to IdSel" on a given device. Look at the Galileo-4PB backplane schematics for examples.

6.5.1 Special Cycles and Interrupt Acknowledge

A Special cycle is generated whenever the Configuration Data register is written to while the Configuration Address register has been previously written with:

- 0x0 for BusNum.
- 0x1f for DevNum.
- 0x7 for FunctNum.
- 0x0 for RegNum.

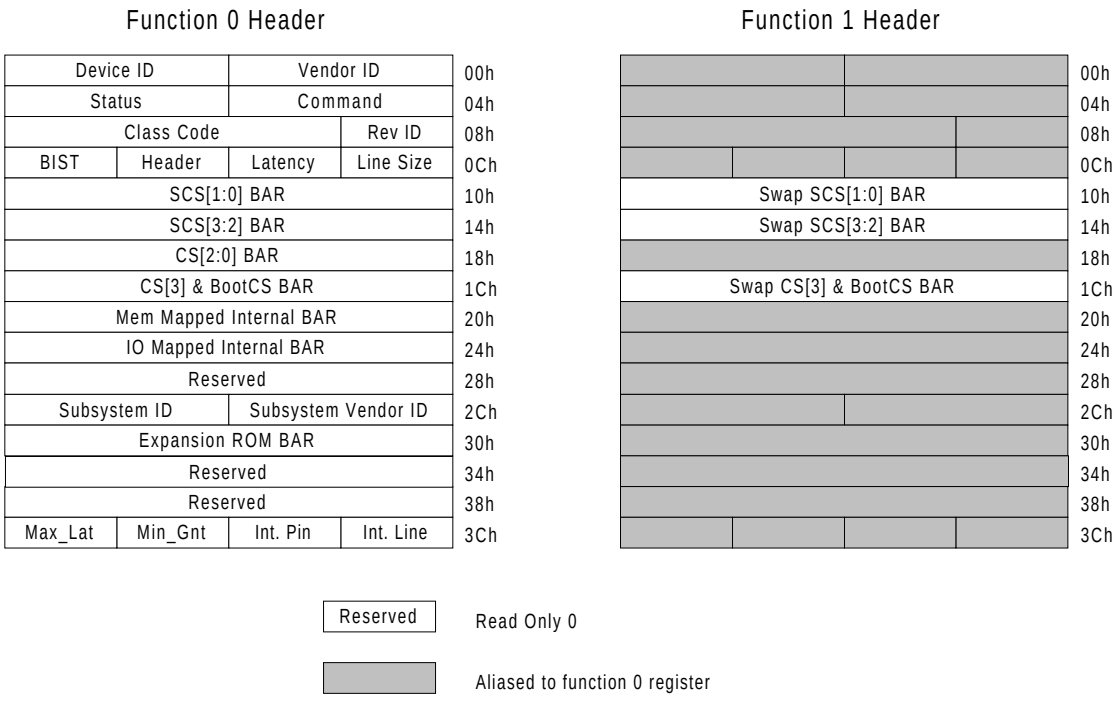
An Interrupt acknowledge cycle is generated whenever the Interrupt Acknowledge (0xc34) register is read.

6.6 Target Configuration and Plug and Play

The GT-64130 includes all of the required plug and play PCI configuration registers. These registers, as well as the GT-64130's internal registers, may be accessed from both the CPU and the PCI bus.

The GT-64130 acts as a two function device when being configured from the PCI bus. The base address registers available in Function 0 are used to decode accesses for which there is no byte swapping. Function 1 is used to decode byte swapped addresses. All other registers are shared between Function 0 and Function 1, as shown in Figure 32.

Figure 32: PCI Configuration Header



6.6.1 Plug and Play Base Address Register Sizing

Systems adhering to the plug and play configuration standard determine the size of a base address register's decode range by writing 0xFFFF.FFFF to the BAR and 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 therefore not part of the address comparison. With this information, the size of the decode region can be determined.[1]

The GT-64130 responds to BAR sizing requests based on the values programmed into the Bank Size Registers. These registers can be loaded automatically after RESET from the system ROM, see Table 35 and Table 36.

6.6.2 PCI Autoconfiguration at RESET

Thirteen PCI registers per PCI interface can be automatically loaded after Rst*.

Enable the autoconfiguration mode by asserting the DAdr[0] pin LOW on Rst*. Any PCI transactions targeted for the GT-64130 are retried while the loading of the PCI configuration registers is in process.

The PCI register values are loaded from the ROM controlled by BootCS* are shown in Table 35 for PCI device0 and in Table 36 for PCI device1.

Table 35: PCI_0 Registers Loaded at RESET

Register	Offset	Boot Device Address
Command	0xc00	0xffffff80
Timeout & Retry Counters	0xc04	0xffffff84
RAS[1:0]* Bank Size	0xc08	0xffffff88
RAS[3:2]* Bank Size	0xc0c	0xffffff8c
CS[2:0]* Bank Size	0xc10	0xffffff90
CS[3]* & Boot CS* Bank Size	0xc14	0xffffff94
Conf_en	0xc3c	0xffffff98
Prefetch/Max_burst	0xc40	0xffffff9c
Device and Vendor ID	0x000	0xffffffa0
Class Code and Revision ID	0x008	0xffffffa4
Cache_line/Latency	0x00c	0xffffffa8
Subsystem Device and Vendor ID	0x02c	0xffffffac
Interrupt Pin and Line	0x03c	0xffffffb0

1. Refer to the PCI specification for more information on the BAR sizing process.

Table 36: PCI_1 Registers Loaded at RESET

Register	Offset	Boot Device Address
Command	0xc80	0xffffffc0
Timeout & Retry Counters	0xc84	0xffffffc4
RAS[1:0]* Bank Size	0xc88	0xffffffc8
RAS[3:2]* Bank Size	0xc8c	0xffffffcc
CS[2:0]* Bank Size	0xc90	0xffffffd0
CS[3]* & Boot CS* Bank Size	0xc94	0xffffffd4
Conf_en	0xcbc	0xffffffd8
Prefetch/Max_burst	0xcc0	0xffffffdc
Device and Vendor ID	0x080	0xfffffe0
Class Code and Revision ID	0x088	0xfffffe4
Cache_line/Latency	0x08c	0xfffffe8
Subsystem Device and Vendor ID	0x0ac	0xfffffec
Interrupt Pin and Line	0x0bc	0xfffffff0

6.6.3 Expansion ROM Functionality

The GT-64130 implements the PCI Expansion ROM as required by PC boot devices. The PCI Expansion ROM functionality is enabled by strapping DAdr[5] low at RESET (see Reset section.)

If the PCI Expansion ROM is disabled, expansion ROM register (offset 0x30 of PCI configuration space) acts as reserved register and returns all 0's when read.

When the expansion ROM is enabled by the pin strapping option, the PCI Expansion ROM BAR appears in the GT-64130's PCI_0 configuration header. However, the Expansion ROM decoder shares functionality with the CS[3]*/BootCS* resource.

In normal operation (i.e., after the BIOS initialization is complete), the Expansion ROM is disabled via bit 0 in the Expansion ROM BAR. During BIOS boot, however, the BIOS "turns on" the Expansion ROM decoder by setting bit 0. When the Expansion ROM decoder is on the following happens in PCI_0:

- The PCI target acts as if the Timeout0 and Timeout1 MSBs are 1 (which means initial value of 0x8f and 0x87 respectively). This is done to allow for the long default access time from 8-bit boot ROMs.
- The Device unit will bypass it's address decoding for all transactions from PCI that are targeted to expansion ROM or SCS[3]*/BootCS*. All of these transactions will assert CS[3] regardless of the actual address.
- The GT-64130 acts this way as long as bit[0] of expansion ROM register is set to 1. It's the BIOS responsibility to clear this bit when it is done executing/probing the Expansion ROM. If the BIOS does not clear bit[0] of expansion ROM BAR, it is still possible to program this bit to 0 from the CPU side.

6.7 PCI Parity Support

The GT-64130 implements all parity features required by PCI spec, including PAR, PERR*, and SERR* generation and checking. These parity features also include PAR64, in the case of 64-bit PCI configuration.

As an initiator, the GT-64130 generates even parity on PAR signal for address phase and data phase of write transactions. It samples PAR on data phase of read transaction. If it detects bad parity and PErrEn bit in Status and Command configuration register is set, it asserts PERR*.

As a target, the GT-64130 generates even parity on PAR signal for data phase of a read transaction. It samples PAR on address phase and data phase of write transaction. If it detects bad parity and PErrEn bit in Status and Command configuration register is set, it asserts PERR*.

The GT-64130 might asserts SERR* if one of the following conditions occur:

- Detects bad address parity as a target
- Detects bad data parity on a write transaction as a master (detects PERR* asserted)
- Detects bad data parity on a read transaction as a master
- Detects ECC error on read from SDRAM or Device
- Performs master abort
- Detects target abort

SERR* will be asserted if SErrEn bit in Status and Command configuration register is set to 1 and if SERR* is not masked through SERR Mask register.

6.8 PCI Bus/Device Bus/CPU Clock Synchronization

The PCI interface is designed to run asynchronously with respect to the AD and CPU buses. The synchronization delay between these two clock domains can be reduced by running the interfaces in synchronized mode. An example would be having the CPU or AD bus running at 66MHz and the PCI bus running at a 33MHz frequency that was derived from the 66MHz.

Latency through the GT-64130 is reduced to a minimum when synchronized mode is selected. The synchronization mode is set via the SyncMode bits in the PCI Internal Command registers (0xc00/0xc80).

NOTE: See Section 5.5.2 “PCI Read Performance from SDRAM” on page 73 for a full description.

The PClk frequency must be lower than TClk by at least 1 MHz. Specifically, since PowerQUICC maximum bus frequency is 50MHz, the PCI clock is limited to 49MHz in case of PowerQUICC configuration. See Section 22.1 “TClk/PClk Restrictions” on page 224 for more information.

6.9 64-bit PCI Configuration

The GT-64130 is configured to work as a 64-bit PCI device if DAdr[2] and FRAME1*/REQ64* pins sampled LOW on reset. See Section 11. “Reset Configuration” on page 139 for more information.

Hold time of REQ64* in respect to RST* rise is 0 as required by PCI spec.

When the GT-64130 is configured to 64-bit PCI, both master and target interfaces are configured to execute 64-bit transactions.

The PCI master interface always attempts to generate 64-bit transactions (asserts REQ64*) except for I/O and configuration transactions or when the required data is less of equal than 64-bit. If the transaction target does not respond with ACK64#, the master completes the transaction as a 32-bit transaction.

PCI target interface always responds with ACK64# to a 64-bit transaction, except for accesses to configuration or internal registers.

NOTE: PClk1 must be tied to PClk0 on 64-bit PCI configuration.

6.10 Retry Enable

In a system that the CPU software programs the GT-64130 internal and configuration registers, it is sometimes required to prevent any PCI access to the device before this programing is completed.

If DAdr[6] pin is sampled low on reset, the GT-64130 PCI target will RETRY any transaction targeted to the GT-64130 space. It will keep this retry mode until Stop Retry bit in CPU configuration register (0x000) is set to 1.

6.11 Locked Cycles

The GT-64130 will lock a cache line (fixed at 32-bytes) in the local memory address space when responding to Lock sequences on the PCI bus. When a cache line is locked, any new PCI accesses to an address within the locked cache line (except of access initiated by the LOCK owner) is terminated with RETRY. Also, every access from CPU or DMA to the locked cache line is held until the LOCK ends.

Although GT-64130 does not support LOCK pin on PCI_1, any access from PCI_1 to a locked cache line is held until the LOCK ends.

6.12 Hot-Plug Support

The GT-64130 is PCI Hot-Plug and CompactPCI Hot Swap capable compliant and supports the following Hot-Plug device requirements:

- All PCI outputs floats when RST# is asserted.
- Each GT-64130 PCI state machine is kept in an idle state while RST# is asserted.
- The GT-64130 PCI interface will not leave it’s idle state until PCI bus is in an IDLE state. If reset is deasserted in the middle of a PCI transaction, the GT-64130 PCI interface will stay in it’s idle state until PCI bus is back in idle.
- The GT-64130 has no assumptions on clock behavior prior to it’s setup to the rising edge of RST#.
- The GT-64130 is tolerant of the 1V precharge voltage during insertion.
- The GT-64130 can be powered from Early VCC.

6.13 PCI Interface Restrictions

6.13.1 Master

1. Latency count, as specified in LatTimer (PCI Configuration Register 0x00c), must not be programmed to less than 6.

6.13.2 Slave

1. The set bits in the Bank Size registers must be sequential.
2. When the slave is locked, in order to prevent a deadlock, all transactions to internal registers (I/O or memory cycles) are not supported (RETRY is issued).
3. Timeout0 value (PCI internal Register 0xc04) must not be programed to less than 2.
4. All PCI reads result in prefetch read from the target device regardless of the BAR prefetch bit value, unless FRAME* is asserted for a single clock cycle.

6.13.3 Master and Slave

1. The PClk frequency must be lower than TClk by at least 1 MHz. See Section 22. “AC Timing” on page 218 for more information.

7. INTELLIGENT I/O (I_2O) STANDARD SUPPORT

The GT-64130 includes hardware support for the Intelligent I/O (I_2O) Standard.

This support includes all of the registered required for implementing the I_2O Messaging Unit as defined in the I_2O Specification. This Messaging Unit is compatible with that found in Intel's i960Rx processors. However, the combination of PowerPC processors and the GT-64130 will deliver much higher performance than the i960RP.

The I_2O hardware support found in the GT-64130 can also provide designers of non-I_2O embedded systems with important benefits. For example, the circular queue support in the Messaging Unit provides a simple, 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 PCI.

NOTE: Even if there is no intention to use the entire I_2O "stack", read through this section in case there may be a clever application of the hardware to the design.

7.1 Overview

The best source for an overview description of I_2O support is in the I_2O specification documentation.

The I_2O specification defines a standard mechanism for passing messages between a host processor , such as a Pentium, and intelligent I/O processors, such as a networking card based on the GT-64130 or a PowerPC processor. This same message passing mechanism may be used to pass messages between peers in a system.

I_2O is defined to be bus independent but in actuality runs over PCI. The basic process is:

1. A master posting a message to a device fetches a pointer from the device. This pointer comes from a defined register in the target device's PCI memory space, one of the I_2O registers.
2. The master assembles the message in the targets memory space and posts the fetched pointer into another register in the target device. The act of posting the pointer generates an interrupt to the target's processor.

The I_2O specification also defines a simpler mechanism for implementing a message passing through doorbell and message registers. The GT-64130 also includes this support.

7.2 I_2O Registers

From the PCI side, the registers used to implement I_2O support reside in the first 128-bytes of the memory region, as defined by SCS[1:0] Base Address Register in PCI function 0 of PCI interface 0[1]. The I_2O registers are accessible from the CPU side through and offset from the CPU Internal Space Base Register.

1. There is no I_2O support on the PCI_1 interface.

7.2.1 I_2O Register Map

Table 37: I_2O Register Map

I_2O Register	PCI SIDE[1]	CPU Side[2]
Inbound Message Register 0	0x10	0x1c10
Inbound Message Register 1	0x14	0x1c14
Outbound Message Register 0	0x18	0x1c18
Outbound Message Register 1	0x1c	0x1c1c
Inbound Doorbell Register	0x20	0x1c20
Inbound Interrupt Cause Register	0x24	0x1c24
Inbound Interrupt Mask Register	0x28	0x1c28
Outbound Doorbell Register	0x2c	0x1c2c
Outbound Interrupt Cause Register	0x30	0x1c30
Outbound Interrupt Mask Register	0x34	0x1c34
Inbound Queue Port Virtual Register	0x40	0x1c40
Outbound Queue Port Virtual Register	0x44	0x1c44
Queue Control Register	0x50	0x1c50
Queue Base Address Register	0x54	0x1c54
Inbound Free Head Pointer Register	0x60	0x1c60
Inbound Free Tail Pointer Register	0x64	0x1c64
Inbound Post Head Pointer Register	0x68	0x1c68
Inbound Post Tail Pointer Register	0x6c	0x1c6c
Outbound Free Head Pointer Register	0x70	0x1c70
Outbound Free Tail Pointer Register	0x74	0x1c74
Outbound Post Head Pointer Register	0x78	0x1c78
Outbound Post Tail Pointer Register	0x7c	0x1c7c
RESERVED	0x80 to 4K	-
SDRAM Ras[1:0]	>4K	-

1. Offset from BAR0 in PCI Memory Space

2. Offset from CPU Internal Space Base Register (location in PowerPC CPU memory space)

7.3 Enabling I_2O Support

The I_2O registers are only visible from the PCI side when I_2O support is enabled via the pin strapping option shown in the RESET Configuration section. When I_2O support is disabled, the locations from BAR0+0x0 to BAR0+0x7F appear as SDRAM.

7.4 Register Map Compatibility with the i960Rx Family

The the GT-64130's register map is compatible with the I_2O Specification. However, some of the registers found in Intel's i960Rx processors are not implemented. This information is shown in Table 38.

Table 38: Register Differences Between GT-64130 and i960Rx

Register Name	GT-64130	i960Rx	Comment
APIC Register Select	Not implemented	Implemented	No APIC support required for the GT-64130. Not a part of the I_2O spec.
APIC Window Select	Not implemented	Implemented	
Index Registers	Not implemented	Implemented	I_2O message passing does not use index registers, so this is not a compatibility issue. Index registers are implemented as normal SDRAM in the GT-64130.

7.5 Message Registers

The GT-64130 uses the message registers to send and receive short messages over the PCI bus, without transferring data into local memory. When written, message registers may cause an interrupt to be generated either to the PowerPC CPU or to the PCI bus. There are two types of message registers:

Table 39: Types of Message Registers

Message Registers	Description
Inbound	Messages sent by an external PCI bus agent and received by the GT-64130.
Outbound	Messages sent by the GT-64130's local CPU and received by an external PCI agent.

The interrupt status for inbound messages is recorded in the Inbound Interrupt Cause register. The interrupt status for outbound messages is recorded in the Outbound Interrupt Cause Register.

7.5.1 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 request will be issued to the PowerPC 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.

7.5.2 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 will be issued to the PCI_0 unit. 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.

7.6 Doorbell Registers

The GT-64130 uses the doorbell registers to request interrupts on both the PCI and CPU buses. There are two types of doorbell registers:

Table 40: Types of Doorbell Registers

Doorbell Registers	Description
Inbound	Set by an external PCI bus agent to request interrupt service from the PowerPC CPU.
Outbound	Set by the GT-64130's local CPU agent to request an interrupt on PCI

7.6.1 Inbound Doorbells

The PCI bus can generate an interrupt request to the local PowerPC processor by setting bits in the Inbound Doorbell Register (IDR). The interrupt may be masked in the IIMR register. 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).

7.6.2 Outbound Doorbells

The local PowerPC 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. 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).

7.7 Circular Queues

The circular queues form the heart of the I_2O message passing mechanism. They are the most powerful part of the messaging unit built into the GT-64130. Four circular queues exist in the Messaging Unit (MU): two inbound; and, two outbound.

7.7.1 Inbound Message Queues

There are two types of inbound message queues.

Table 41: Types of Inbound Message Queues

Message Queue	Description
Inbound Post	Messages from other PCI agents for the PowerPC CPU to process.
Inbound Free	Messages from the PowerPC CPU to PCI agents in response to an incoming message.

The two inbound queues allow external PCI agents to post inbound messages for the local PowerPC CPU in one queue and receive from the PowerPC CPU free messages that are no longer in use. The process is as follows:

1. An external PCI agent posts an inbound message.
2. The PowerPC CPU receives and processes the message.
3. When the processing is complete, the PowerPC CPU places the message back into the inbound free queue so that it may be reused.

7.7.2 Outbound Message Queues

There are two types of outbound message queues.

Table 42: Outbound Message Queues

Message Queue	Description
Outbound Post	Messages from the PowerPC CPU to other PCI agents for processing.
Outbound Free	Messages from the PCI agent to the PowerPC CPU in response to an outgoing message

The two outbound queues allow the PowerPC CPU to post outbound messages for external PCI agents in one queue and receive free messages (no longer in use) returning from external PCI agents. The process is as follows:

1. The PowerPC 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.

7.7.3 Circular Queue Data Storage

Data storage for the circular queues must be allocated in local SDRAM.

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.

The four queues must be the same size and 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 in Table 43.

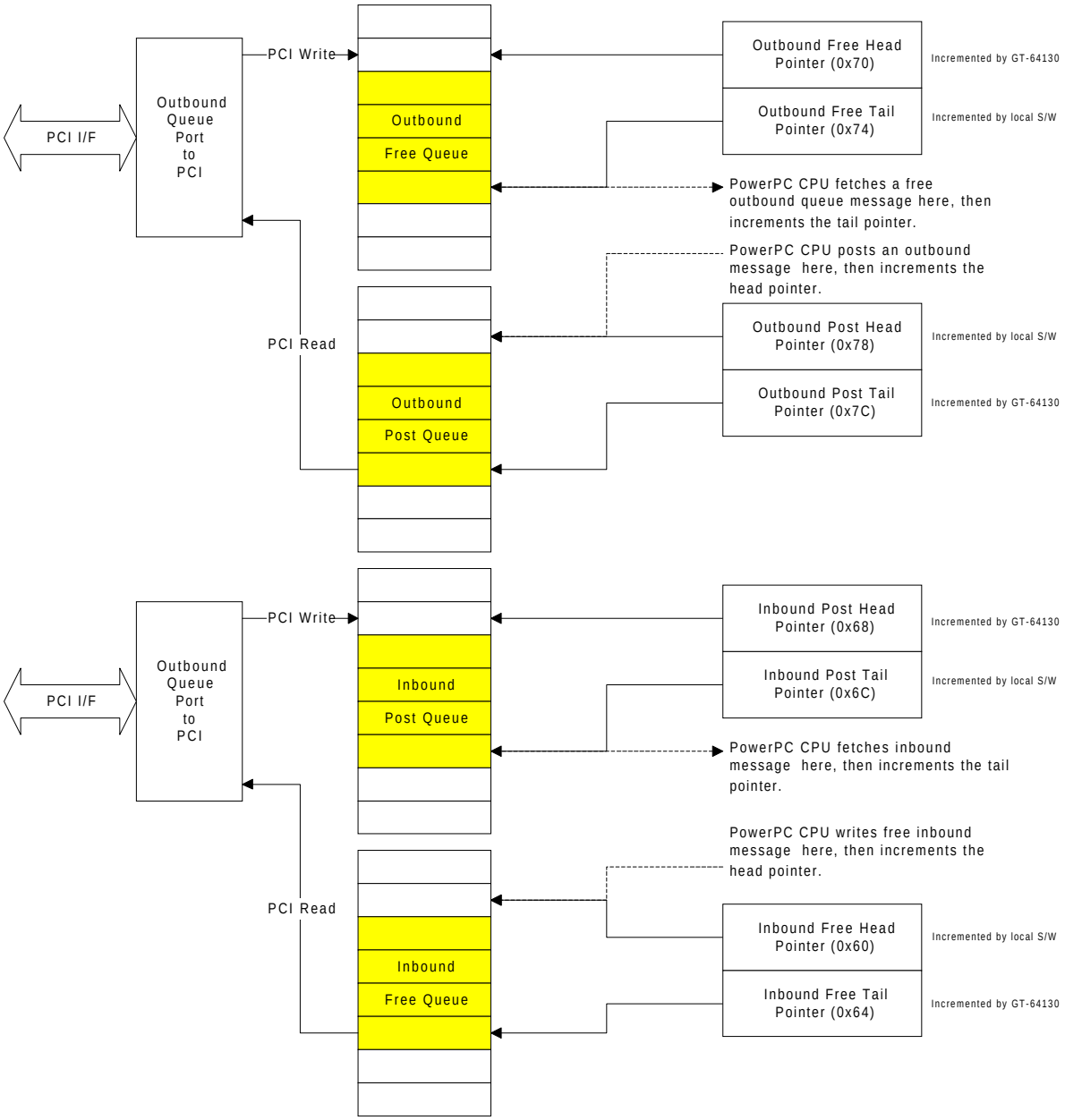
Table 43: Circular Queue Starting Addresses

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-64130 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 queue size.

PCI read/write from queue is always single-word. An attempt to burst from an I_2O queue will result in disconnect after the first data transfer.

Figure 33: I_2O Circular Queue Operation



7.7.4 Inbound/Outbound Queue Port Register Function

The circular queues are accessed by external PCI agents through the Inbound and Outbound Queue Port virtual registers in the I_2O/PCI address space decoded by BAR0. This does not read and write a physical register within the GT-64130. Rather, this reads and writes pointers into the circular queues that are located in SDRAM and controlled by the GT-64130.

Refer to Figure 33 as you read the following sections.

7.7.4.1 Inbound Queue Port Reads and Writes

When Inbound Queue Port (IQP) is written from the PCI, the written data is placed on the Inbound Post Queue. The PCI is posting the message to the local CPU.

An interrupt is generated to the PowerPC CPU when the Inbound Post Queue is written to alert the CPU that a message needs processing. When this register is read from the PCI side, it is returning a free message from the tail of Inbound Free Queue.

7.7.4.2 The Outbound Queue Port

The Outbound Queue Port (OQP) returns data from the tail of the Outbound Post Queue when read from the PCI side. The CPU is returning the next message requiring service by the external PCI agent.

When this register is written from PCI, the data for the write is placed on the Outbound Free Queue; thus returning a free message for reuse by the local PowerPC CPU.

7.7.5 Inbound Post Queue

The Inbound Post Queue holds posted messages from external PCI agents to the PowerPC CPU. The PowerPC CPU fetches the next message process from the queue tail and external agents post new messages to the queue head. The tail pointer is maintained in software by the PowerPC CPU. The head pointer is maintained automatically by the GT-64130 upon posting of a new inbound message.

PCI writes to the Inbound Queue Port are passed to local memory location at QBAR + Inbound Post Head Pointer. After this write completes, the GT-64130 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 PowerPC CPU to indicate the presence of a new message pointer.

From the time the PCI write ends till the data is actually written to DRAM, any new write to Inbound port results in RETRY. If queue is full, a new PCI write to the queue results in RETRY.

Inbound messages are fetched by the PowerPC 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.

7.7.6 Inbound Free Queue

The Inbound Free Queue holds available inbound free messages for external PCI agents to use.

The PowerPC CPU places free messages at the queue head and external agents fetch free messages from the queue tail. The head pointer is maintained in software by the PowerPC CPU. The tail pointer is maintained automatically by the GT-64130 upon a PCI agent fetching a new inbound free message.

PCI reads from the Inbound Queue Port returns the data in the local memory location at QBAR + Inbound Free Tail Pointer. The following conditions apply:

- If the Inbound Free Queue is full (indicated by Head Pointer not equal to Tail Pointer), then 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. This indicates there are no Inbound Message slots available. This results in an error condition.

The PowerPC processor places free message buffers in the Inbound Free Queue by writing the pointer to the location pointed to by the head pointer. It is the processor's responsibility to then increment the head pointer.

7.7.7 Outbound Post Queue

The Outbound Post Queue holds outbound posted messages from the PowerPC CPU to external PCI agents.

The PowerPC 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-64130. The head pointer must be incremented by the local PowerPC 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-64130 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 PowerPC 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.

7.7.8 Outbound Free Queue

The Outbound Free Queue holds available outbound message buffers for the local PowerPC processor to use.

External PCI agents place free messages at the queue head. The PowerPC CPU fetches free message pointers from the queue tail. The tail pointer is maintained in software by the PowerPC CPU. The head pointer is maintained automatically by the GT-64130 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-64130 increments the head pointer.

From the time the PCI write ends till the data is actually written to DRAM, 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 PowerPC CPU. If queue is full, a new PCI write to the queue will result in RETRY.

The PowerPC 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 then increment the tail pointer.

Table 44: 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-64130	PowerPC CPU
Inbound Free		No	No	PowerPC CPU	GT-64130
Outbound Post	Outbound Queue Port	Yes, when queue is not empty	No	PowerPC CPU	GT-64130
Outbound Free		No	Yes, when queue is full	GT-64130	PowerPC CPU

7.8 Index Registers

Index registers are not supported in the GT-64130.

8. DMA CONTROLLERS

The GT-64130 has four independent DMA controllers.

The DMA controllers are used to optimize system performance by moving large amounts of data without significant CPU intervention. Rather than having the CPU read data from one source and write it to another, the DMA controllers can be programmed to automatically transfer data independent of the CPU. This frees the CPU and allows it to continue executing other instructions simultaneous to the movement of data.

Each DMA controller moves data between:

- Peripherals on the SDRAM/Device Controller bus.
- Devices on the PCI buses.
- Peripherals on the SDRAM/Device Controller bus and devices on the PCI buses.

Each DMA transfer uses one of two internal 64-byte FIFOs for moving data. Data is transferred from the source device into an internal FIFO and from the internal FIFO to the destination device.

The DMA controller can be programmed to move up to 64KBytes of data per transaction. The burst length of each transfer of DMA can be set from 1- to 64- bytes. Accesses can be non-aligned both in the source and the destination.

There are two 72-byte FIFOs available for the utilization of the DMA engines. Although the maximum DMA burst size is 64-bytes, the extra 8-bytes in the FIFO are required for non-double word aligned transfers. These two FIFOs allow for concurrency between two DMA transactions. This way, one DMA channel can be reading data from the SDRAM into one FIFO while the second channel is writing data from the other FIFO to a PCI target.

The DMA channels support chained mode of operation. The descriptors are stored in memory and the DMA engine moves the data until the Null Pointer is reached.

Fly-By DMA transfers are also supported. This greatly increases memory bandwidth. Fly-By transfers are permitted to and from a device or to and from SDRAM.

The DMA can be initiated by:

- The CPU writing a register.
- An external request via a DMAReq* pin.
- A timer/counter.

In cases where the transfer needs to be externally terminated, an End of Transfer (EOT[3:0]) pin can be asserted (driven low) for the corresponding DMA channel.

8.1 DMA Channel Registers

Each DMA Channel record consists of the following registers. These registers can be written by the CPU, PCI, or DMA controller in the process of fetching a new record from memory.

8.1.1 Byte Count Register

The byte count register consists of four registers at offsets 0x800 - 0x80c.

This register is programmed with a 16-bit number containing the number of data bytes that this channel must DMA. The maximum number of bytes that the DMA controller can be configured to transfer is 64 Kbytes. This register decrements at the end of every burst of transmitted data from source to destination.

When the byte count register is 0, or the End of Transfer pin is asserted, the DMA transaction is finished or terminated.

8.1.2 Source Address Register

The source address register is at offset 0x810 - 0x81c.

This register is programmed with a 32-bit address. This is the source address for data and can be from the SDRAM/Device controller or from PCI.

This register either increments, decrements, or holds the same value according to bits 3:2 of the Channel Control Register, see Section 8.2.3 SrcDir, bits[3:2] on page 123.

8.1.3 Destination Address Register

The destination address register is at offset 0x820 - 0x82c.

This register is programmed with a 32-bit address. This is the destination address for data and can be to the SDRAM/Device or to PCI.

This register either increments, decrements, or holds the same value according to bits 5:4 of the Channel Control Register, see Section 8.2.4 DestDir, bits[5:4] on page 123.

8.1.4 Pointer to the Next Record Register

The pointer to next record register is at offset 0x830 - 0x83c

This register contains a 32-bit address where the values for the next DMA Channel record can be loaded for chained operation. This can be from the SDRAM/Device controller or from PCI. The byte count, source address, and destination address must be located at sequential addresses.

NOTE: The next record pointer must be 16 bytes aligned. This means bits [3:0] must be set to 0.

This register is only used when the channel is configured for Chained Mode, see Section 8.2.6 ChainMod, bit 9 on page 124. A value of 0 (NULL) for this register indicates this is the last record in the chain, see Section 8.2.6 ChainMod, bit 9 on page 124 for more information about Chained DMA mode.

8.2 DMA Channel Control Register (0x840 - 0x84c)

Each DMA Channel has a unique Control register where certain DMA modes can be programmed. The following sections are the bit descriptions for each field in the control register.

8.2.1 FlyByEn bit

FlyByEn determines whether a DMA transfer will use an internal DMA FIFO to host the data from the source prior to transferring it to the destination.

Setting this bit to 0 causes all DMA transfers to be internal, i.e. the transfer will utilize a DMA FIFO.

Setting this bit to 1 means DMA transfer is in Fly-By mode. Data is transferred from source to destination externally, without using a DMA FIFO.

NOTE: The SDRAM address must always be programmed in the Source Address register when performing a fly-by DMA whether the DMA source or destination is the SDRAM.

8.2.2 R/W bit

R/W indicates whether the DMA transaction with the SDRAM is a read or a write.

NOTES: This bit is meaningful only in Fly-By mode. The FlyByEn bit must be set to 1 for Fly-By mode to be active.

Setting this bit to 0 indicates a Fly-By READ from SDRAM.

Setting this bit to 1 indicates a Fly-By WRITE to SDRAM.

8.2.3 SrcDir, bits[3:2]

The SrcDir field contains information about how the source address for the DMA transfer is handled.

If set to 00 (the default setting) these bits automatically increments the source address.

If set to 01, the source address automatically decrements.

If set to 10, the source address remains constant throughout the DMA burst.

A setting of 11 is reserved.

8.2.4 DestDir, bits[5:4]

The DestDir field contains information about how the destination address for the DMA transfer is handled.

If set to 00 (the default setting), these bits automatically increment the destination address.

If set to 01, the destination address will decrements.

If set to 10, the destination address remains constant throughout the DMA burst.

A setting of 11 is reserved.

8.2.5 DatTransLim, bits[8:6]

The DatTransLim field contains the burst limit of each data transfer. The burst limit can vary from 1 byte to 64 bytes in modulo-2 steps (i.e. 1, 2, 4, 8,..., 64).

8.2.6 ChainMod, bit 9

ChainMod contains whether this channel is set in chained mode or not.

Setting this bit to 0 (the default setting) enables chained mode for the channel. At the completion of a DMA transaction, the Pointer to Next Record register provides the address of a new DMA Record. If this register contains a 0 (NULL) value, this is the last record in the chain.

In chained mode, the channel record's parameters for the current transaction (Byte Count, Source, Destination, and Next Record Pointer) must be initialized in SDRAM/Device memory space or PCI devices. The address of the first record must be initialized by writing it to the channel's Next Record Pointer register. The channel must be enabled by setting ChanEn to 1 (see Section 8.2.9 ChanEn, bit 12 on page 124), and setting FetNexRec (see Section 8.2.10 FetNexRec, bit 13 on page 125) to 1. Setting these two bits can be done during the same write or FetNexRec must be set to 1 on the first write and then the DMA can be initiated by setting ChanEn to 1 on another write. See Figure 34 on page 128 for a diagram of chained mode DMA.

Setting the ChainMod bit to 1 disables chained mode and the Pointer to Next Record register is not loaded at the completion of the DMA transaction.

NOTE: In non-chained mode the Byte Count, Source, and Destination registers must be initialized prior to enabling the channel.

8.2.7 IntMode, bit 10

IntMode controls when this channel asserts the DMAComp (DMA Complete) Interrupt.

Setting this bit to 0 (the default setting) means the channel asserts the DMAComp Interrupt every time the DMA byte count reaches 0.

Setting this bit to 1 with the channel set to chained mode means the DMAComp Interrupt is only asserted when both the Pointer to Next Record Register has a NULL value and Byte Count is 0. If chained mode is disabled, the setting of IntMode is irrelevant and DMAComp Interrupt will be asserted every time the Byte Count reaches 0.

8.2.8 TransMod, bit 11

TransMod indicates whether the channel is set to operate in demand mode or block mode.

Setting this bit to 0 (the default setting) sets the channel to operate in demand mode. DMA accesses are initiated by externally asserting one of the four DMAReq[3:0]* pins or through the timer/counter terminal count.

Setting this bit to 1 sets channel to operate in block mode where DMA accesses are initiated by setting ChanEn, Bit 12. In Block mode, the DMAReq* lines are not sampled.

8.2.9 ChanEn, bit 12

ChanEn enables or disables the DMA channel.

Setting this bit to 0 means the channel is disabled.

Setting this bit to 1 means DMA is initiated based on the current setting loaded in the channel record (i.e. Byte Count, Source Address and Destination Address).

NOTE: The DMA channel can be enabled or disabled via ChanEn if the channel is in Demand or Block Mode.

8.2.10 FetNexRec, bit 13

FetNexRec is only meaningful when chained mode is enabled for the channel.

Setting this bit to 1 forces a fetch of the next record based on the value in the Pointer to Next Record register. This bit can be set even if the current DMA has not yet completed.

This bit is reset to 0 after the fetch of the new record is complete.

If the descriptor equals Null, the CPU should not assert FetNexRec.

8.2.11 DMAActSt, bit 14 (Read Only)

DMAActSt is a read only field that can be polled to see the DMA activity status of the channel.

Setting this bit to 0 means the channel is not active.

Setting this bit to 1 means the channel is active.

In non-chain mode, this bit is deasserted when Byte_Count reaches zero. In chain-mode, this bit is deasserted when pointer to next record is NULL and Byte_Count reaches zero.

This bit is reset if the CPU sets ChanEn to 0 during DMA transfer.

8.2.12 SDA, Source/Destination Alignment, bit 15

The SDA bit determines whether address alignment is done for source or destination.

Setting this bit to 0 means the alignment is towards the source. After the DMA's first read, all reads will be to 64-bit word aligned address.

Setting this bit to 1means the alignment is towards the destination. After the first write, each subsequent writes are with all Byte Enables asserted.

When a device such as a FIFO is the destination of a DMA, it is recommended to use Destination Alignment to avoid destructive writes. Likewise, if a device such as a FIFO is the source of a DMA, it is recommended to use Source Alignment to avoid destructive reads.

NOTE: If both the DMA Source and Destination addresses are aligned, the meaning of this bit is irrelevant.

8.2.13 Mask DMA Requests, MDREQ, bit 16

Some slower devices require extra time in order to deassert a DMA request signal. Use this bit to provide this extra time.

Setting this bit to 1 means the DMA request for the channel is masked for three cycles, starting from the DMA arbitration cycle.

Setting this bit to 0 means there is no masking. This allows devices more time to deassert DMAReq* when DMAAck* qualified with CSTiming* is asserted without the possibility that another premature DMA request entered.

8.2.14 Close Descriptor Enable, CDE, bit 17

A DMA transfer may end or stop, when an EOT signal or FetchNextRec is asserted, with some data remaining in the buffer pointed at by the current descriptor. This bit allows writing the remaining byte count in bits 31:16 of the Byte_Count field of the descriptor (located in memory).

By writing this field, ownership of the descriptor is returned to the CPU. The CPU then calculates the total number of bytes transferred by the DMA channel by subtracting the remaining byte count from the original Byte_Count.

Setting this bit to 1 means that prior to fetching a new descriptor the current descriptor will be closed with remaining byte count.

Setting this bit to 0 means the descriptor will not be closed. The descriptor will be closed only if it is open (i.e. data has been transferred from the source to the destination).

8.2.15 End Of Transfer Enable, EOTE, bit 18

This bit provides devices which have access to a DMA engine to stop a DMA transfer prior to its completion.

In chain mode this causes fetching a new descriptor from memory (if pointer to next record is not equal to NULL) and executing this next DMA. If DMA channel is in non-chain mode, the current DMA transfer is stopped without further action.

Setting this bit to 1means that ending DMA transfers on channel is enabled via the EOT pin.

Setting this pin to 0 means that End Of Transfer (EOT) input is ignored for the channel.

8.2.16 End Of Transfer Interrupt Enable, EOTIE, bit 19

EOTIE enables or disables interrupts due to End Of Transfer (EOT) signal activation.

Setting this bit to 1 means interrupts are enabled, otherwise they are disabled. The interrupt is set immediately after the channel empties it FIFO. If the channel is idle, then the interrupt is set immediately. If the CDE bit is set, then the interrupt is set immediately after the channel closes its current record (in the case that there is an open descriptor).

8.2.17 Abort DMA, ABR, bit 20

It is possible that the CPU may want sometimes to abort a DMA transfer and reprogram the DMA. This bit flushes internal indications which do not get flushed by a mere channel disable.

To do this, the CPU must set ABR bit to 1 and set the En/Dis bit to 0. This bit should be used together with En/Dis if CDE or EOT are enabled and the CPU wants to abort a transfer for reprogramming.

8.2.18 SLP (bits [22:21]), DLP (bits [24:23]), RLP (bits [26:25])

SLP, DLP, and RLP bits are used to redefine the address space of source, destination, or record address locations. They enable overriding the local address space with PCI_0 or PCI1 memory address space.

Table 45: Location of Source Address, SLP

SLP ([22:21])	Function
00	Source address is in local address space.
01	Source address is in PCI_0 memory space.
10	Source address is in PCI_1 memory space.
11	Reserved.

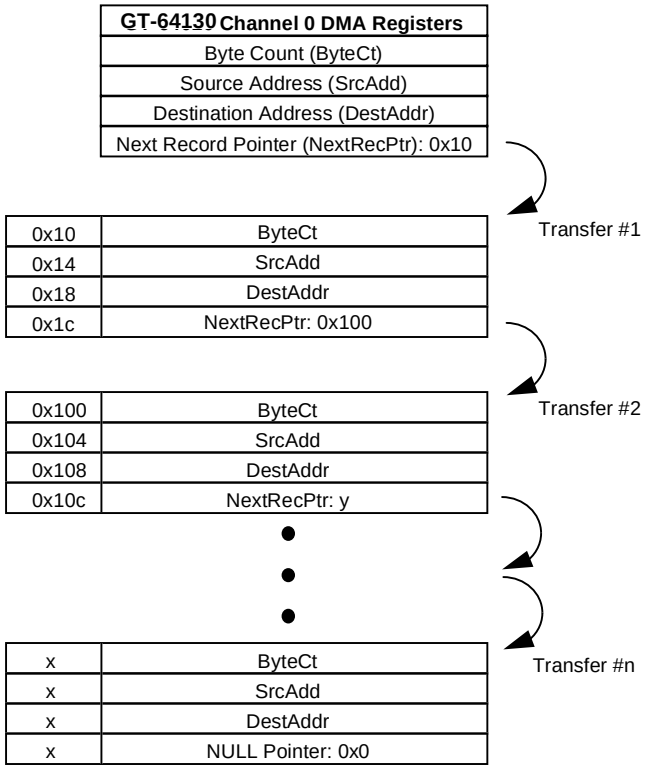
Table 46: Location of Destination Address, DLP

DLP ([24:23])	Function
00	Destination address is in local address space.
01	Destination address is in PCI_0 memory space.
10	Destination address is in PCI_1 memory space.
11	Reserved.

Table 47: Location of Record Address, RLP

RLP ([26:25])	Function
00	Record address is in local address space.
01	Record address is in PCI_0 memory space.
10	Record address is in PCI_1 memory space.
11	Reserved.

Figure 34: Chained Mode DMA



8.3 Restarting a Disabled Channel

In Non-Chained mode, ChanEn must be set to 1.

In Chained mode, the software should find out if the first fetch took place. If it did, only ChanEn should be set to 1. If it did not, the FetNexRec should also be set to 1.

8.4 Reprogramming an Active Channel

To reprogram an active channel, the channel must first be disabled by setting ChanEn to 0. If CDE and/or EOTE are set, then ABR must also be set. It must also be assured that the channel is no longer active. One way to confirm that the channel is no longer active is to poll the channel's DMAActSt.

New DMA parameters must be programmed prior to re-enabling the channel via setting ChanEn to 1.

8.5 Arbitration

The DMA controller has a programmable arbitration scheme between its four channels. These channels are grouped into two groups. One group includes channel 0 and 1. The other group includes channels 2 and 3.

The channels in each group can be programmed so that a selected channel has the higher priority or programmed to have the same priority in round robin.

Similarly, the priority between the two groups can be programmed in a similar way so that a selected group has a higher priority or programmed to have the same priority in round robin.

The priority scheme has additional flexibility with the programmable Priority Option. With the Priority Option, the DMA bandwidth allocation can be divided in a fairer way.

The DMA arbiter control register can be reprogrammed any time regardless of the channels' status (active or not active).

8.6 Current Descriptor Pointer Registers

Each DMA channel has a Current Descriptor Pointer register (CDPTR) associated with it. They are located at offsets 0x870-0x7c.

These registers are read and write registers. However, the CPU should not write these registers. When the NPTR (Next pointer) is written by the CPU, the CDPTR reloads itself with the same value written to NPTR. After processing a descriptor, the DMA channel:

- Updates the current descriptor using CDPTR.
- Saves NPTR into CDPTR.
- Fetches a new descriptor.

This register is used for closing the current descriptor before fetching the next descriptor.

8.7 Design Information

The following sections contain more detailed information about the GT-64130's DMA controllers. The following definitions are used throughout this section:

Table 48: Design Information Terms

Term	Description
Device	A device located on the memory bus mapped to one of the GT-64130's Chip Selects (including BootCS).
PCI Agent	Any device located on the PCI bus.
SDRAM	SDRAM memory located on the memory bus.

8.7.1 DMA in Demand Mode

Demand mode is designed for transferring data between Memory (Device, SDRAM, or PCI agents) and a Device.

This is because the DMAAck* is asserted only when the GT-64130 is accessing a Device. In this mode the transfer initiator, usually a Device, asserts DMAReq* to signal the GT-64130 to begin a new DMA transfer. As an acknowledgment response, the GT-64130 asserts the DMAAck* to signal that the asserted DMAReq* is currently being processed.

In each DMA transfer, the DMA attempts to read the amount of DatTranLim bytes from the source address and write it to the destination address. At the beginning and end of the DMA in the source direction, there may be less than DatTranLim bytes if the address is not aligned or the remaining Byte Count for transfer is smaller then the DatTranLim. In the destination direction, the DMA writes all the data read from the source to the destination. This may take place in two DMA accesses if the destination address is not aligned.

The channel stays active until the Byte Count reaches the terminal count or until the CPU disables the channel.

NOTE: NOTE:if the DMAReq* is always asserted, it is equivalent to transfer data in Block Mode.

*8.7.1.1 Asserting and Deasserting DMAReq**

Assert DMAReq* as long as the transfer initiator has at least DatTranLim of bytes to provide (in case that it is the source) or has space to absorb at least DatTranLim of bytes (in case that it is the destination).

Deassert DMAReq* when the transfer initiator sees that it does not have at least DatTranLim of bytes to provide (i.e. FIFO empty). DMAReq* must also be deasserted by the destination when the it does not have enough space to absorb at least DatTranLim of bytes (i.e. FIFO full) AND DMAAck* is asserted LOW.

*8.7.1.2 Asserting DMAAck**

Assert DMAAck* when the GT-64130 accesses a Device. It is asserted when ALE is asserted.

8.7.1.3 DMAReq Sampling*

The DMAReq* is continually sampled but it influences arbitration only in the DMA arbitration cycle or when all channels are idling. The DMA arbitration cycle is the cycle in which the destination unit inside the GT-64130 acknowledges the last written data from the DMA unit.

8.7.1.4 Transferring Data Examples/recommendations

Table 49: Source and Data Transfer Examples

Source	Destination	Description
SDRAM or PCI	SDRAM or PCI	In this case, it use BlockMode because Memory is typically ready all the time and DMAAck* is NOT asserted. If you choose to work in Demand mode, then DMAAck* should be externally generated (i.e. polling accesses of the GT-64130 to certain addresses).
Device	SDRAM or PCI	The Device transfer initiator asserts a DMAReq* when it has at least DatTranLim number of bytes to give. It should deassert the DMAReq* when no more data is ready and DMAAck* is asserted. Even if another master drives the DMAReq*, the DMAAck* can signal to the master that the GT-64130 is currently accessing the device for read.

Table 49: Source and Data Transfer Examples

Source	Destination	Description
SDRAM or PCI	Device	The Device transfer initiator asserts a DMAReq* when it has enough room for DatTranLim number of bytes to be written to it and DMAAck* is asserted. Even if another master drives the DMAReq*, the DMAAck* can signal the master that the GT is currently accessing the device for write. **NOTE:** In this situation, the GT-64130 asserts the DMAAck* when its accessing the device for write. This is problematic if Dat-TranLimit is smaller or equal to 8 bytes because the DMAAck* is seen outside late. Due to this, deassertion of the DMAReq* is NOT seen in the DMA arbitration cycle. The result is a new transfer may begin although the device does not want to be accessed. • If DatTranLimit is bigger then 8 bytes, there is no problem. In this case, it is recommended to have a device that can accepts bursts. • A solution when using one channel only and DatTranLim is less equal to 8 bytes is to set the MDREQ bit, see Section 8.2.13 Mask DMA Requests, MDREQ, bit 16 on page 125.

8.7.1.5 DMA in Block Mode

In block mode no hand shake signals are used to initiate DMA transfers. The DMA unit completes the transfer once the CPU/PCI has programmed the DMA and enabled it.

8.7.2 Non-Chain Mode

In non-chain mode, the CPU or PCI master initiates the DMA channel parameters (Source, Destination Byte Count, and command Registers).The DMA starts to transfer data after the enable bit in the Command register is set to 1. The DMA remains in an active state until the Byte Count reaches a terminal count or until the channel is disabled.

8.7.3 Chain Mode

In chain mode, the DMA channel parameters (Source, Destination Byte Count, and Pointer to Next Record) are read from records located in Memory, Device, or PCI. The DMA channel stays in the active state until Pointer to Next Record is NULL and the Byte Count reaches the terminal count.

In this mode, an interrupt can be asserted every time the byte count reaches the terminal count or when the Byte Count reaches the terminal count and the Pointer to Next Record is NULL.

8.7.4 Dynamic DMA chaining

Dynamic chaining takes place when DMA records are added to a chain which a DMA controller is actively working on.

The goal is to synchronize between when the GT-64130 reads the last chain record (the NULL pointer) to the time the CPU changes the current last DMA record. Following is an algorithm which provides this synchronization mechanism.

1. Prepare the new record.
2. Change the last record's Pointer to Next Record to point to the new record.
3. Read the DMA control register.

If the DMAActSt bit is 0 (NOT active) {

Update the Pointer to Next Record in the GT-64130 and assert the FetNexRec bit:

```
}
else      {
 read the Pointer to Next Record GT-64130. If it's equal to NULL {
 Mark (by using a flag or something) that in the next DMA chain
 complete interrupt you'll need  to -
[[[[{
  Update the NRP register in the GT-64130 and Write  the  Fetch  Next  Record
}]]]]
 }
   }
```

8.7.5 Fly-by DMA

8.7.5.1 Background

FlyBy is a way to move data directly from the source of the data to its destination. While the source drives the data onto the data bus, the destination immediately latches it into its buffers/memory.

The data does not pass through the GT-64130. This saves at least half of the AD bus bandwidth.

FlyBy cycles are requested by the DMA channel, and controlled by the memory unit.

8.7.5.2 FlyBy in the GT-64130

During FlyBy, the GT-64130 supplies the full information to the SDRAM involved in the transaction. This includes all control signals (SRAS*, SCAS*, SWr*, SCS*, and SDQM) and the address lines (DAdr, BA0, and BA1).

For the Device, the GT-64130 supplies the control signals (CSTiming*, DmaAck*, and DevRdWr*) that support the same waveforms as if the device were not working in FlyBy mode. This means that the DmaAck* and DevRdWr* are latched using ALE.

The external logic must form the address to the device, if needed, and the correct write signals to the device, if the device is the destination.

It is important to understand that the FlyBy cycle is totally compliant with the SDRAM waveforms and the device has to keep up with its speed.

NOTE: The device parameter register is ignored during FlyBy transactions.

8.7.5.3 Programming the GT-64130 for FlyBy

The DMA control register has two bits for FlyBy indications.

Table 50: FlyBy Bits

Bit	Description
FlyByEn	DMA accesses are FlyBy, i.e., the data does not enter the internal FIFO.
FlyByDir	Determines if the device is the source or the destination. This bit affects the DevRdWr* orientation towards the device. **NOTE:** The SDRAM address must be written to the source register of the DMA channel, regardless of whether the SDRAM is the source or the destination.

8.7.5.4 Design Considerations

The Device (FIFOs, FPGA) must be fast enough to maintain read/write at SDRAM burst speed.

For RAS to CAS, a setting of 3 DMAReq* must be deasserted within three TClk cycles following CSTiming* assertion.

For RAS to CAS, a setting of 2 DMAReq* should be deasserted within two TClk cycles following CSTiming* assertion.

8.7.5.5 Determining CS During FlyBy

Bits [29:26,22] in the DeviceX (Bank0, Bank1, Bank2, Bank3, and Boot) parameter registers are used to determine which Chip Select (CS) is activated during FlyBy.

- DMA channel 0 uses bits [29:26, 22] of Bank0 parameter register.
- DMA channel 1 uses bits [29:26, 22] of Bank1.
- DMA channel 2 uses bits [29:26, 22] of Bank2.
- DMA channel 3 uses bits [29:26, 22] of Bank3.

The interpretation of bits [29:26,22] is as follows:

- Bit 22 Low - BootCS* will be the active CS.
- Bit 26 Low - CS0* will be the active CS.
- Bit 27 Low - CS1* will be the active CS.
- Bit 28 Low - CS2* will be the active CS.
- Bit 29 Low - CS3* will be the active CS.

NOTE: As a consequence of the nature of this mechanism, only one bit within bits [29:26, 22] should be Low.

By default, bit 26 is low. This means CS0 is the active CS unless otherwise programmed.

8.8 Initiating a DMA From a Timer/Counter

Each channel can be programmed to have the DMAReq* sourced from the external DMAReq* pin or from the associated timer/counter. For example: DMA channel 0 can only be enabled by Timer/Counter 0; DMA channel 1 can only be enabled by Timer/Counter 1; and, so on.

If bit 28 in the DMA command register is set to 1 and when the timer/counter reaches the terminal count, an internal DMAReq* is set and a new DMA transfer is initiated.

When this bit is set to 1, DMAReq* is ignored.

When this bit is set to 0, DMAs are initiated by asserting DMAReq*. Initiating DMA from timer/counter is enabled only in demand mode.

8.9 DMA Restrictions

1. To reprogram a channel after it has been enabled, it must first be checked that the DMAActSt bit is set to NOT ACTIVE, see Section 8.2.11 DMAActSt, bit 14 (Read Only) on page 125. If working in CDE mode or EOTE mode, the ABR bit must be set to 1, along with the En/Dis bit.
2. When Source or Destination address is decremented, both addresses should be double-word-aligned (A2, A1, and A0 must be 0) and Byte Count must be a multiple of 8 (this applies for burst limits greater than 8 bytes).
3. Burst reads of more than 2 double-words from PCI devices will have all Byte Enables (BEs) active. This implies that DMA read from PCI I/O space must be aligned or with burst limit no bigger that 8 bytes, in order to avoid PCI spec violation. The PCI spec defines the correlation between two LSB address bits and byte enables on I/O transactions.
4. When using the address hold option in the source direction (see Section 8.2.3 SrcDir, bits[3:2] on page 123) and SDA bit is set to 1, the source and destination addresses must be double-word aligned.
5. When using the address hold option in the destination direction, both source and destination addresses should be double-word aligned.
6. Records addresses (NPTR) must be a multiple of 16 bytes. If the descriptors are stored in a device in chained mode, the device must be 32 or 64 bit. If the descriptors are stored in SDRAM or in PCI memory, there are no restrictions on the width of the resource. All descriptors must reside on word-aligned addresses.
7. No support for destination alignment (SDA has no affect) when burst limit is 1, 2, or 4 bytes.
8. When DMA accesses an unmapped address (see Section 3.5 Address Space Decoding Errors on page 34), it will result in unpredicted behavior and the DMA channel might need to be stopped by clearing the activate bit of the channel control register.
9. If the DMA state machine has a pending read of the next descriptor AND the descriptor is located in the PCI space, any PCI accesses to the DMA registers are stopped.
10. If the PCI master accessing the GT–64120A DMA registers and the DMA descriptors resides on the far side of a PCI-to-PCI bridge, a lock-up may occur because the PCI requires that all writes must occur before any reads can take place across a PCI-to-PCI bridge.

8.9.1 Fly-by Mode DMA Restrictions

1. The device and SDRAM must be the same width, 32 or 64 bits.
2. In Fly-By transfers, Byte_Count must be a multiple of 8 and source address must be word aligned.
3. The SDRAM CAS latency must be programmed to 2.
4. SRASPrchg in all SDRAM parameter registers must be programed to 1(e.g. 3 cycles).

9. TIMER/COUNTERS

There are three 24-bit wide and one 32-bit wide timer/counters on the GT-64130. Each one can be selected to operate as a timer or as a counter. The timer/counter's count frequency is equal to the TCLk.

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. For example, note that even though the registers are programmed to an initial value of 0 the counters will read 0xffffff. In order to reprogram a timer/counter, it must first be disabled, loaded with a new value, and then enabled as either a counter or timer.

NOTE: The GT-64130 has no external input pins, for enable/disable, or any output timer pins.

10. INTERRUPT CONTROLLER

The GT-64130 includes an interrupt controller that routes internal interrupt requests to both the CPU and the PCI bus. The interrupt controller ORs all internal interrupt sources and asserts an interrupt to the CPU or to the PCI when one or more internal interrupts are asserted.

10.1 Interrupt Cause Registers

There are two interrupt cause registers which can be checked to detect the occurrence of certain events.

One cause register consists of the interrupts asserted for events caused by the GT-64130 and by PCI_0. This register is located at offset 0xc18.

The second cause register (also known as the high cause register) consists of the interrupts asserted for events caused by PCI_1. This register is located at offset 0xc1c.

NOTE: If the device is configured for PCI_0 only, the high cause register is reserved and is not used.

Connect the Interrupt* pin directly to one of the CPU input interrupt lines. The Int0* pin is used for the devices on PCI_0. If a source asserts an interrupt, its respective bit in the Cause register will be set. All of the bits from the cause register(s) are OR-ed together. The result is outputted to the Interrupt* line for the CPU and on Int0* for PCI devices.

Interrupt* signals the CPU to read the interrupt cause register and run a particular service routine depending on the interrupt bit that was set. Int0* serves the same purpose but signals PCI_0 devices. There is not a separate interrupt for PCI_1 but Interrupt* and Int0* can asserted due to events which occur on PCI_1.

The interrupt is acknowledged by the CPU or by the PCI bus by resetting its bit in the cause register (writing zero to the specific bit and one to all other bits). Exceptions the above is the CPUInt ([25:21] and PCIInt ([29:26])) which are used by the PCI to generate interrupt to the CPU and vice versa. These are set by writing zero from the interrupt originating side and cleared by writing zero from the interrupt destination side.

See Section 19.15 “PCI Internal” on page 200 for more information about the event which causes each interrupt cause register bit to assert.

10.2 Interrupt Mask Registers

There are two interrupt mask registers for both the Interrupt* pin and the Int0* pin (4 mask registers total).

These registers allow interrupt bits to be masked off from asserting an interrupt to the CPU or to PCI. For Interrupt*, the mask register for the main cause register is located at 0xc1c and at 0xc9c for the high cause register. For Int0*, the mask register for the main cause register is located at 0xc24 and at 0xc38 for the high cause register. A zero in a mask register bit will mask the interrupt from asserting an interrupt. A one in a mask register bit will allow this interrupt bit to be included in the OR-ing of the cause register bits for Interrupt* or Int0* output pin.

10.3 Interrupt Summaries

IntSum in the Interrupt Cause register is the logical OR of bits[29:1], regardless of the Mask registers' values. This is used for polling if any interrupt occurred within the GT-64130. Therefore, bit[0] of all of the mask registers is set to 0. If both PCI_0 and PCI_1 are used, IntSum is the logical OR of both the cause register bits as well as the HIGH cause register bits.

CPU IntSum in the main cause register is the logical OR of bits[29:26,20:1], masked by bits[29:26,20:1] of the corresponding mask register. Therefore, bits[25:21] of the particular mask register, being non-relevant to interrupts directed to the CPU, are read-only 0 and bits[31:30], being summaries, are read-only 0.

PCIIntSum in the main cause register is the logical OR of bits[25:1], masked by bits[25:1] of the corresponding mask register. Therefore, bits[29:26] of the PCI Mask register, being non-relevant to interrupts directed to the PCI, are read-only 0 and bits[31:30], being summaries, are read-only 0.

10.4 Interrupt Select Registers

When the GT-64130 is configured for both PCI_0 and PCI_1, two interrupt select registers are used to optimize interrupt routines. One select register exists for the CPU interrupt (Interrupt*) at 0xc70 and another register exists for the PCI_0 interrupt (Int0*) at 0xc74. If the device is configured for PCI_0 only, both of these registers are reserved and are not used.

Instead of checking both the main cause register and the high cause register when interrupted, the CPU or PCI device has the option to read the appropriate select register. The select register contains the cause register bits of either the main or the high cause register, depending on which register has interrupt bits set. Bit 30 of the select register indicates which cause register (main or high) is the source of the interrupt. Bit 29:1 contains the aliased interrupt bits of the appropriate cause register.

If both the main and the high cause registers have interrupt bits set, bit 31is set to 1. Bit 30 indicates that the select register is aliassing the main cause register (set to 0).

11. RESET CONFIGURATION

The GT-64130 must acquire some knowledge about the system before it is configured by the software. Special modes of operation are sampled on RESET in order to enable the GT-64130 to function as required. Certain pins must be pulled up to VCC or down to GND (4.7K Ohm recommended) externally to accomplish this.

NOTES: Except Frame1*/Req64* which requires zero hold time in respect to RESET rise (as defined in PCI spec).

The following configuration pins are continuously sampled from Rst* assertion until 3 TClk cycles after Rst* is deasserted.

Rst* must be de-asserted for at least 10 PClk cycles before any CPU transactions are generated.

Table 51: Reset Configuration

Pin	Configuration Function
Frame1*/Req64*,DAdr[2]:	PCI Bus Configuration
00- 01- 10- 11-	Only PCI_0 is used as 64-bit. Only PCI_0 is used as 32-bit, PCI_1 is NOT used. RESERVED Both PCI_0 and PCI_1 are used as 32-bit
Interrupt*:	CPU Data Endianess
0- 1-	Big endian Little endian
Ready*, CSTiming*	Multi-GT-64130 Address ID
00- 01- 10- 11-	GT responds to A/DL[5-6]= 00 GT responds to A/DL[5-6]= 01 GT responds to A/DL[5-6]= 10 GT responds to A/DL[5-6]= 11 **NOTE:** Boot GT-64130 must be programmed to 11.
BankSel[0]:	PCI Class Code Default Select
0- 1-	Memory Controller (0x580) Host Bridge (0x600)
DAdr[10]:	Multiple GT-64130 Support
0- 1-	Not supported Supported
DMAReq[0],DAdr[9]:	CPU Select
00- 01- 10- 11-	PowerQUICC 603e, 32-bit bus PowerPC (64-bit bus) with external QuickSwitch. PowerPC (64-bit bus) with no external QuickSwitch.

Table 51: Reset Configuration (Continued)

Pin	Configuration Function
DAdr[8]:	I_2O Support
0- 1-	Enable Disable
DAdr[7]:	UMA Support
0- 1-	Enable Disable
DAdr[6]:	Programming Conditional PCI Retry
0- 1-	Enable Disable
DAdr[5]:	Expansion ROM Enable
0- 1-	Enable Disable
DAdr[4:3]:	Device Boot Bus Width (Controlled by BootCS*)
00- 01- 10- 11-	8 bits 16 bits 32 bits 64 bits
DAdr[1]:	Reserved
0-	Must pull low.
DAdr[0]:	Autoload
0- 1-	Enable Disable
DMAReq[3]*	Duplicate ALE
0- 1-	Do not Duplicate ALE. Duplicate ALE output on ADP[1] (no parity in system).
DMAReq[1]*	Duplicate SDRAM Control Signals
0- 1-	Do not Duplicate SRAS*, SCAS* and DWr*. Duplicate SRAS*, SCAS* and DWr* on ADP[7], ADP[6] and ADP[3] (no parity in system).

12. Connecting the Memory Controller to SDRAM and Devices

In order to connect the memory (SDRAM and Devices) correctly, it is necessary to follow the pin connections listed in the tables below.

12.1 SDRAM

The GT-64130 supports both 64-bit and 32-bit SDRAM.

Table 52: 64-bit SDRAM

Connection	Connect...	To...
SDRAM Address	DAdr[11:0]	SDRAM address pins
SDRAM Bank Address	BankSel[1:0]	Bank Address pins
SDRAM Data	AD[63:0]	D[63:0], SDRAM Data pins
SDRAM Control Pins	SRAS* SCAS* DWr*	SDRAM Row Address Strobe SDRAM Column Address Strobe Write Enable
Chip Selects	SCS[0]* SCS[1]* SCS[2]* SCS[3]*	Chip Select, Bank 0 Chip Select, Bank 1 Chip Select, Bank 2 Chip Select, Bank 3
Byte Enables	SDQM[0]* SDQM[1]* SDQM[2]* SDQM[3]* SDQM[4]* SDQM[5]* SDQM[6]* SDQM[7]*	D[7:0] Byte Enable D[15:8] Byte Enable D[23:16] Byte Enable D[31:24] Byte Enable D[39:32] Byte Enable D[47:40] Byte Enable D[55:48] Byte Enable D[63:56] Byte Enable
ECC Bits	ADP[7:0]	D[63:0] ECC Byte
Clock	Same Clock Output used for TClk.	Clock Input

Table 53: 32-bit SDRAM

Connection	Connect...	To...
SDRAM Address	DAdr[11:0]	SDRAM address pins
SDRAM Bank Address	BankSel[1:0]	Bank Address pins
SDRAM Even Data SDRAM Odd Data	AD[31:0] AD[63:32]	Even D[31:0] SDRAM Data pins Odd D[31:0] SDRAM Data pins
SDRAM Control Pins	SRAS* SCAS* DWr*	SDRAM Row Address Strobe SDRAM Column Address Strobe Write Enable

Table 53: 32-bit SDRAM (Continued)

Connection	Connect...	To...
Chip Selects	SCS[0]* SCS[1]* SCS[2]* SCS[3]*	Chip Select, Bank 0 Chip Select, Bank 1 Chip Select, Bank 2 Chip Select, Bank 3
Byte Enables	SDQM[0]* SDQM[1]* SDQM[2]* SDQM[3]* SDQM[4]* SDQM[5]* SDQM[6]* SDQM[7]*	Even D[7:0] Byte Enable Even D[15:8] Byte Enable Even D[23:16] Byte Enable Even D[31:24] Byte Enable Odd D[7:0] Byte Enable Odd D[15:8] Byte Enable Odd D[23:16] Byte Enable Odd D[31:24] Byte Enable
ECC Bits	Not supported for 32-bit SDRAM.	
Clock	Same Clock Output used for TClk.	Clock Input

12.2 Devices

The GT-64130 supports 64, 32, 16 and 8-bit devices.

Table 54: 64-bit Devices

Connection	Connect...	To...
Device Address (no burst limit)	BAdr[2:0] AD[31:6] ALE Latch Outputs	To LSB address bits of the device Address Latch Inputs Address LE Device Address Bits [28:3]
Device Address (burst limited to 4)	BAdr[1:0] AD[31:5] ALE Latch Outputs	To LSB address bits of the device Address Latch Inputs Address LE Device Address Bits [28:2]
Device Data	AD[63:0]	Device Data Pins [63:0]
Device Control Pins	ALE AD[41:32] Control latch bit[41] output Control latch bit[40] output Control latch bit[39:36] outputs Control latch bit[35:32] outputs	Control latch LE Control Latch Inputs Becomes DevRW* Becomes BootCS* Becomes CS[3:0]* Becomes DMAAck[3:0]*

Table 54: 64-bit Devices (Continued)

Connection	Connect...	To...
Write Strobes	Wr[0]* Wr[1]* Wr[2]* Wr[3]* Wr[4]* Wr[5]* Wr[6]* Wr[7]*	D[7:0] Write Strobe D[15:8] Write Strobe D[23:16] Write Strobe D[31:24] Write Strobe D[39:32] Write Strobe D[47:40] Write Strobe D[55:48] Write Strobe D[63:56] Write Strobe
ECC Bits	Not supported for Devices.	

Table 55: 32-bit Devices

Connection	Connect...	To...
Device Address	BAdr[2:0] AD[31:5] ALE Latch Outputs	To LSB address bits of the device Address Latch Inputs Address LE Device Address Bits [29:3]
Device Data	AD[31:0] AD[63:32]	Device Even Data Pins [31:0] Device Odd Data Pins[31:0]
Device Control Pins	ALE AD[41:32] Control latch bit[41] output Control latch bit[40] output Control latch bit[39:36] outputs Control latch bit[35:32] outputs	Control latch LE Control Latch Inputs Becomes DevRW* Becomes BootCS* Becomes CS[3:0]* Becomes DMAAck[3:0]*
Write Strobes	Wr[0]* Wr[1]* Wr[2]* Wr[3]* Wr[4]* Wr[5]* Wr[6]* Wr[7]*	Even D[7:0] Write Strobe Even D[15:8] Write Strobe Even D[23:16] Write Strobe Even D[31:24] Write Strobe Odd D[7:0] Write Strobe Odd D[15:8] Write Strobe Odd D[23:16] Write Strobe Odd D[31:24] Write Strobe
ECC Bits	Not supported for Devices.	

Table 56: 16-bit Devices

Connection	Connect...	To...
Device Address	BAdr[2:0] AD[31:4] ALE Latch Outputs	To LSB address bits of the device Address Latch Inputs Address LE Device Address Bits [30:3]
Device Data	AD[15:0] AD[47:32]	Device Even Data Pins [15:0] Device Odd Data Pins[15:0]
Device Control Pins	ALE AD[41:32] Control latch bit[41] output Control latch bit[40] output Control latch bit[39:36] outputs Control latch bit[35:32] outputs	Control latch LE Control Latch Inputs Becomes DevRW* Becomes BootCS* Becomes CS[3:0]* Becomes DMAAck[3:0]*
Write Strobes	Wr[0]* Wr[1]* Wr[4]* Wr[5]*	Even D[7:0] Write Strobe Even D[15:8] Write Strobe Odd D[7:0] Write Strobe Odd D[15:8] Write Strobe
ECC Bits	Not supported for Devices.	

Table 57: 8-bit Devices

Connection	Connect...	To...
Device Address	BAdr[2:0] AD[31:3] ALE Latch Outputs	To LSB address bits of the device Address Latch Inputs Address LE Device Address Bits 31:3]
Device Data	AD[7:0] AD[39:32]	Device Even Data Pins [7:0] Device Odd Data Pins[7:0]
Device Control Pins	ALE AD[41:32] Control latch bit[41] output Control latch bit[40] output Control latch bit[39:36] outputs Control latch bit[35:32] outputs	Control latch LE Control Latch Inputs Becomes DevRW* Becomes BootCS* Becomes CS[3:0]* Becomes DMAAck[3:0]*
Write Strobes	Wr[0]* Wr[4]*	Even D[7:0] Write Strobe Odd D[7:0] Write Strobe
ECC Bits	Not supported for Devices.	

13. JTAG Application Notes

The GT-64130 supports test mode operation through it's JTAG interface.

The GT-64130 JTAG interface does not include JTRST* pin. When activating test mode, JTMS must be set to 1 for at least five JTCK clock cycles, in order for the JTAG state machine to return it's idle state. When running in test mode, the Rst* pin must be set to 1.

To place GT-64130 in the functional mode, JTAG interface must be disabled. Disable JTAG by configuring the following pins:

- JTCK must be pulled LOW through a 4.7 KOhm resistor.
- JTMS must be pulled HIGH through a 4.7 KOhm resistor.
- JTDI must be pulled HIGH through a 4.7 KOhm resistor.
- JTDO must be left UNCONNECTED.

14. BIG AND LITTLE ENDIAN

14.1 Endian Background

NOTE: For a description of big and little endian and how it is used in Galileo Technology system controllers, go to Endianess Explained! (http://www.GalileoT.com/library/syslib.htm) on the Galileo Technology Website.

There are three bits in the GT-64130 which control byte swapping. One bit is located in the CPU Interface Configuration Register (0x000) bit 12. The other two bits are in PCI Internal Command register (0xc00) bits 0 and 16.

All bits are given the same value as sampled at Rst* via pullup or pulldown on Interrupt* pin. All bits can also be programmed after reset is de-asserted.

If all bits are set to 1, the GT-64130 assumes Little-endian data format and no byte swapping is done within the device.

In addition, there are three WORD-SWAP bits in GT-64130 which controls 32-bit word swap on access to/from PCI.

- Bit 10 controls PCI master interface word swap
- Bit 11 controls PCI target interface word swap when accessed through non-swap BARs
- Bit 12 controls PCI target interface word swap when accessed through swap BARs.

Since the PCI bus is 32-bits wide and the GT-64130 data path is 64-bits wide, byte swap is not good enough in case of working in a BIG endian PCI bus configuration. In this case, these three bits are used for endianess compensation.

The nomenclature for this section is shown in Table 58.

Table 58: Nomenclature Used in Endian Description

Name	Definition
W, Word	32-bits of data.
DW, Double Word	64-bits of data.
Even Address	Address of which A[2] == 0. In Little-endian format, this address points to the least significant W of a DW. In Big-endian format, this address points to the most significant W of a DW.
Odd Address	Address of which A[2] == 1. In Little-endian format, this address points to the most significant W of a DW. In Big-endian format, this address points to the least significant W of a DW.
Even Word	Least significant W of a DW.
Odd Word	Most significant W of a DW.

14.1.1 Bit 12 of the CPU Interface Configuration Register

Bit 12 of the CPU Interface Configuration register (0x000) affects the following:

Table 59: Endianness Settings for Bit 12 of the CPU Interface Configuration Register (0x000)

Setting	Description
1 (Little-endian mode)	No byte swapping within the CPU Interface unit on any data transfer.
0 (Big-endian mode)	Byte swapping of data transfers to or from the GT-64130 internal registers (including Configuration Data register, 0xcfc). There is no byte swapping of data transfers of which the source or target is external.

14.1.2 Bits 0 and 16 of the PCI Internal Command Register

Bit 0 of the PCI Internal Command register (0xc00) controls byte swapping of the GT-64130 PCI master interface. Bit 16 controls byte swapping of the GT-64130 PCI target interface. The bits affect the following:

Table 60: Settings for Bits 0 (0xc00) and 16 of the PCI Internal Command Register

Setting	Description
1 (No byte swapping)	No byte swapping within the PCI Interface unit of any data transfer.
0 (Byte swapping)	Byte swapping of data transfers of which the source or target is external. There is no byte swapping of data transfers to or from the PCI Interface unit's internal registers.

14.1.3 Bits 10-12 of the PCI Internal Command Register

Table 61: Settings for Bits 0 (0xc00) and 16 of the PCI Internal Command Register

Setting	Description
0	No word swapping.
1	Word swapping of data transfers of which the source/target is external. No word swapping of data transfers to or from PCI Interface unit's internal registers.

14.2 Configuring a System for Big and Little Endian

Table 62 shows all combinations of the resources and swapping bits with sample data.

- CPU bit = Bit 12 of the CPU Interface Configuration register (0x000).
- PCI byte swap bit = Bits 0 and 16 of the PCI Internal Command register (0xc00).
- PCI word swap bit = Bits 10-12 of the PCI Internal Command register (0xc00).

NOTE: The sample data is 0x04030201.

Table 62: Configuring for Big and Little Endian

	Swap Bits (CPU bit : PCI byte swap bit : PCI word swap bit)			
Resource	**110**	**001**	**010**	**101**
Internal Registers (CPU access)	04030201	01020304	01020304	04030201
Internal Registers (PCI access)	04030201	04030201	04030201	04030201
Internal PCI Configuration Registers (CPU access)	04030201	01020304	01020304	04030201
Internal PCI Configuration Registers (PCI access)	04030201	04030201	04030201	04030201
External PCI Configuration Registers	04030201	04030201	01020304	01020304
Memory (DRAM and Devices) (CPU access)	04030201	04030201	04030201	04030201
Memory (DRAM and Devices) (PCI access)	04030201	01020304	04030201	01020304
CPU to PCI (Except external PCI Configuration Registers)	04030201	01020304	04030201	01020304

15. USING THE GT-64130 WITHOUT THE CPU INTERFACE

Table 63 lists the pins that must be strapped when the GT-64130 is used without the CPU interface (i.e. PCI Memory Controller only).

NOTE: Rst* and TClk must always be connected in any system. Each pin must be strapped with a separate resistor unless otherwise noted.

Table 63: CPU-less Pin Strapping

Pin	Strapping[1]
TS*	Pulled up to VCC through a resistor.
L2BR*	Pulled up to VCC through a resistor.
ARTRY*	Pulled up to VCC through a resistor.
A/DL[0-31][2]	Pulled up to VCC through a resistor.
DH[0-31][3]	Pulled up to VCC through a resistor.
TT[1]	Pulled up to VCC through a resistor.
TT[3]	Pulled up to VCC through a resistor.
TBST*	Pulled up to VCC through a resistor.
TSIZ[0-2]	Pulled up to VCC through a resistor.
AACK*	No Connect.
TA*	No Connect.
QsAEn*	No Connect.
BG*	No Connect.
DBG*	No Connect.
L2BG*	No Connect.
L2DBG*	No Connect.
Interrupt*	Sampled at RESET. See Section 11. “Reset Configuration” on page 139.

1. Galileo Technology recommends using 4.7KOhm resistors.
2. A/DL[0-31] can be pulled up through a single resistor instead of 32 separate resistors.
3. DH[0-31] can be pulled up through a single resistor instead of 32 separate resistors.

16. USING THE GT–64130 IN DIFFERENT PCI CONFIGURATIONS

The PCI interface of the GT–64130 can be used in four different modes:

- No PCI.
- PCI_0 as 32-bit PCI
- PCI_0 and PCI_1 as 32-bit PCI.
- PCI_0 as 64-bit PCI.

Table 64 lists what must be done with the pins when the GT–64130 is used without any PCI interface.

NOTE: Rst* must always be connected. Most pins must be strapped HIGH or LOW through a resistor (Galileo Technology recommends to use 4.7 KOhm resistors).

Table 64: No PCI

Pin	Pin Usage
VREF0	VREF0.
PClk0	Pulled up to VCC through a resistor.
DevSel0*	Pulled up to VCC through a resistor.
Stop0*	Pulled up to VCC through a resistor.
Par0	No Connect.
PErr0*	Pulled up to VCC through a resistor.
Frame0*	Pulled up to VCC through a resistor.
IRdy0*	Pulled up to VCC through a resistor.
TRdy0*	Pulled up to VCC through a resistor.
Gnt0*	Pulled down to GND through a resistor.
IdSel0	Pulled down to GND through a resistor.
SErr0*	Pulled up to VCC through a resistor.
Req0*	No Connect.
Int0*	Pulled up to VCC through a resistor.
Lock0*	Pulled up to VCC through a resistor.
PAD0[31:0]	No Connect.
CBE0[3:0]*	No Connect.
VREF1	Tie directly to 3V or 5V power plane.
PClk1	Pulled up to VCC through a resistor.
DevSel1*/Ack64*	Pulled up to VCC through a resistor.
Stop1*	Pulled up to VCC through a resistor.
Par1/Par64	No Connect.

Table 64: No PCI (Continued)

Pin	Pin Usage
PErr1*	Pulled up to VCC through a resistor.
Frame1*/Req64*	Pulled up to VCC through a resistor.
IRdy1*	Pulled up to VCC through a resistor.
TRdy1*	Pulled up to VCC through a resistor.
Gnt1*	Pulled down to GND through a resistor.
IdSel1	Pulled down to GND through a resistor.
SErr1*	Pulled up to VCC through a resistor.
Req1*	No Connect.
PAD1[31:0]/PAD0[63:32]	No Connect.
CBE1[3:0]*/CBE0[7:4]*	No Connect.

Table 65 lists what must be done with the pins when the GT–64130 is used with PCI_0 as a 32-bit PCI interface only (i.e. no PCI_1).

NOTE: When the GT–64130 is configured to a single 32-bit PCI_0 interface, the GT–64130 drives all PCI_1 interface signals to a random value. Therefore, there is no need to put pull ups or pull downs on PCI_1 interface signals.

Table 65: PCI_0 as 32-bit PCI Only

Pin	Pin Usage
VREF0	VREF0
PClk0	PClk0
DevSel0*	DevSel0*
Stop0*	Stop0*
Par0	Par0
PErr0*	PErr0*
Frame0*	Frame0*
IRdy0*	IRdy0*
TRdy0*	TRdy0*
Gnt0*	Gnt0*
IdSel0	IdSel0
SErr0*	SErr0*
Req0*	Req0*

Table 65: PCI_0 as 32-bit PCI Only (Continued)

Pin	Pin Usage
Int0*	Int0*
Lock0*	Lock0*
PAD0[31:0]	PAD0[31:0]
CBE0[3:0]*	CBE0[3:0]*
VREF1	Tie directly to 3V or 5V power plane.
PClk1	Pulled up to VCC through a resistor or tied directly to PClk0.
DevSel1*/Ack64*	Pulled up to Vcc through a resistor.
Stop1*	Pulled up to Vcc through a resistor.
Par1/Par64	No Connect.
PErr1*	Pulled up to Vcc through a resistor.
Frame1*/Req64*	Pulled up to Vcc through a resistor.
IRdy1*	Pulled up to Vcc through a resistor.
TRdy1*	Pulled up to Vcc through a resistor.
Gnt1*	Pulled down to GND through a resistor; or, pulled up to Vcc through a resistor.
IdSel1	Pulled down to GND through a resistor.
SErr1*	Pulled up to Vcc through a resistor.
Req1*	No Connect.
PAD1[31:0]/PAD0[63:32]	No Connect.
CBE1[3:0]*/CBE0[7:4]*	No Connect.

Table 66 lists the what should be done with the pins when the GT–64130 is used with PCI_0 as a 32-bit PCI interface and PCI_1 as a 32-bit PCI interface.

Table 66: PCI_0 as 32-bit PCI and PCI_1 as 32-bit PCI

Pin	Pin Usage
VREF0	VREF0
PClk0	PClk0
DevSel0*	DevSel0*
Stop0*	Stop0*
Par0	Par0

Table 66: PCI_0 as 32-bit PCI and PCI_1 as 32-bit PCI (Continued)

Pin	Pin Usage
PErr0*	PErr0*
Frame0*	Frame0*
IRdy0*	IRdy0*
TRdy0*	TRdy0*
Gnt0*	Gnt0*
IdSel0	IdSel0
SErr0*	SErr0*
Req0*	Req0*
Int0*	Int0*
Lock0*	Lock0*
PAD0[31:0]	PAD0[31:0]
CBE0[3:0]*	CBE0[3:0]*
VREF1	VREF1
PClk1	PClk1
DevSel1*/Ack64*	DevSel1*
Stop1*	Stop1*
Par1/Par64	Par1
PErr1*	PErr1*
Frame1*/Req64*	Frame1*
IRdy1*	IRdy1*
TRdy1*	TRdy1*
Gnt1*	Gnt1*
IdSel1	IdSel1
SErr1*	SErr1*
Req1*	Req1*
PAD1[31:0]/PAD0[63:32]	PAD1[31:0]
CBE1[3:0]*/CBE0[7:4]*	CBE1[3:0]*

Table 67 lists the what should be done with the pins when the GT–64130 is used with PCI_0 as a 64-bit PCI interface only (i.e. no PCI_1).

Table 67: PCI_0 as 64-bit PCI only

Pin	Pin Usage
VREF0	VREF0
PClk0	PClk0
DevSel0*	DevSel0*
Stop0*	Stop0*
Par0	Par0
PErr0*	PErr0*
Frame0*	Frame0*
IRdy0*	IRdy0*
TRdy0*	TRdy0*
Gnt0*	Gnt0*
IdSel0	IdSel0
SErr0*	SErr0*
Req0*	Req0*
Int0*	Int0*
Lock0*	Lock0*
PAD0[31:0]	PAD0[31:0]
CBE0[3:0]*	CBE0[3:0]*
VREF1	Tie directly to 3V or 5V power plane (same as VREF0)
PClk1	Tie directly to PClk0
DevSel1*/Ack64*	Ack64*
Stop1*	Pulled up to VCC through a resistor
Par1/Par64	Par64
PErr1*	Pulled up to VCC through a resistor
Frame1*/Req64*	Req64*
IRdy1*	Pulled up to VCC through a resistor
TRdy1*	Pulled up to VCC through a resistor
Gnt1*	Pulled up to VCC through a resistor
IdSel1	Pulled down to GND through a resistor

Table 67: PCI_0 as 64-bit PCI only (Continued)

Pin	Pin Usage
SErr1*	Pulled up to VCC through a resistor.
Req1*	No Connect
PAD1[31:0]/PAD0[63:32]	PAD0[63:32]
CBE1[3:0]*/CBE0[7:4]*	CBE0[7:4]*

17. USING THE GT-64130 IN MPC860 CONFIGURATION

To use the GT-64130 in an MPC860 configuration, configure the pins as follows:

Table 68: MPC860 Pin Configuration

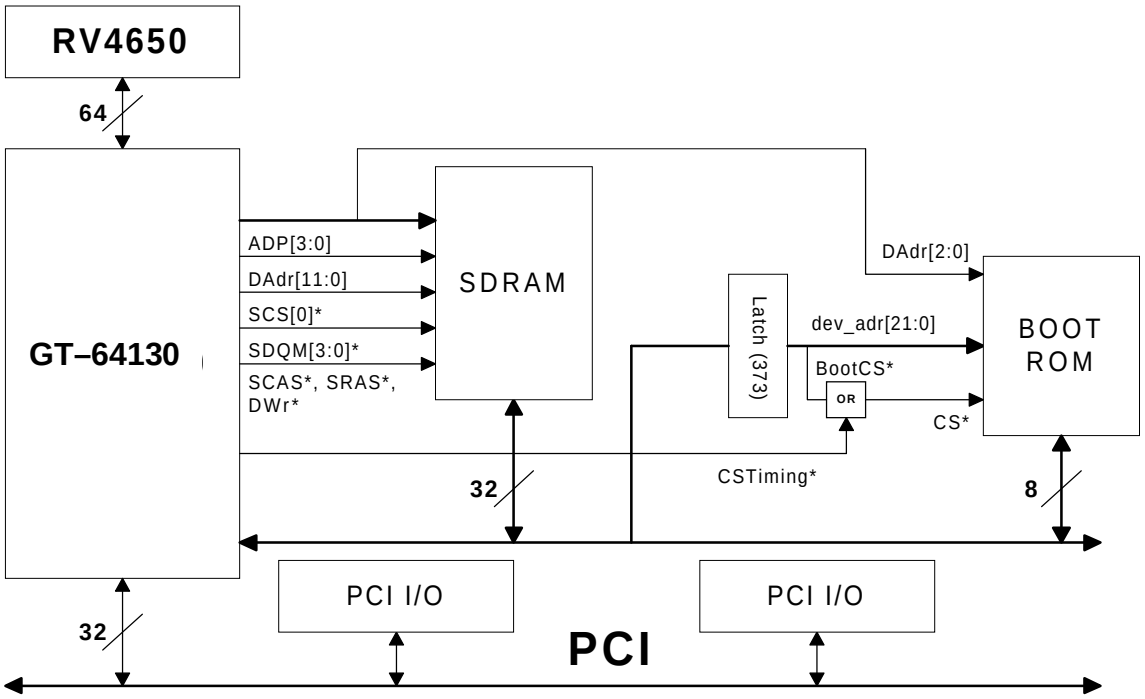
Pin	Pin Usage
TS*	TS*
TT[1]	RD/WR*
TT[3]	Pulled up to VCC through a resistor.
TBST*	BURST*
TSIZ[2]	TSIZ[1]
TSIZ[1]	TSIZ[0]
TSIZ[0]	Pulled down to GND through a resistor.
TA*	TA*
QsAEn	No Connect
QsAEn2	No Connect
A/DL[0-31]	A[0-31]
DH[0-31]	D[0-31]
DP[0-3]	DP[0-3]
DP[4-7]	No Connect
AACK*	No Connect
BG*	No Connect
DBG*	No Connect
L2BR*	Pulled up to VCC through a resistor.
L2BG*	No Connect
L2DBG*	No Connect
ARTRY*	Pulled up to VCC through a resistor.
L2HIT*	Pulled up to VCC through a resistor.
Interrupt*	Interrupt*

18. SYSTEM CONFIGURATIONS

18.1 Minimal System Configuration

- Low Cost RV4650 CPU
- 32-bit SDRAM
- 8-bit Boot EPROM
- 32-bit PCI

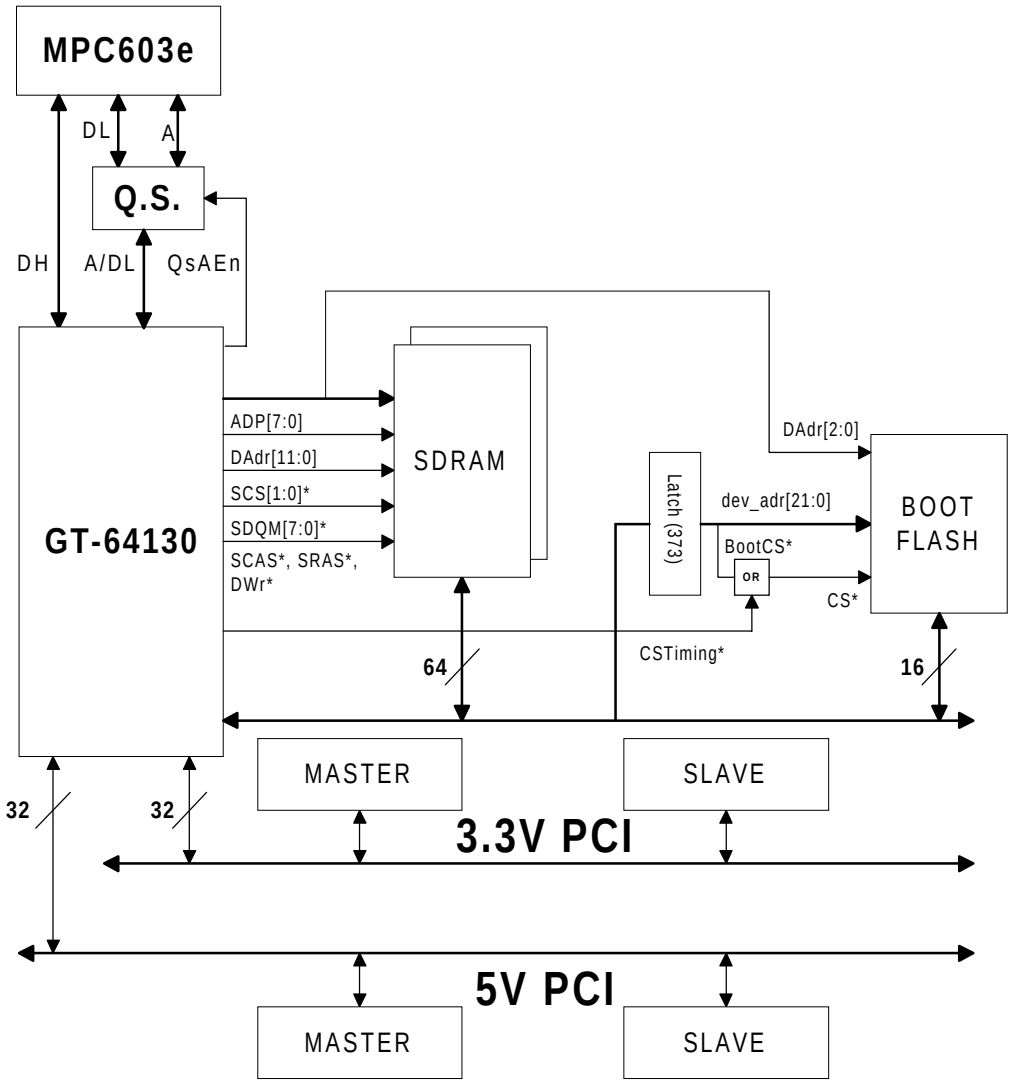
Figure 35: Minimal System Configuration



18.2 Typical System

- MPC603e CPU
- 16-bit Devices
- 64-bit SDRAM
- Two 32-bit PCI buses (3.3 and 5V support)

Figure 36: Typical System Configuration

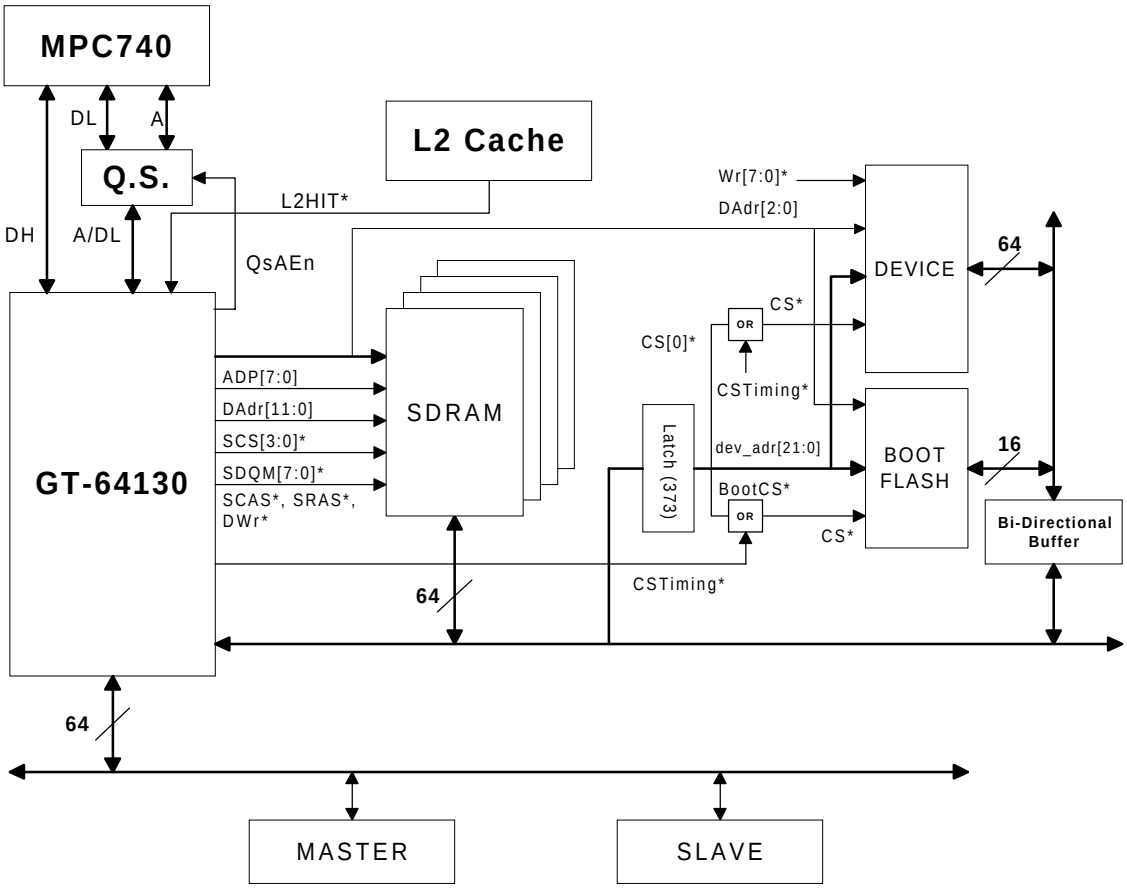


18.3 High Performance System

- MPC740 CPU
- 2nd Level Cache
- 64-bit Devices
- 64-bit PCI
- Multiple Banks of SDRAM

- Buffer used for Large AD loading

Figure 37: High Performance System Configuration



19. REGISTER TABLES

NOTES: When referring to an address in this section, the high number is the most significant bit and the lowest number is the least significant bit. For example, bit [31:0] means that the most significant bit is 31 and the least significant bit is 0.

This is opposite of the PowerPC convention in which the lowest bit number is the most significant bit and the highest bit number is the least significant bit.

The GT-64130's internal registers can be accessed by the CPU or from the PCI bus.

They are memory-mapped for the CPU and memory- or I/O-mapped for the PCI. The registers' address is comprised of the value in the Internal Space Decode register and the register Offset. The value in the Internal Space Decode register [10:0] is matched against bits [31:21] of the actual address. This value must be the actual address bits [31:21] shifted right once.

For example, to access DMA Channel 0 Byte Count register (offset 0x800) immediately after Reset, the full address will be the default value in the Internal Space Decode register which is 0x0a0 shifted left once, which gives 0x140, two zero's and the offset 0x800, to become a 32-bit address of 0x14000800.

The location of the registers in the memory space can be changed by changing the value programmed into the Internal Space Decode register. For example, after changing the value in the Internal Space Decode register by writing to 0x14000068 a value of 0bd, an access to the Channel 0 DMA Byte Count register is with 0x17a00800.

When writing to the internal registers from the PCI with Byte Enable = 0xF, the write is ignored (as per PCI specification). If a write occurs to the following registers with at least one CBE* pin asserted, the entire 32-bit word is written:

- CPU Interface.
- Processor Address Space Decoders.
- Device Address Space Decoders.
- All SDRAM and Device Registers.
- All DMA Registers.
- Timer/Counter.

The following internal registers are CBE* sensitive:

- PCI Internal Registers.
- PCI Configuration Registers.
- Interrupt Registers.

19.1 Access to On-Chip PCI Configuration Space Registers

An access from the CPU to one of the GT-64130 PCI configuration register is performed differently than accesses to all other registers. The access is performed indirectly by writing the PCI configuration register offset into the Configuration Address register and then reading or writing the data from or to the Configuration Data register.

For example, to read data from the Status and Command register, the register offset 0x004 is written into the Configuration Address register, offset 0xcf8 (or full address from the previous example 0xbd000cf8). Then, reading from the Configuration Data register (offset 0xcfc), will return the data of the Status and Command register.

19.2 Register Map

Table 69: Register Map

Description	Offset	Page Number
CPU Configuration		
CPU Interface Configuration	0x000	page 169
Multi-GT Register	0x120	page 170
CPU Address Decode		
SCS[1:0]* Low Decode Address	0x008	page 171
SCS[1:0]* High Decode Address	0x010	page 171
SCS[3:2]* Low Decode Address	0x018	page 171
SCS[3:2]* High Decode Address	0x020	page 171
CS[2:0]* Low Decode Address	0x028	page 172
CS[2:0]* High Decode Address	0x030	page 172
CS[3]* & BootCS* Low Decode Address	0x038	page 172
CS[3]* & BootCS* High Decode Address	0x040	page 172
PCI_0 I/O Low Decode Address	0x048	page 172
PCI_0 I/O High Decode Address	0x050	page 173
PCI_0 Memory 0 Low Decode Address	0x058	page 173
PCI_0 Memory 0 High Decode Address	0x060	page 173
PCI_0 Memory 1 Low Decode Address	0x080	page 173
PCI_0 Memory 1 High Decode Address	0x088	page 173
PCI_1 I/O Low Decode Address	0x090	page 174
PCI_1 I/O High Decode Address	0x098	page 174
PCI_1 Memory 0 Low Decode Address	0x0a0	page 174
PCI_1 Memory 0 High Decode Address	0x0a8	page 174
PCI_1 Memory 1 Low Decode Address	0x0b0	page 174
PCI_1 Memory 1 High Decode Address	0x0b8	page 174

Table 69: Register Map (Continued)

Description	Offset	Page Number
CPU Address Decode (Continued)		
Internal Space Decode	0x068	page 175
Bus Error Address, Offset:	0x070	page 175
SCS[1:0]* Address Remap	0x0d0	page 175
SCS[3:2]* Address Remap	0x0d8	page 175
CS[2:0]* Remap	0x0e0	page 176
CS[3]* & BootCS* Remap	0x0e8	page 176
PCI_0 I/O Remap	0x0f0	page 176
PCI_0 Memory 0 Remap	0x7f8	page 176
PCI_0 Memory 1 Remap	0x100	page 176
PCI_1 I/O Remap	0x108	page 177
PCI_1 Memory 0 Remap	0x110	page 177
PCI_1 Memory 1 Remap	0x118	page 177
CPU Sync Barrier		
PCI_0 Sync Barrier Virtual Register	0x0c0	page 177
PCI_1 Sync Barrier Virtual Register	0x0c8	page 178
SDRAM and Device Address Decode		
SCS[0]* Low Decode Address	0x400	page 178
SCS[0]* High Decode Address	0x404	page 178
SCS[1]* Low Decode Address	0x408	page 178
SCS[1]* High Decode Address	0x40c	page 179
SCS[2]* Low Decode Address	0x410	page 179
SCS[2]* High Decode Address	0x414	page 179
SCS[3]* Low Decode Address	0x418	page 179
SCS[3]* High Decode Address	0x41c	page 179
CS[0]* Low Decode Address	0x420	page 180
CS[0]* High Decode Address	0x424	page 180
CS[1]* Low Decode Address	0x428	page 180
CS[1]* High Decode Address	0x42c	page 180
CS[2]* Low Decode Address	0x430	page 180

Table 69: Register Map (Continued)

Description	Offset	Page Number
SDRAM and Device Address Decode (Continued)		
CS[2]* High Decode Address	0x434	page 181
CS[3]* Low Decode Address	0x438	page 181
CS[3]* High Decode Address	0x43c	page 181
BootCS* Low Decode Address	0x440	page 181
BootCS* High Decode Address	0x444	page 181
Address Decode Error	0x470	page 182
SDRAM Configuration		
SDRAM Configuration	0x448	page 182
SDRAM Operation Mode	0x474	page 183
SDRAM Burst Mode	0x478	page 183
SDRAM Address Decode	0x47c	page 184
SDRAM Parameters		
SDRAM Bank0 Parameters	0x44c	page 184
SDRAM Bank1 Parameters	0x450	page 185
SDRAM Bank2 Parameters	0x454	page 185
SDRAM Bank3 Parameters	0x458	page 186
ECC		
ECC Upper Data	0x480	page 186
ECC Lower Data	0x484	page 186
ECC from Memory	0x488	page 186
ECC Calculated	0x48c	page 186
ECC Error report	0x490	page 187
Device Parameters		
Device Bank0 Parameters	0x45c	page 187
Device Bank1 Parameters	0x460	page 188
Device Bank2 Parameters	0x464	page 188
Device Bank3 Parameters	0x468	page 188
Device Boot Bank Parameters	0x46c	page 189

Table 69: Register Map (Continued)

Description	Offset	Page Number
DMA Record		
Channel 0 DMA Byte Count	0x800	page 189
Channel 1 DMA Byte Count	0x804	page 189
Channel 2 DMA Byte Count	0x808	page 189
Channel 3 DMA Byte Count	0x80c	page 190
Channel 0 DMA Source Address	0x810	page 190
Channel 1 DMA Source Address	0x814	page 190
Channel 2 DMA Source Address	0x818	page 190
Channel 3 DMA Source Address	0x81c	page 190
Channel 0 DMA Destination Address	0x820	page 191
Channel 1 DMA Destination Address	0x824	page 191
Channel 2 DMA Destination Address	0x828	page 191
Channel 3 DMA Destination Address	0x82c	page 191
Channel 0 Next Record Pointer	0x830	page 191
Channel 1 Next Record Pointer	0x834	page 191
Channel 2 Next Record Pointer	0x838	page 192
Channel 3 Next Record Pointer	0x83c	page 192
Channel 0 Current Descriptor Pointer	0x870	page 192
Channel 1 Current Descriptor Pointer	0x874	page 192
Channel 2 Current Descriptor Pointer	0x878	page 192
Channel 3 Current Descriptor Pointer	0x87c	page 193
DMA Channel Control		
Channel 0 Control	0x840	page 193
Channel 1 Control	0x844	page 196
Channel 2 Control	0x848	page 196
Channel 3 Control	0x84c	page 196
DMA Arbiter		
Arbiter Control	0x860	page 197

Table 69: Register Map (Continued)

Description	Offset	Page Number
Timer/Counter		
Timer /Counter 0	0x850	page 198
Timer /Counter 1	0x854	page 198
Timer /Counter 2	0x858	page 198
Timer /Counter 3	0x85c	page 198
Timer /Counter Control	0x864	page 198
PCI Internal		
PCI_0 Command	0xc00	page 200
PCI_1 Command	0xc80	page 201
PCI_0 Time Out & Retry	0xc04	page 201
PCI_1 Time Out & Retry	0xc84	page 202
PCI_0 SCS[1:0]* Bank Size	0xc08	page 202
PCI_1 SCS[1:0]* Bank Size	0xc88	page 202
PCI_0 SCS[3:2]* Bank Size	0xc0c	page 203
PCI_1 SCS[3:2]* Bank Size	0xc8c	page 203
PCI_0 CS[2:0]* Bank Size	0xc10	page 203
PCI_1 CS[2:0]* Bank Size	0xc90	page 204
PCI_0 CS[3]* & BootCS* Bank Size	0xc14	page 204
PCI_1 CS[3]* & BootCS* Bank Size	0xc94	page 204
PCI_0 Base Address Registers' Enable	0xc3c	page 204
PCI_1 Base Address Registers' Enable	0xcbc	page 206
PCI_0 Prefetch/Max Burst Size	0xc40	page 206
PCI_1 Prefetch/Max Burst Size	0xcc0	page 206
PCI_0 SCS[1:0]* Base Address Remap	0xc48	page 207
PCI_1 SCS[1:0]* Base Address Remap	0xcc8	page 207
PCI_0 SCS[3:2]* Base Address Remap	0xc4c	page 207
PCI_1 SCS[3:2]* Base Address Remap	0xccc	page 208
PCI_0 CS[2:0]* Base Address Remap	0xc50	page 208
PCI_1 CS[2:0]* Base Address Remap	0xcd0	page 209
PCI_0 CS[3]* & BootCS* Address Remap	0xc54	page 209

Table 69: Register Map (Continued)

Description	Offset	Page Number
PCI Internal (Continued)		
PCI_1 CS[3]* & BootCS* Address Remap	0xcd4	page 209
PCI_0 Swapped SCS[1:0]* Base Address Remap	0xc58	page 207
PCI_1 Swapped SCS[1:0]* Base Address Remap	0xcd8	page 207
PCI_0 Swapped SCS[3:2]* Base Address Remap	0xc5c	page 207
PCI_1 Swapped SCS[3:2]* Base Address Remap	0xcdc	page 208
PCI_0 Swapped CS[3]* & BootCS* Base Address Remap	0xc64	page 209
PCI_1 Swapped CS[3]* & BootCS* Base Address Remap	0xce4	page 210
PCI_0 Configuration Address	0xcf8	page 210
PCI_1 Configuration Address	0xcf0	page 210
PCI_0 Configuration Data Virtual Register	0xcfc	page 210
PCI_1 Configuration Data Virtual Register	0xcf4	page 211
PCI_0 Interrupt Acknowledge Virtual Register	0xc34	page 211
PCI_1 Interrupt Acknowledge Virtual Register	0xc30	page 211
Interrupts		
Interrupt Cause Register	0xc18	page 211
High Interrupt Cause Register	0xc98	page 213
CPU Interrupt Mask Register	0xc1c	page 214
CPU High Interrupt Mask Register	0xc9c	page 215
PCI_0 Interrupt Cause Mask Register	0xc24	page 215
PCI_0 High Interrupt Cause Mask Register	0xca4	page 216
PCI_0 SErr0 Mask	0xc28	page 217
PCI_1 SErr1 Mask	0xca8	page 218
CPU Select Cause Register	0xc70	page 214
PCI_0 Interrupt Select Register	0xc74	page 214
PCI Configuration		
PCI_0 Device and Vendor ID	0x000	page 219
PCI_1 Device and Vendor ID	0x080	page 219
PCI_0 Status and Command	0x004	page 219
PCI_1 Status and Command	0x084	page 221

Table 69: Register Map (Continued)

Description	Offset	Page Number
PCI Configuration (Continued)		
PCI_0 Class Code and Revision ID	0x008	page 221
PCI_1 Class Code and Revision ID	0x088	page 221
PCI_0 BIST, Header Type, Latency Timer, Cache Line	0x00c	page 221
PCI_1 BIST, Header Type, Latency Timer, Cache Line	0x08c	page 222
PCI_0 SCS[1:0]* Base Address	0x010	page 223
PCI_1 SCS[1:0]* Base Address	0x090	page 223
PCI_0 SCS[3:2]* Base Address	0x014	page 223
PCI_1 SCS[3:2]* Base Address	0x094	page 224
PCI_0 CS[2:0]* Base Address	0x018	page 224
PCI_1 CS[2:0]* Base Address	0x098	page 225
PCI_0 CS[3]* & BootCS* Base Address	0x01c	page 225
PCI_1 CS[3]* & BootCS* Base Address	0x09c	page 225
PCI_0 Internal Registers Memory Mapped Base Address	0x020	page 225
PCI_1 Internal Registers Memory Mapped Base Address	0x0a0	page 226
PCI_0 Internal Registers I/O Mapped Base Address	0x024	page 226
PCI_1 Internal Registers I/O Mapped Base Address	0x0a4	page 226
PCI_0 Subsystem ID and Subsystem Vendor ID	0x02c	page 227
PCI_1 Subsystem ID and Subsystem Vendor ID	0x0ac	page 227
Expansion ROM Base Address Register	0x030	page 227
PCI_0 Interrupt Pin and Line	0x03c	page 227
PCI_1 Interrupt Pin and Line	0x0bc	page 228
PCI Configuration, Function 1		
PCI_0 Swapped SCS[1:0]* Base Address	0x110	page 228
PCI_1 Swapped SCS[1:0]* Base Address	0x190	page 229
PCI_0 Swapped SCS[3:2]* Base Address	0x114	page 229
PCI_1 Swapped SCS[3:2]* Base Address	0x194	page 229
PCI_0 Swapped CS[3]* & BootCS* Base Address	0x11c	page 229
PCI_1 Swapped CS[3]* & BootCS* Base Address	0x19c	page 230

Table 69: Register Map (Continued)

Description	Offset	Page Number
I_2O Support Registers [1]		
Inbound Message Register 0	0x10	page 231
Inbound Message Register 1	0x14	page 231
Outbound Message Register 0	0x18	page 231
Outbound Message Register 1	0x1c	page 232
Inbound Doorbell Register	0x20	page 232
Inbound Interrupt Cause Register	0x24	page 232
Inbound Interrupt Mask Register	0x28	page 233
Outbound Doorbell Register	0x2c	page 234
Outbound Interrupt Cause Register	0x30	page 234
Outbound Interrupt Mask Register	0x34	page 235
Inbound Queue Port Virtual Register	0x40	page 235
Outbound Queue Port Virtual Register	0x44	page 235
Queue Control Register	0x50	page 236
Queue Base Address Register	0x54	page 236
Inbound Free Head Pointer Register	0x60	page 236
Inbound Free Tail Pointer Register	0x64	page 237
Inbound Post Head Pointer Register	0x68	page 237
Inbound Post Tail Pointer Register	0x6c	page 237
Outbound Free Head Pointer Register	0x70	page 238
Outbound Free Tail Pointer Register	0x74	page 238
Outbound Post Head Pointer Register	0x78	page 238
Outbound Post Tail Pointer Register	0x7c	page 239

1. It is possible to access the I_2O registers from the CPU and PCI_0 sides (unless stated otherwise). If accessed from PCI_0 side, address offset is with respect to the PCI_0 SCS[1:0]* Base Address Register contents. If accessed from CPU side, address offset is with respect to the CPU Internal Space Base Register + 0x1c00.

NOTE: When referring to an address in this section, the high number is the most significant bit and the lowest number is the least significant bit. For example, bit [31:0] means that the most significant bit is 31 and the least significant bit is 0.

This is opposite of the PowerPC convention in which the lowest bit number is the most significant bit and the highest bit number is the least significant bit.

19.3 CPU Configuration

Table 70: CPU Interface Configuration, Offset: 0x000

Bits	Field name	Function	Initial Value
7:0	NoMatchCnt	CPU address miss counter.	0xff
8	NoMatchCntEn	CPU address miss counter enable. Relevant only if MultiGT was enabled. 0 - disabled 1 - enabled.	0x0
9	Reserved	Must be 0.	0x0
10	BlockRdBuffer	Block Read Buffer (relevant only for 64-bit bus) 0 - Block read data passes through read buffer. 1 - Block read data bypasses the read buffer.	0x0
11	AACK Delay	Address Acknowledge Delay 0 - AACK* is asserted one cycle after TS*. 1 - AACK* is asserted two cycles after TS*.	0x1
12	Endianess	Byte orientation. 0 - Big Endian 1 - Little Endian	Sampled at RESET via the Interrupt* pin.
13	Reserved	Reserved.	Sampled at RESET via the DAdr[1] pin.[1]
14	L2 present	Second level cache present. 0 - L2 not present L2HIT* input not monitored. 1 - L2 present L2HIT* input monitored.	0x0
15	L2Hit* Delay	L2 HIT Delay 0 - L2HIT* is asserted one cycle after TS*. 1 - L2HIT* is asserted two cycles after TS*.	0x0

Table 70: CPU Interface Configuration, Offset: 0x000 (Continued)

Bits	Field name	Function	Initial Value
16	Reserved	Reserved.	0x0
17	Stop Retry	Relevant only if PCI Retry is enabled. 0 - Continue retrying all PCI transactions targeted to the GT-64130. 1 - Stop retrying all PCI transactions.	0x0
18	MultiGT	Multiple GT-64130 support 0 - Not Supported 1 - Supported	Sampled at RESET via the DAdr[10] pin
19	Reserved	Reserved.	0x0
21:20	PCI_0 Override	PCI_0 Address Decoding Override 00 - No override. 01 - 1Gbyte PCI_0 memory address space. 10 - 2Gbyte PCI_0 memory address space. 11 - Overrides if no match in any of the address decoders.	0x0
23:22	Reserved	Reserved.	
25:24	PCI_1 Override	PCI_1 Address Decoding Override 00 - No override. 01 - 1Gbyte PCI_1 memory address space. 10 - 2Gbyte PCI_1 memory address space. 11 - Overrides if no match is found in any of the address decoders	0x0
31:26	Reserved	Reserved.	0x0

1. DADr[1] must be set to ‘0’ at reset.

Table 71: Multi-GT register, Offset: 0x120

Bits	Field Name	Function	Initial Value
1:0	MultiGTAct	Multi-GT Activity bits. These bits represent the code that the GT-64130 responds with activity.	Value sampled at reset on Redy* and CSTiming*
31:2	Reserved		0x0

19.4 CPU Address Decode

Table 72: SCS[1:0]* Low Decode Address, Offset: 0x008

Bits	Field Name	Function	Initial Value
10:0	Low	SDRAM banks 1 and 0 are accessed when the decoded addresses are between Low and High.	0x000
31:11	Reserved		0x0

Table 73: SCS[1:0]* High Decode Address, Offset: 0x010

Bits	Field Name	Function	Initial Value
6:0	High	SDRAM banks 1 and 0 are accessed when the decoded addresses are between Low and High.	0x07
31:7	Reserved		0x0

Table 74: SCS[3:2]* Low Decode Address, Offset: 0x018

Bits	Field Name	Function	Initial Value
10:0	Low	SDRAM banks 3 and 2 are accessed when the decoded addresses are between Low and High.	0x008
31:11	Reserved		0x0

Table 75: SCS[3:2]* High Decode Address, Offset: 0x020

Bits	Field Name	Function	Initial Value
6:0	High	SDRAM banks 3 and 2 are accessed when the decoded addresses are between Low and High.	0x0f
31:7	Reserved		0x0

Table 76: CS[2:0]* Low Decode Address, Offset: 0x028

Bits	Field Name	Function	Initial Value
10:0	Low	Device banks 2, 1, and 0 are accessed when the decoded addresses are between Low and High.	0x00e0
31:11	Reserved		0x0

Table 77: CS[2:0]* High Decode Address, Offset: 0x030

Bits	Field Name	Function	Initial Value
6:0	High	Device banks 2, 1, and 0 are accessed when the decoded addresses are between Low and High.	0x70
31:7	Reserved		0x0

Table 78: CS[3]* & BootCS* Low Decode Address, Offset: 0x038

Bits	Field Name	Function	Initial Value
10:0	Low	Device bank 3 and the boot bank are accessed when the decoded addresses are between Low and High.	0x7f8
31:11	Reserved		0x0

Table 79: CS[3]* & BootCS* High Decode Address, Offset: 0x040

Bits	Field Name	Function	Initial Value
6:0	High	Device bank 3 and the boot bank are accessed when the decoded addresses are between Low and High.	0x7f
31:7	Reserved		0x0

Table 80: PCI_0 I/O Low Decode Address, Offset: 0x048

Bits	Field Name	Function	Initial Value
10:0	Low	The PCI0 I/O address space is accessed when the decoded addresses are between Low and High.	0x080
31:11	Reserved		0x0

Table 81: PCI_0 I/O High Decode Address, Offset: 0x050

Bits	Field Name	Function	Initial Value
6:0	High	The PCI0 I/O address space is accessed when the decoded addresses are between Low and High.	0x0f
31:7	Reserved		0x0

Table 82: PCI_0 Memory 0 Low Decode Address, Offset: 0x058

Bits	Field Name	Function	Initial Value
10:0	Low	The PCI0 memory address space is accessed when the decoded addresses are between Low and High.	0x090
31:11	Reserved		0x0

Table 83: PCI_0 Memory 0 High Decode Address, Offset: 0x060

Bits	Field Name	Function	Initial Value
6:0	High	The PCI0 memory address space is accessed when the decoded addresses are between Low and High.	0x1f
31:7	Reserved		0x0

Table 84: PCI_0 Memory 1 Low Decode Address, Offset: 0x080

Bits	Field Name	Function	Initial Value
10:0	Low	The PCI0 memory address space is accessed when the decoded addresses are between Low and High.	0x790
31:11	Reserved		0x0

Table 85: PCI_0 Memory 1 High Decode Address, Offset: 0x088

Bits	Field Name	Function	Initial Value
6:0	High	The PCI0 memory address space is accessed when the decoded addresses are between Low and High.	0x1f
31:7	Reserved		0x0

Table 86: PCI_1 I/O Low Decode Address, Offset: 0x090

Bits	Field Name	Function	Initial Value
10:0	Low	The PCI1 I/O address space is accessed when the decoded addresses are between Low and High.	0x100
31:11	Reserved		0x0

Table 87: PCI_1 I/O High Decode Address, Offset: 0x098

Bits	Field Name	Function	Initial Value
6:0	High	The PCI1 I/O address space is accessed when the decoded addresses are between Low and High.	0x0f
31:7	Reserved		0x0

Table 88: PCI_1 Memory 0 Low Decode Address, Offset: 0x0a0

Bits	Field Name	Function	Initial Value
10:0	Low	The PCI1 memory address space is accessed when the decoded addresses are between Low and High.	0x110
31:11	Reserved		0x0

Table 89: PCI_1 Memory 0 High Decode Address, Offset: 0x0a8

Bits	Field Name	Function	Initial Value
6:0	High	The PCI1 memory address space is accessed when the decoded addresses are between Low and High.	0x1f
31:7	Reserved		0x0

Table 90: PCI_1 Memory 1 Low Decode Address, Offset: 0x0b0

Bits	Field Name	Function	Initial Value
10:0	Low	The PCI1 memory address space is accessed when the decoded addresses are between Low and High.	0x120
31:11	Reserved		0x0

Table 91: PCI_1 Memory 1 High Decode Address, Offset: 0x0b8

Bits	Field Name	Function	Initial Value
6:0	High	The PCI1 memory address space is accessed when the decoded addresses are between Low and High.	0x2f
31:7	Reserved		0x0

Table 92: Internal Space Decode, Offset: 0x068

Bits	Field Name	Function	Initial Value
10:0	IntDecode	Registers inside the GT-64130 is accessed when CPU address bits 31:21 match the value programmed in bits 10:0.	0x0a0
31:11	Reserved		0x0

Table 93: Bus Error Address, Offset: 0x070

Bits	Field Name	Function	Initial Value
31:0	IlegAdd	This register captures illegal CPU addresses.	0x0

Table 94: SCS[1:0]* Address Remap, Offset: 0x0d0

Bits	Field Name	Function	Initial Value
10:0	SCS[1:0]* Remap	CPU address remap to resources for the SDRAM 0 region.	0x0
31:11	Reserved		0x0

Table 95: SCS[3:2]* Address Remap, Offset: 0x0d8

Bits	Field Name	Function	Initial Value
10:0	SCS[3:2]* Remap	CPU address remap to resources of the SDRAM 1 region.	0x008
31:11	Reserved		0x0

Table 96: CS[2:0]* Address Remap, Offset: 0x0e0

Bits	Field Name	Function	Initial Value
10:0	CS[2:0]* Remap	CPU address remap to resources of the Device 0 region.	0x0e0
31:11	Reserved		0x0

Table 97: CS[3]* & BootCS* Address Remap, Offset: 0x0e8

Bits	Field Name	Function	Initial Value
10:0	CS[3]* & BootCS* Remap	CPU address remap to resources of the Device 1 region.	0x7f8
31:11	Reserved		0x0

Table 98: PCI_0 I/O Address Remap, Offset: 0x0f0

Bits	Field Name	Function	Initial Value
10:0	PCI_0 I/O Remap	CPU address remap to resources of the PCI_0 I,O region	0x080
31:11	Reserved		0x0

Table 99: PCI_0 Memory 0 Address Remap, Offset: 0x0f8

Bits	Field Name	Function	Initial Value
10:0	PCI_0 Mem0 Remap	CPU address remap to resources of the PCI_0 Memory 0 region.	0x090
31:11	Reserved		0x0

Table 100: PCI_0 Memory 1 Address Remap, Offset: 0x100

Bits	Field Name	Function	Initial Value
10:0	PCI_0 Mem1 Remap	CPU address remap to resources of the PCI_0 Memory 1 region.	0x790
31:11	Reserved		0x0

Table 101: PCI_1 I/O Address Remap, Offset: 0x108

Bits	Field Name	Function	Initial Value
10:0	PCI_1 I/O Remap	CPU address remap to resources of the PCI_1 I/O region.	0x100
31:11	Reserved		0x0

Table 102: PCI_1 Memory 0 Address Remap, Offset: 0x110

Bits	Field Name	Function	Initial Value
10:0	PCI_1 Mem0 Remap	CPU address remap to resources of the PCI_1 Memory 0 region.	0x110
31:11	Reserved		0x0

Table 103: PCI_1 Memory 1 Address Remap, Offset: 0x118

Bits	Field Name	Function	Initial Value
10:0	PCI_1 Mem1 Remap	CPU address remap to resources of the PCI_1 Memory 1 region.	0x120
31:11	Reserved		0x0

19.5 CPU Sync Barrier

Table 104: PCI_0 Sync Barrier Virtual Register, Offset: 0x0c0

Bits	Field Name	Function	Initial Value
31:0	SyncBarrier_0	A CPU read from this register creates a synchronization barrier cycle. When ValidIn* is returned to the CPU, both of the PCI_0 slave FIFOs are guaranteed empty. The read data returned to the CPU will be random and must be ignored. This register is READ ONLY.	0x0

Table 105: PCI_1 Sync Barrier Virtual Register, Offset: 0x0c8

Bits	Field Name	Function	Initial Value
31:0	SyncBarrier_1	A CPU read from this register creates a synchronization barrier cycle. When ValidIn* is returned to the CPU, both of the PCI_1 slave FIFOs are guaranteed empty. The read data returned to the CPU will be random and must be ignored. This register is READ ONLY.	0x0

19.6 SDRAM and Device Address Decode

Table 106: SCS[0]* Low Decode Address, Offset: 0x400

Bits	Field Name	Function	Initial Value
7:0	Low	SDRAM bank 0 is accessed when the decoded addresses are between Low and High.	0x00
31:8	Reserved		0x0

Table 107: SCS[0]* High Decode Address, Offset: 0x404

Bits	Field Name	Function	Initial Value
7:0	High	SDRAM bank 0 is accessed when the decoded addresses are between Low and High.	0x07
31:8	Reserved		0x0

Table 108: SCS[1]* Low Decode Address, Offset: 0x408

Bits	Field Name	Function	Initial Value
7:0	Low	SDRAM bank 1 is accessed when the decoded addresses are between Low and High.	0x08
31:8	Reserved		0x0

Table 109: SCS[1]* High Decode Address, Offset: 0x40c

Bits	Field Name	Function	Initial Value
7:0	High	SDRAM bank 1 is accessed when the decoded addresses are between Low and High.	0x0f
31:8	Reserved		0x0

Table 110: SCS[2]* Low Decode Address, Offset: 0x410

Bits	Field Name	Function	Initial Value
7:0	Low	SDRAM bank 2 is accessed when the decoded addresses are between Low and High.	0x10
31:8	Reserved		0x0

Table 111: SCS[2]* High Decode Address, Offset: 0x414

Bits	Field Name	Function	Initial Value
7:0	High	SDRAM bank 2 is accessed when the decoded addresses are between Low and High.	0x17
31:8	Reserved		0x0

Table 112: SCS[3]* Low Decode Address, Offset: 0x418

Bits	Field Name	Function	Initial Value
7:0	Low	SDRAM bank 3 is accessed when the decoded addresses are between Low and High.	0x18
31:8	Reserved		0x0

Table 113: SCS[3]* High Decode Address, Offset: 0x41c

Bits	Field Name	Function	Initial Value
7:0	High	SDRAM bank 3 is accessed when the decoded addresses are between Low and High.	0x1f
31:8	Reserved		0x0

Table 114: CS[0]* Low Decode Address, Offset: 0x420

Bits	Field Name	Function	Initial Value
7:0	Low	Device bank 0 is accessed when the decoded addresses are between Low and High.	0xc0
31:8	Reserved		0x0

Table 115: CS[0]* High Decode Address, Offset: 0x424

Bits	Field Name	Function	Initial Value
7:0	High	Device bank 0 is accessed when the decoded addresses are between Low and High.	0xc7
31:8	Reserved		0x0

Table 116: CS[1]* Low Decode Address, Offset: 0x428

Bits	Field Name	Function	Initial Value
7:0	Low	Device bank 1 is accessed when the decoded addresses are between Low and High.	0xc8
31:8	Reserved		0x0

Table 117: CS[1]* High Decode Address, Offset: 0x42c

Bits	Field Name	Function	Initial Value
7:0	High	Device bank 1 is accessed when the decoded addresses are between Low and High.	0xcf
31:8	Reserved		0x0

Table 118: CS[2]* Low Decode Address, Offset: 0x430

Bits	Field Name	Function	Initial Value
7:0	Low	Device bank 2 is accessed when the decoded addresses are between Low and High.	0xd0
31:8	Reserved		0x0

Table 119: CS[2]* High Decode Address, Offset: 0x434

Bits	Field Name	Function	Initial Value
7:0	High	Device bank 2 is accessed when the decoded addresses are between Low and High.	0xdf
31:8	Reserved		0x0

Table 120: CS[3]* Low Decode Address, Offset: 0x438

Bits	Field Name	Function	Initial Value
7:0	Low	Device bank 3 is accessed when the decoded addresses are between Low and High.	0xf0
31:8	Reserved		0x0

Table 121: CS[3]* High Decode Address, Offset: 0x43c

Bits	Field Name	Function	Initial Value
7:0	High	Device bank 3 is accessed when the decoded addresses are between Low and High.	0xfb
31:8	Reserved		0x0

Table 122: BootCS* Low Decode Address, Offset: 0x440

Bits	Field Name	Function	Initial Value
7:0	Low	Boot bank is accessed when the decoded addresses are between Low and High.	0xfc
31:8	Reserved		0x0

Table 123: BootCS* High Decode Address, Offset: 0x444

Bits	Field Name	Function	Initial Value
7:0	High	Boot bank is accessed when the decoded addresses are between Low and High.	0xff
31:8	Reserved		0x0

Table 124: Address Decode Error, Offset: 0x470

Bits	Field Name	Function	Initial Value
31:0	ErrAddr	The addresses of accesses to invalid address ranges are captured in this register. These are the address ranges not in the range programmed in the SDRAM or device decode registers.	0xffffffff

19.7 SDRAM Configuration

Table 125: SDRAM Configuration, Offset: 0x448

Bits	Field name	Function	Initial Value
13:0	RefIntCnt	Refresh Interval Count value	0x0200
14	Interleave	Bank Interleaving Control 0 - Interleaving Enabled 1 - Interleaving Disabled	0x0
15	RMW	Enable Read Modify Write 0 - Disabled 1 - Enabled	0x0
16	StagRef	Staggered refresh 0 - Staggered refresh 1- Non staggered refresh	0x0
18:17	Reserved		0x0
19	DupCntl	Duplicate Control Pins 0 - Do not duplicate. 1 - Duplicate Control pins.[1]	0x0
20	DupBA	Duplicate Bank Addresses 0 - Do not duplicate. 1 - Duplicate Bank Addresses.[2] {	0x0
21	DupEOT0	Duplicate End of Transfer 0 0 - Do not duplicate. 1 - Duplicate End of Transfer 0.[3]	0x0

Table 125: SDRAM Configuration, Offset: 0x448 (Continued)

Bits	Field name	Function	Initial Value
22	DupEOT1	Duplicate End of Transfer 1 0 - Do not duplicate. 1 - Duplicate End of Transfer 1.[4]	0x0
31:23	Reserved		0x0

1. DMAReq0* = SRAS*, DMAReq3* = SCAS*, BypsOE* = DWr*.

2. DMAReq[2]* = DAdr[11], DMAReq[1]* = BankSel[1]}

3. DMAReq[3]* = EOT0

4. Ready* = EOT1

Table 126: 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
31:3	Reserved		0x0

Table 127: SDRAM Burst Mode, Offset: 0x478

Bits	Field Name	Function	Initial Value
1:0	Reserved	Must be 0x1.	0x1
2	Burst Order	0 - Linear 1 - Sub-block	0x1
11:3	Reserved	Must be 0x1.	0x1
31:12	Reserved	Read Only 0.	0x0

Table 128: SDRAM Address Decode, Offset: 0x47c

Bits	Field Name	Function	Initial Value
2:0	AddrDecode	SDRAM Address Decode. See Section 5.1.4 SDRAM Address Decode Register (0x47c) on page 67 for more information.	0x2
31:3	Reserved		Undefined

19.8 SDRAM Parameters

Table 129: SDRAM Bank0 Parameters, Offset: 0x44c

Bits	Field name	Function	Initial Value
1:0	CASLat	CAS Latency Identical for all SDRAM banks. 1 - 2 cycles 2 - 3 cycles 3,0 - Reserved	0x1
2	FlowThrough	Flow-Through Enable 0 - One sample Must be set to 0 if ECC is enabled and 64-bit bypass is disabled. 1 - No Sample	0x1
3	SRASPrchg	SRAS Precharge Time 0 - 2 cycles 1 - 3 cycles	0x0
4	Reserved		0x0
5	64bitInt	64-bit SDRAM Interleaving 0 - Two way bank interleaving. 1 - Four way bank interleaving.	0x0
6	BankWidth	Width of the SDRAM bank. 0 - 32-(36-)bit wide SDRAM 1 - 64-(72-)bit wide SDRAM	0x0
7	BankLoc	32-bit SDRAM Bank Location Not applicable for 64-bit SDRAMs. 0 - Even, AD[31:0] 1 - Odd, AD[63:32]	0x0
8	ECC	ECC support for the bank. 0 - Not supported 1 - Supported	0x0

Table 129: SDRAM Bank0 Parameters, Offset: 0x44c (Continued)

Bits	Field name	Function	Initial Value
9	Bypass	Bypass enable to CPU. 0 - No Bypass 1 - Bypass	0x0
10	SRAStoSCAS	SRAS to SCAS delay 0 - 2 cycles 1 - 3 cycles	0x0
11	SDRAMSize	16- or 64/128-Mbit SDRAM 0 - 16-Mbit SDRAM 1 - 64/128-Mbit SDRAM	0x0
12	Reserved		0x0
13	BrstLen	Burst Length 0 - Burst of 8. 1 - Burst of 4.	0x0
31:14	Reserved		Undefined

Table 130: SDRAM Bank1 Parameters, Offset: 0x450

Bits	Field Name	Function	Initial Value
3:0	Various	Fields function as in SDRAM Bank0.	0x5
4	Disable Refresh	Disables refresh of all SDRAM banks. 0 - Refresh 1 - Disable refresh.	0x0
13:5	Various	Fields function as in SDRAM Bank0.	0x0
31:14	Reserved		Undefined

Table 131: SDRAM Bank2 Parameters, Offset: 0x454

Bits	Field Name	Function	Initial Value
3:0	Various	Fields function as in SDRAM Bank0.	0x5
4	TREQ* Enable	UMA Total Request pin enable 0 - Disable 1 - Enable DMAReq[3]*/SCAS* pin functions as total a request pin.	0x0
13:5	Various	Fields function as in SDRAM Bank0.	0x0
31:14	Reserved		Undefined

Table 132: SDRAM Bank3 Parameters, Offset: 0x458

Bits	Field Name	Function	Initial Value
13:0	Various	Fields function as in SDRAM Bank0.	0x5
31:14	Reserved		Undefined

19.9 ECC

Table 133: ECC Upper Data, Offset: 0x480

Bits	Field Name	Function	Initial Value
31:0	ECCUpData	Bits[63:32] of the last data with ECC error.	0x0

Table 134: ECC Lower Data, Offset: 0x484

Bits	Field Name	Function	Initial Value
31:0	ECCLoData	Bits[31:0] of the last data with ECC error.	0x0

Table 135: ECC from Mem, Offset: 0x488

Bits	Field Name	Function	Initial Value
7:0	ECCMem	Eight bits of ECC code read from memory.	0x0
31:8	Reserved		0x0

Table 136: ECC Calculated, Offset: 0x48c

Bits	Field Name	Function	Initial Value
7:0	ECCCalc	Eight bits of ECC code calculated inside the GT-64130.	0x0
31:8	Reserved		0x0

Table 137: ECC Error Report, Offset: 0x490

Bits	Field Name	Function	Initial Value
1:0	ECCErr	Number of ECC errors. 00 - No errors. 01 - One error detected and corrected. 10 - Two or more errors detected. 11 - Reserved.	0x0
31:2	Reserved		0x0

19.10 Device Parameters

Table 138: Device Bank0 Parameters, Offset: 0x45c

Bits	Field name	Function	Initial Value
2:0	TurnOff	The number of cycles from the deassertion of DevOE* and a new AD bus cycle. This is an externally extracted signal which is the logical OR between CSTiming* and inverted DevRW*.	0x7
6:3	AccToFirst	The number of cycles in a read access from the assertion of CS* to the cycle that the data is latched by the external latches. Extend the number of cycles via the Ready* pin.	0xf
10:7	AccToNext	The number of cycles in a read access from the cycle that the first data was latched to the cycle that the next data is latched, in burst accesses. Extend the number of cycles via the Ready* pin.	0xf
13:11	ALEtoWr	The number of cycles from the deassertion of ALE to the assertion of Wr*.	0x7
16:14	WrActive	The number of cycles Wr* signals are active. Extend the number of cycles via the Ready* pin.	0x7
19:17	WrHigh	The number of cycles between deassertion and assertion of Wr* signals.	0x7

Table 138: 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 - 64 bits	0x2
22	DMAFlyBy	Forwarded to BootCS* during Flyby DMA.	0x1
23	DevLoc	32-/16-/8-bit device Location 0 - Even bank (AD[31:0], AD[15:0], AD[7:0]) 1 - Odd bank (AD[63:32], AD[47:32], AD[39:32])	0x0
24	Reserved	Read only.	0x0
25	Reserved	Must be 0.	0x0
29:26	DMAFlyBy	Forwarded to CS[3:0]* during FlyBy DMA	0xe
31:30	Reserved	Must be 0.	0x0

Table 139: Device Bank1 Parameters, Offset: 0x460

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Device Bank0.	0x386fffff

Table 140: Device Bank2 Parameters, Offset: 0x464

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Device Bank0.	0x386fffff

Table 141: Device Bank3 Parameters, Offset: 0x468

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Device Bank0.	0x38?fffff

Table 142: Device Boot Bank Parameters, Offset: 0x46c

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Device Bank0, except for bits 29:26 and bit 22. These bits that are reserved.	14?fffff

NOTE: In case of Bank3 or Boot Bank, bits 23:20 are shown as '?' because bits 21:20 are sampled at reset via DAdr[4:3] to define the width of CS[3] and boot device.

19.11 DMA Record

Table 143: Channel 0 DMA Byte Count, Offset: 0x800

Bits	Field Name	Function	Initial Value
15:0	ByteCt	The number of bytes that are left in DMA transfers.	0x0
31:16	ByteRemain	If CDE is set to 1 and the DMA engine owns the descriptor (i.e. DMA is currently in progress), the remaining bytes to transfer are written. If the CPU owns the descriptor, these bits can be written to any value.	0x0

Table 144: Channel 1 DMA Byte Count, Offset: 0x804

Bits	Field Name	Function	Initial Value
31:0		Functions as in Channel 0 DMA Byte Count.	0x0

Table 145: Channel 2 DMA Byte Count, Offset: 0x808

Bits	Field Name	Function	Initial Value
31:0		Functions as in Channel 0 DMA Byte Count.	0x0

Table 146: Channel 3 DMA Byte Count, Offset: 0x80c

Bits	Field Name	Function	Initial Value
31:0		Functions as in Channel 0 DMA Byte Count.	0x0

Table 147: Channel 0 DMA Source Address, Offset: 0x810

Bits	Field Name	Function	Initial Value
31:0	SrcAdd	The address from which the DMA controller reads the data.	0x0

Table 148: Channel 1 DMA Source Address, Offset: 0x814

Bits	Field Name	Function	Initial Value
31:0	SrcAdd	The address from which the DMA controller reads the data.	0x0

Table 149: Channel 2 DMA Source Address, Offset: 0x818

Bits	Field Name	Function	Initial Value
31:0	SrcAdd	The address from which the DMA controller reads the data.	0x0

Table 150: Channel 3 DMA Source Address, Offset: 0x81c

Bits	Field Name	Function	Initial Value
31:0	SrcAdd	The address from which the DMA controller reads the data.	0x0

Table 151: Channel 0 DMA Destination Address, Offset: 0x820

Bits	Field Name	Function	Initial Value
31:0	DestAdd	The address to which the DMA controller writes the data.	0x0

Table 152: Channel 1 DMA Destination Address, Offset: 0x824

Bits	Field Name	Function	Initial Value
31:0	DestAdd	The address to which the DMA controller writes the data.	0x0

Table 153: Channel 2 DMA Destination Address, Offset: 0x828

Bits	Field Name	Function	Initial Value
31:0	DestAdd	The address to which the DMA controller writes the data.	0x0

Table 154: Channel 3 DMA Destination Address, Offset: 0x82c

Bits	Field Name	Function	Initial Value
31:0	DestAdd	The address to which the DMA controller writes the data.	0x0

Table 155: Channel 0 Next Record Pointer, Offset: 0x830

Bits	Field Name	Function	Initial Value
31:0	NextRecPtr	The address for the next record of DMA. A 0 value means a NULL pointer (end of the chained list). **NOTE:** The next record pointer must be 16 bytes aligned. This means bits [3:0] must be set to 0.	0x0

Table 156: Channel 1 Next Record Pointer, Offset: 0x834

Bits	Field Name	Function	Initial Value
31:0	NextRecPtr	The address for the next record of DMA. A 0 value means a NULL pointer (end of the chained list). **NOTE:** The next record pointer must be 16 bytes aligned. This means bits [3:0] must be set to 0.	0x0

Table 157: Channel 2 Next Record Pointer, Offset: 0x838

Bits	Field Name	Function	Initial Value
31:0	NextRecPtr	The address for the next record of DMA. A 0 value means a NULL pointer (end of the chained list). **NOTE:** The next record pointer must be 16 bytes aligned. This means bits [3:0] must be set to 0.	0x0

Table 158: Channel 3 Next Record Pointer, Offset: 0x83c

Bits	Field Name	Function	Initial Value
31:0	NextRecPtr	The address for the next record of DMA. A 0 value means a NULL pointer (end of the chained list). **NOTE:** The next record pointer must be 16bytes aligned. This means bits [3:0] must be set to 0.	0x0

Table 159: Current Descriptor Pointer 0, Offset: 0x870

Bits	Field Name	Function	Initial Value
31:0	CDPTR0	Channel 0 current address descriptor pointer.	0x0

Table 160: Current Descriptor Pointer 1, Offset: 0x874

Bits	Field Name	Function	Initial Value
31:0	CDPTR1	Channel 1 current address descriptor pointer.	0x0

Table 161: Current Descriptor Pointer 2, Offset: 0x878

Bits	Field Name	Function	Initial Value
31:0	CDPTR2	Channel 2 current address descriptor pointer.	0x0

Table 162: Current Descriptor Pointer 3, Offset: 0x87c

Bits	Field Name	Function	Initial Value
31:0	CDPTR3	Channel 3 current address descriptor pointer.	0x0

19.12 DMA Channel Control

Table 163: Channel 0 Control, Offset: 0x840

Bits	Field name	Function	Initial Value
0	FlyByEn	Data Internal/External to DMA FIFO. 0 - Internal Data is read from the source address into DMA FIFO and written to destination address 1 - External (Fly By) Data is transferred to/from devices on the SDRAM bus from/to SDRAM	0x0
1	RdWrFly	SDRAM Read/Write This is meaningful only in Fly-by mode. 0 - Read from SDRAM. 1 - Write to SDRAM.	0x0
3:2	SrcDir	Source Direction 00 - Increment source address 01 - Decrement source address 10 - Hold in the same value 11 - Reserved	0x0
5:4	DestDir	Destination Direction 00 - Increment destination address 01 - Decrement destination address 10 - Hold in the same value 11 - Reserved	0x0
8:6	DatTransLim	Data Transfer Limit in each DMA access. 101 - 1 Byte 110 - 2 Bytes 010 - 4 Bytes 000 - 8 Bytes 001 - 16 Bytes 011 - 32 Bytes 111 - 64 bytes 100 - Reserved	0x0

Table 163: Channel 0 Control, Offset: 0x840 (Continued)

Bits	Field name	Function	Initial Value
9	ChainMod	Chained Mode. 0 - Chained mode When a DMA access is terminated, the parameters of the next DMA access comes from a record in memory pointed at by a NextRecPtr register points. 1 - Non-Chained mode Only the values that are programmed by the CPU (or PCI) directly into the ByteCt, SrcAdd, and DestAdd registers are used.	0x0
10	IntMode	Interrupt Mode 0 - Interrupt asserted every time the DMA byte count reaches terminal count. 1 - Interrupt every NULL pointer (in chained mode).	0x0
11	TransMod	Transfer Mode 0 - Demand 1 - Block	0x0
12	ChanEn	Channel Enable 0 - Disable 1 - Enable	0x0
13	FetNexRec	Fetch Next Record 1 - Forces a fetch of the next record (even if the current DMA has not ended). In chained mode, this bit is reset after fetch is completed.	0x0
14	DMAActSt	DMA Activity Status (read only) Assertion of this bit is caused by asserting ChanEn (bit 12). 0 - Channel is not active. 1 - Channel is active.	0x0
15	SDA	Source Destination Alignment 0 - Alignment is towards the source address. 1 - Alignment is done towards the destination address. (No support for destination alignment when burst limit is 1, 2, or 4 bytes.)	0x0

Table 163: Channel 0 Control, Offset: 0x840 (Continued)

Bits	Field name	Function	Initial Value
16	MDREQ	Mask DMA Requests 0 - Don't Mask. 1 - Mask DMA requests for 3 cycles starting DMA arbitration cycle.	0x0
17	CDE	Close Descriptor Enable. If enabled, DMA writes remaining of byte count to bits [31:16] of ByteCount field in the descriptor. 0 - Disable 1 - Enable	0x0
18	EOTE	End Of Transfer Enable. If enabled, DMA transfer can be stopped in the middle of transfer using EOT signal. If DMA channel is working in chain mode then this causes fetching a new descriptor, otherwise the DMA transfer is stopped. 0 - Disable 1 - Enable	0x0
19	EOTE	End Of Transfer Interrupt Enable If enabled and EOT pin is asserted, DMA generates an interrupt 0 - Disable 1 - Enable	0x0
20	ABR	Abort DMA transfer. Only set this bit if the CPU wants to stop and re-program DMA, and CDE (bit 17) and/or EOTE (bit 19) are set to 1. When the CPU issues this command, it should also set ChanEn (bit 12) to 0. 0 - No influence on channel behavior. 1 - Abort DMA.	0x0
22:21	SLP	Override Source Address 00 - No override. Use local address space for source. 01 - Source address is in PCI_0 memory space. 10 - Source address is in PCI_1 memory space. 11 - Reserved.	0x0

Table 163: Channel 0 Control, Offset: 0x840 (Continued)

Bits	Field name	Function	Initial Value
24:23	DLP	Override Destination Address 00 - No override. Use local address space for destination. 01 - Destination address is in PCI_0 memory space. 10 - Destination address is in PCI_1 memory space. 11 - Reserved.	0x0
26:25	RLP	Override Record Address 00 - No override. Use local address space for record. 01 - Record address is in PCI_0 memory space. 10 - Record address is in PCI_1 memory space. 11 - Reserved.	0x0
27	Reserved		0x0
28	DMAReqSrc	DMA Request Source 0 - Request taken from DMAReq* pin. 1 - Request taken from timer/counter.	0x0
31:29	Reserved		0x0

Table 164: Channel 1 Control, Offset: 0x844

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Channel 0 Control.	0x0

Table 165: Channel 2 Control, Offset: 0x848

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Channel 0 Control.	0x0

Table 166: Channel 3 Control, Offset: 0x84c

Bits	Field Name	Function	Initial Value
31:0	Various	Fields function as in Channel 0 Control.	0x0

19.13 DMA Arbiter

Table 167: Arbiter Control, Offset: 0x860

Bits	Field name	Function	Initial Value
1:0	PrioChan1/0	Priority between Channel 0 and Channel 1. 00 - Round Robin 01 - Priority to channel 1 over channel 0 10 - Priority to channel 0 over channel 1 11 - Reserved	0x0
3:2	PrioChan3/2	Priority between Channel 2 and Channel 3. 00 - Round Robin 01 - Priority to channel 3 over 2 10 - Priority to channel 2 over 3 11 - Reserved	0x0
5:4	PrioGrps	Priority between the group of channels 0/1 and the group of channels 2/3. 00 - Round Robin 01 - Priority to channels 2/3 over 0/1 10 - Priority to channels 0/1 over 2/3 11 - Reserved	0x0
6	PrioOpt	Priority Option enabled/disable. 0 - The high priority device relinquishes the bus for a requesting device for one DMA transaction after it was serviced. 1 - The High priority device is granted as long as it requests the bus.	0x0
31:7	Reserved		0x0

19.14 Timer/Counter

Table 168: Timer/Counter 0, Offset: 0x850

Bits	Field Name	Function	Initial Value
31:0	TC0Value	The counter or timer initial value.	0xffffffff

Table 169: Timer/Counter 1, Offset: 0x854

Bits	Field Name	Function	Initial Value
23:0	TC1Value	The counter or timer initial value.	0xffffff
31:24	Reserved		0x0

Table 170: Timer/Counter 2, Offset: 0x858

Bits	Field Name	Function	Initial Value
23:0	TC2Value	The counter or timer initial value.	0xffffff
31:24	Reserved		0x0

Table 171: Timer/Counter 3, Offset: 0x85c

Bits	Field Name	Function	Initial Value
23:0	TC3Value	The counter or timer initial value.	0xffffff
31:24	Reserved		0x0

Table 172: Timer/Counter Control, Offset: 0x864

Bits	Field name	Function	Initial Value
0	EnTC0	The timer or counter counts only when it is enabled. 0 - Disable 1 - Enable	0x0
1	SelTC0	Timer or counter selection. 0 - Counter 1 - Timer	0x0

Table 172: Timer/Counter Control, Offset: 0x864 (Continued)

Bits	Field name	Function	Initial Value
2	EnTC1	The timer or counter counts only when it is enabled. 0 - Disable 1 - Enable	0x0
3	SelTC1	Timer or counter selection. 0 - Counter 1 - Timer	0x0
4	EnTC2	The timer or counter counts only when it is enabled. 0 - Disable 1 - Enable	0x0
5	SelTC2	Timer or counter selection. 0 - Counter 1 - Timer	0x0
6	EnTC3	The timer/counter counts only when it is enabled. 0 - Disable 1 - Enable	0x0
7	SelTC3	Timer or counter selection. 0 - Counter 1 - Timer	0x0
31:8	Reserved		0x0

19.15 PCI Internal

Table 173: PCI_0 Command, Offset: 0xc00

Bits	Field name	Function	Initial Value
0	MByteSwap	Master Byte Swap When set to 0, the GT-64130 PCI master swaps the bytes of the incoming and outgoing PCI data (swap the 8 bytes of a long-word).	Set to the same value sampled at reset into bit[12] of the CPU Interface Configuration register.
3:1	SyncMode[1]	Indicates the following ratios between TClk and PClk: x00 - When the PClk ranges from DC to 66MHz (default mode) use the following settings for higher performance. 001 - When PClk is HIGHER than or Equal to half the TClk frequency (e.g. when TClk is 66MHz, SyncMode can be set to 001 if the PCI frequency is HIGHER than or Equal to 33MHz). 01x - When the two clocks are synchronized (derived from the same clock) and PClk is HIGHER than or EQUAL to half of TCLK frequency (e.g. TClk = 66MHz, PClk = 33MHz) 101 - When PClk is HIGHER than or Equal to third the TClk frequency but LESS than half of TCLK frequency 11x - When the two clocks are synchronized and PClk is HIGHER than or EQUAL to third of TCLK frequency but LESS than half of TCLK frequency	0x0
7:4	Reserved	Read only.	0x0
9:8	Reserved	Must be 0.	0x0
10	MWordSwap*	Master Word Swap When set to 1, the GT-64130 PCI master swaps the words of the incoming and outgoing PCI data (swap the 2 words of a long-word).	0x0
11	SWordSwap*	Slave Word Swap When set to 1and there is an address hit in one of SDRAM or Devices BARs, the GT-64130 PCI slave swaps the words of the incoming and outgoing PCI data (swap the 2 words of a long-word).	0x0

Table 173: PCI_0 Command, Offset: 0xc00 (Continued)

Bits	Field name	Function	Initial Value
12	SSBWordSwap*	Slave Swap BAR Word Swap When set to 1 and there is an address hit in one of SDRAM or Devices BARs, the GT-64130 PCI slave swaps the words of the incoming and outgoing PCI data (swap the 2 words of a long-word).	0x0
15:13	Reserved	Must be 0.	0x0
16	SByteSwap	Slave Byte Swap When set to 0, the GT-64130 PCI slave swaps the bytes of the incoming and outgoing PCI data (swap the 8 bytes of a long-word).	Set to the same value sampled at reset into bit[12] of the CPU Interface Configuration register.
31:17	Reserved	Must be 0	0x0

1. Regardless of the syncmode selected, PClk frequency must be smaller than TClk frequency by at least 1MHz.

*. This bit is not supported in a 64-bit PCI configuration.

Table 174: PCI_1 Command, Offset: 0xc80

Bits	Field name	Function	Initial Value
31:0	Various	Same as for PCI_0 Command.	

Table 175: PCI_0 Time Out & ReTry, Offset: 0xc04

Bits	Field name	Function	Initial Value
7:0	Timeout0	Specifies in PCI clock units the number of clocks the GT-64130, as a slave, holds the PCI bus before the generation of retry termination. Used for the first data transfer.	0x0f
15:8	Timeout1	Specifies in PCI clock units the number of clocks the GT-64130, as a slave, holds the PCI bus before the generation of disconnect termination. Used for data transfers following the first data.	0x07

Table 175: PCI_0 Time Out & ReTry, Offset: 0xc04

Bits	Field name	Function	Initial Value
23:16	RetryCtr	Specifies the number of retries of the GT-64130 master. The GT-64130 generates an interrupt when this timer expires. A value of 0x00 means “retry forever”. The number in RetryCtr does not include the first access of the transaction.	0x0
31:24	Reserved		0x0

Table 176: PCI_1 Time Out & ReTry, Offset: 0xc84

Bits	Field name	Function	Initial Value
31:0	Various	Same as for PCI_0 Time Out & ReTry.	0x070f

Table 177: PCI_0 SCS[1:0] Bank Size, Offset: 0xc08

Bits	Field Name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	BankSize	Specifies the SCS[1:0]* address mapping in conjunction with the SCS[1:0]* Base Address register. Set to 0 indicates that the corresponding bit in the address and in the base address must be equal in order to have a hit. Set to 1 indicates that the corresponding bit in the address is a don’t-care. For example, bit 12 set to 1 indicates that the SCS[1:0]* size is 8KBytes. The set bits in the Bank Size must be sequential (e.g. 000...001, 000...011, 000...111 are correct values, whereas 000...010 and 000...100 are incorrect).	0x00fff

Table 178: PCI_1 SCS[1:0]* Bank Size, Offset: 0xc88

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 SCS[1:0]* Bank Size.	0x00fff000

Table 179: PCI_0 SCS[3:2]* Bank Size, Offset: 0xc0c

Bits	Field Name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	BankSize	Specifies the SCS[3:2]* address mapping in conjunction with the SCS[3:2]* Base Address register. Set to 0 indicates that the corresponding bit in the address and in the base address must be equal in order to have a hit. Set to 1 indicates that the corresponding bit in the address is a don't-care. For example, bit 12 set to 1 indicates that the SCS[3:2]* size is 8KBytes. The set bits in the Bank Size must be sequential (e.g. 000...001, 000...011, 000...111 are correct values, whereas 000...010 and 000...100 are not).	0x00fff

Table 180: PCI_1 SCS[3:2]* Bank Size, Offset: 0xc8c

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 SCS[3:2]* Bank Size.	0x00fff000

Table 181: PCI_0 CS[2:0]* Bank Size, Offset: 0xc10

Bits	Field Name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	BankSize	Specifies the CS[2:0]* address mapping in conjunction with the CS[2:0]* Base Address register. Set to 0 indicates that the corresponding bit in the address and in the base address must be equal in order to have a hit. Set to 1 indicates that the corresponding bit in the address is a don't-care. For example, bit 12 set to 1 indicates that the CS[2:0]* size is 8KBytes. The set bits in the Bank Size must be sequential (e.g. 000...001, 000...011, 000...111 are correct values, whereas 000...010 and 000...100 are not).	0x01fff

Table 182: PCI_1 CS[2:0]* Bank Size, Offset: 0xc90

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 CS[2:0]* Bank Size.	0x01fff000

Table 183: PCI_0 CS[3]* and BootCS* Bank Size, Offset: 0xc14

Bits	Field Name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	BankSize	Specifies the CS[3]* and BootCS* address mapping in conjunction with the CS[3]* and BootCS* Base Address register. Set to 0 indicates that the corresponding bit in the address and in the base address must be equal in order to have a hit. Set to 1 indicates that the corresponding bit in the address is a don't-care. For example, bit 12 set to 1 indicates that the CS[3]*/ BootCS* size is 8KBytes. The set bits in the Bank Size must be sequential (e.g. 000...001, 000...011, 000...111 are correct values, whereas 000...010 and 000...100 are not).	0x00fff

Table 184: PCI_1 CS[3]* and BootCS* Bank Size, Offset: 0xc94

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 CS[3]* and BootCS* Bank Size.	0x00fff000

Table 185: PCI_0 Base Address Registers' Enable, Offset: 0xc3c

Bits	Field name	Function	Initial Value
0	SwCS[3] & BootCSEn	Controls address matching with Swapped CS[3]* & BootCS* base/size. 0 - Enable 1 - Disable	0x1
1	SwSCS[3:2]En	Controls address matching with Swapped SCS[3:2]* base/size. 0 - Enable 1 - Disable	0x1

Table 185: PCI_0 Base Address Registers' Enable, Offset: 0xc3c (Continued)

Bits	Field name	Function	Initial Value
2	SwSCS[1:0]En	Controls address matching with Swapped SCS[1:0]* base/size. 0 - Enable 1 - Disable	0x1
3	IntIOEn	Controls address matching with internal registers I/O mapped base/size. 0 - Enable 1 - Disable	0x1
4	IntMeMEn	Controls address matching with internal registers Memory mapped base/size. 0 - Enable 1 - Disable	0x0
5	CS[3] & BootCSEn	Controls address matching with CS[3]* and BootCS* base/size. 0 - Enable 1 - Disable	0x0
6	CS[2:0]En	Controls address matching with CS[2:0]* base/size. 0 - Enable 1 - Disable	0x0
7	SCS[3:2]En	Controls address matching with SCS[3:2]* base/size. 0 - Enable 1 - Disable	0x0
8	SCS[1:0]En	Controls address matching with SCS[1:0]* base/size. 0 - Enable 1 - Disable	0x0
31:9	Reserved		0x0

Table 186: PCI_1 Base Address Registers' Enable, Offset: 0xcbc (RESERVED if configured for only PCI_0)

Bits	Field name	Function	Initial Value
31-0		Same As for PCI_0 Base Address registers' enable.	0x0f

NOTE: The GT-64130 prevents disabling both memory mapped base/size address matching and I/O mapped base/size address matching at the same time (bits 3 and 4 cannot simultaneously be set to 1).

Table 187: PCI_0 Prefetch/Max Burst Size, Offset: 0xc40

Bits	Field name	Function	Initial Value
0	Dram0MaxBrst	SCS[1:0]* Max Burst length.	0x0
1	Dram1MaxBrst	SCS[3:2]* Max Burst length.	0x0
2	Dev0MaxBrst	CS[2:0]* Max Burst length.	0x0
3	Dev1MaxBrst	CS[3]* and BootCS* Max Burst length.	0x0
4	RdMemPref	Read Memory Prefetch	0x0
5	RdLnPref	Read Line Prefetch	0x0
6	RdMulPref	Read Multiple Prefetch	0x1
31:7	Reserved		0x0

Table 188: PCI_1 Prefetch/Max Burst Size, Offset: 0xcc0 (RESERVED if configured for only PCI_0)

Bits	Field name	Function	Initial Value
31-0	Various	Same As for PCI_0 Prefetch/Max Burst size .	0x040

Table 189: PCI_0 SCS[1:0]* Base Address Remap, Offset: 0xc48

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only	0x0
31:12	DRAM0Remap	SCS[1:0]* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_0 Base Address registers' enable.)	0x0

Table 190: PCI_1 SCS[1:0]* Base Address Remap, Offset: 0xcc8 (RESERVED if configured for only PCI_0)

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	DRAM0Remap	SCS[1:0]* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_1 Base Address registers' enable.)	0x0

Table 191: PCI_0 Swapped SCS[1:0]* Base Address Remap, Offset: 0xc58

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	Swapped DRAM0Remap	Swapped SCS[1:0]* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_0 Base Address registers' enable.)	0x0

Table 192: PCI_1 Swapped SCS[1:0]* Base Address Remap, Offset: 0xcd8 (RESERVED if configured for only PCI_0)

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	Swapped DRAM0Remap	Swapped SCS[1:0]* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_1 Base Address registers' enable.)	0x0

Table 193: PCI_0 SCS[3:2]* Base Address Remap, Offset: 0xc4c

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	DRAM1Remap	SCS[3:2]* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_0 Base Address registers' enable.)	0x01000

Table 194: PCI_1 SCS[3:2]* Base Address Remap, Offset: 0xccc (RESERVED if configured for only PCI_0)

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	DRAM1Remap	SCS[3:2]* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_1 Base Address registers' enable.)	0x01000

Table 195: PCI_0 Swapped SCS[3:2]* Base Address Remap, Offset: 0xc5c

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	Swapped DRAM1Remap	Swapped SCS[3:2]* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_0 Base Address registers' enable.)	0x01000

Table 196: PCI_1 Swapped SCS[3:2]* Base Address Remap, Offset: 0xcdc (RESERVED if configured for only PCI_0)

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	Swapped DRAM1Remap	Swapped SCS[3:2]* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_1 Base Address registers' enable.)	0x01000

Table 197: PCI_0 CS[2:0]* Base Address Remap, Offset: 0xc50

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	Dev0Remap	CS[2:0]* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_0 Base Address registers' enable.)	0x1c000

Table 198: PCI_1 CS[2:0]* Base Address Remap, Offset: 0xcd0 (RESERVED if configured for only PCI_0)

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	Dev0Remap	CS[2:0]* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_1 Base Address registers' enable.)	0x1c000

Table 199: PCI_0 CS[3]* & BootCS* Base Address Remap, Offset: 0xc54

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	Dev1Remap	CS[3]* & BootCS* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_0 Base Address registers' enable.)	0xff000

Table 200: PCI_1 CS[3]* & BootCS* Base Address Remap, Offset: 0xcd4 (RESERVED if configured for only PCI_0)

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	Dev1Remap	CS[3]* & BootCS* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_1 Base Address registers' enable.)	0xff000

Table 201: PCI_0 Swapped CS[3]* & BootCS* Base Address Remap, Offset: 0xc64

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	Swapped Dev1Remap	Swapped CS[3]* & BootCS* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_0 Base Address registers' enable.)	0xff000

Table 202: PCI_1 Swapped CS[3]* & BootCS* Base Address Remap, Offset: 0xce4 (RESERVED if configured for only PCI_0)

Bits	Field name	Function	Initial Value
11:0	Reserved	Read only.	0x0
31:12	Swapped Dev1Remap	Swapped CS[3]* & BootCS* memory address remap from PCI side. (RESERVED if this BAR is disabled through PCI_1 Base Address registers' enable.)	0xff000

Table 203: PCI_0 Configuration Address, Offset: 0xcf8

Bits	Field Name	Function	Initial Value
1:0	Reserved	Read only.	0x0
7:2	RegNum	Indicates the register number.	0x00
10:8	FunctNum	Indicates the function number.	0x0
15:11	DevNum	Indicates the device number.	0x00
23:16	BusNum	Indicates the 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 204: PCI_0 Configuration Data, Offset: 0xcfc

Bits	Field Name	Function	Initial Value
31:0	Config	The data is transferred to or 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-64130 to perform a Configuration or Special cycle on the PCI bus.	0x000

Table 205: PCI_1 Configuration Address, Offset: 0xcf0 (RESERVED if configured for only PCI_0)

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 Configuration Address.	0x000

Table 206: PCI_1 Configuration Data, Offset: 0xcf4 (RESERVED if configured for only PCI_0)

Bits	Field Name	Function	Initial Value
31:0	Config	Same as for PCI_0 Configuration Data.	0x000

Table 207: PCI_0 Interrupt Acknowledge Virtual Register, Offset: 0xc34

Bits	Field Name	Function	Initial Value
31:0	IntAck_0	Read only. A CPU read access to this register forces an interrupt acknowledge cycle on PCI_0.	0x0

Table 208: PCI_1 Interrupt Acknowledge Virtual Register, Offset: 0xc30

Bits	Field Name	Function	Initial Value
31:0	IntAck_1	Read only. A CPU read access to this register forces an interrupt acknowledge cycle on PCI_1.	0x0

19.16 Interrupts

Table 209: Interrupt Cause Register - Offset: 0xc18
(all bits are cleared by writing a value of '0' by the CPU or PCI, unless stated otherwise)

Bits	Field Name	Function	Initial Value
0	IntSum	Interrupt summary Logical OR of all the interrupt bits, regardless of the Mask registers' values.	0x0 Read only
1	MemOut	Asserts when the CPU or PCI accesses an address out of range in the memory decoding or a burst access to 8-/16-bit devices.	0x0
2	DMAOut	Asserts when the DMA accesses an address out of range.	0x0
3	CPUOut	Asserts when the CPU accesses an address out of range.	0x0
4	DMA0Comp	Asserts at completion of DMA Channel 0 transfer.	0x0
5	DMA1Comp	Asserts at completion of DMA Channel 1 transfer.	0x0

Table 209: Interrupt Cause Register - Offset: 0xc18 (Continued)

(all bits are cleared by writing a value of '0' by the CPU or PCI, unless stated otherwise)

Bits	Field Name	Function	Initial Value
6	DMA2Comp	Asserts at completion of DMA Channel 2 transfer.	0x0
7	DMA3Comp	Asserts at completion of DMA Channel 3 transfer.	0x0
8	T0Exp	Asserts when Timer 0 expires.	0x0
9	T1Exp	Asserts when Timer 1 expires.	0x0
10	T2Exp	Asserts when Timer 2 expires.	0x0
11	T3Exp	Asserts when Timer 3 expires.	0x0
12	MasRdErr0	Asserts when the GT-64130 detects a parity error during a PCI_0 master read operation.	0x0
13	SlvWrErr0	Asserts when the GT-64130 detects a parity error during a PCI_0 slave write operation.	0x0
14	MasWrErr0	Asserts when the GT-64130 detects a parity error during a PCI_0 master write operation.	0x0
15	SlvRdErr0	Asserts when the GT-64130 detects a parity error during a PCI_0 slave read operation.	0x0
16	AddrErr0	Asserts when the GT-64130 detects a parity error on the PCI_0 address line.	0x0
17	MemErr	Asserts when a memory parity error is detected.	0x0
18	MasAbort0	Asserts upon PCI_0 master abort.	0x0
19	TarAbort0	Asserts upon PCI_0 target abort.	0x0
20	RetryCtr0	Asserts when the PCI_0 retry counter expires.	0x0
25:21	CPUInt	These bits are set by the CPU by writing 0 to generate an interrupt on the PCI bus. They are cleared when the PCI writes 0.	0x0
29:26	PCIInt	These bits are set by the PCI by writing 0 to generate an interrupt on the CPU. They are cleared when the CPU writes 0.	0x0

Table 209: Interrupt Cause Register - Offset: 0xc18 (Continued)

(all bits are cleared by writing a value of '0' by the CPU or PCI, unless stated otherwise)

Bits	Field Name	Function	Initial Value
30	CPUIntSum	CPU Interrupt Summary Logical OR of bits[29:26,20:1], masked by bits[29:26,20:1] of the CPU Mask register.	0x0
31	PCIIntSum	Interrupt Summary Logical OR of bits[25:1], masked by bits[25:1] of the PCI_0 Mask register.	0x0

Table 210: High Interrupt Cause Register Offset: 0xc98 (RESERVED if configured for only PCI_0)

Bits	Field Name	Function	Initial Value
11:0		Read only.	0x0
12	MasRdErr1	Asserts when the GT-64130 detects a parity error during a PCI_1 master read operation.	0x0
13	SlvWrErr1	Asserts when the GT-64130 detects a parity error during a PCI_1 slave write operation.	0x0
14	MasWrErr1	Asserts when the GT-64130 detects a parity error during a PCI_1 master write operation.	0x0
15	SlvRdErr1	Asserts when the GT-64130 detects a parity error during a PCI_1 slave read operation.	0x0
16	AddrErr1	Asserts when the GT-64130 detects a parity error on the PCI_1 address lines.	0x0
17	Reserved	Read only.	0x0
18	MasAbort1	Asserts upon PCI_1 master abort.	0x0
19	TarAbort1	Asserts upon PCI_1 target abort.	0x0
20	RetryCtr1	Asserts when the PCI_1 retry counter expires.	0x0
31:21	Reserved	Read only.	0x0

Table 211: CPU Interrupt Select - Offset: 0xc70 (RESERVED if configured for only PCI_0)

Bits	Field Name	Function	Initial Value
0	Reserved	Read only.	0x0
29:1	AliasedBits	Aliased to bits [29:1] of the selected cause register.	0x0
30	SelectCause	Selected Cause Register 0 - Low cause register 1 - High cause register	0x0
31	LowAndHigh	Interrupt is both Low and High Cause registers.	0x0

Table 212: PCI_0 Interrupt Select - Offset: 0xc74 (RESERVED if configured for only PCI_0)

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

19.16.1 Interrupt MASK Registers

Table 213: CPU Interrupt Mask Register, Offset: 0xc1c

Bits	Field Name	Function	Initial Value
0	Reserved	Read only.	0x0
1	MemOutMask	Masks MemOut Interrupt to CPU.	0x0
2	DMAOutMask	Masks DMAOut Interrupt to CPU.	0x0
3	CPUOutMask	Masks CPUOut Interrupt to CPU.	0x0
4	DMA0CompMask	Masks DMA0Comp Interrupt to CPU.	0x0
5	DMA1CompMask	Masks DMA1Comp Interrupt to CPU.	0x0
6	DMA2CompMask	Masks DMA2Comp Interrupt to CPU.	0x0
7	DMA3CompMask	Masks DMA3Comp Interrupt to CPU.	0x0
8	T0ExpMask	Masks T0Exp Interrupt to CPU.	0x0
9	T1ExpMask	Masks T1Exp Interrupt to CPU.	0x0
10	T2ExpMask	Masks T2Exp Interrupt to CPU.	0x0
11	T3ExpMask	Masks T3Exp Interrupt to CPU.	0x0
12	MasRdErr0Mask	Masks MasRdErr0 Interrupt to CPU.	0x0
13	SlvWrErr0Mask	Masks SlvWrErr0 Interrupt to CPU.	0x0

Table 213: CPU Interrupt Mask Register, Offset: 0xc1c (Continued)

Bits	Field Name	Function	Initial Value
14	MasWrErr0Mask	Masks MasWrErr0 Interrupt to CPU.	0x0
15	SlvRdErr0Mask	Masks SlvRdErr0 Interrupt to CPU.	0x0
16	AddrErr0Mask	Masks AddrErr0 Interrupt to CPU.	0x0
17	MemErrMask	Masks MemErr Interrupt to CPU.	0x0
18	MasAbort0Mask	Masks MasAbort0 Interrupt to CPU.	0x0
19	TarAbort0Mask	Masks TarAbort0 Interrupt to CPU.	0x0
20	RetryCtr0Mask	Masks RetryCtr0 Interrupt to CPU.	0x0
25:21	Reserved	Read only.	0x0
29:26	PCIIntMask	Masks PCIInt Interrupt to CPU.	0x0
31:30	Reserved	Read only.	0x0

Table 214: CPU HIGH Interrupt Cause Mask Offset: 0xc9c (RESERVED if configured for only PCI_0)

Bits	Field Name	Function	Initial Value
11:0	Reserved	Read only.	0x0
12	MasRdErr1Mask	Masks MasRdErr1 to CPU.	0x0
13	SlvWrErr1Mask	Masks SlvWrErr1 to CPU.	0x0
14	MasWrErr1Mask	Masks MasWrErr1 to CPU.	0x0
15	SlvRdErr1Mask	Masks SlvRdErr1 to CPU.	0x0
16	AddrErr1Mask	Masks AddrErr1 to CPU.	0x0
17	Reserved	Reserved.	0x0
18	MasAbort1Mask	Masks MasAbort1 to CPU.	0x0
19	TarAbort1Mask	Masks TarAbort1 to CPU.	0x0
20	RetryCtr1Mask	Masks RetryCtr1 to CPU.	0x0
31:21	Reserved	Read only.	0x0

Table 215: PCI_0 Interrupt Cause Mask, Offset: 0xc24

Bits	Field Name	Function	Initial Value
0	Reserved	Read only.	0x0
1	MemOutMask	Masks MemOut Interrupt to PCI_0.	0x0

Table 215: PCI_0 Interrupt Cause Mask, Offset: 0xc24 (Continued)

Bits	Field Name	Function	Initial Value
2	DMAOutMask	Masks DMAOut Interrupt to PCI_0.	0x0
3	CPUOutMask	Masks CPUOut Interrup to PCI_0t.	0x0
4	DMA0CompMask	Masks DMA0Comp Interrupt to PCI_0.	0x0
5	DMA1CompMask	Masks DMA1Comp Interrupt to PCI_0.	0x0
6	DMA2CompMask	Masks DMA2Comp Interrupt to PCI_0.	0x0
7	DMA3CompMask	Masks DMA3Comp Interrupt to PCI_0.	0x0
8	T0ExpMask	Masks T0Exp Interrupt to PCI_0.	0x0
9	T1ExpMask	Masks T1Exp Interrupt to PCI_0.	0x0
10	T2ExpMask	Masks T2Exp Interrupt to PCI_0.	0x0
11	T3ExpMask	Masks T3Exp Interrupt to PCI_0.	0x0
12	MasRdErr0Mask	Masks MasRdErr0 Interrupt to PCI_0.	0x0
13	SlvWrErr0Mask	Masks SlvWrErr0 Interrupt to PCI_0.	0x0
14	MasWrErr0Mask	Masks MasWrErr0 Interrupt to PCI_0.	0x0
15	SlvRdErr0Mask	Masks SlvRdErr0 Interrupt to PCI_0.	0x0
16	AddrErr0Mask	Masks AddrErr0 Interrupt to PCI_0.	0x0
17	MemErrMask	Masks MemErr Interrupt to PCI_0.	0x0
18	MasAbort0Mask	Masks MasAbort0 Interrupt to PCI_0.	0x0
19	TarAbort0Mask	Masks TarAbort0 Interrupt to PCI_0.	0x0
20	RetryCtr0Mask	Masks RetryCtr0 Interrupt to PCI_0.	0x0
25:21	CPUInt	Masks CPUInt Interrupt to PCI_0.	0x0
31:26	Reserved	Read only.	0x0

Table 216: PCI_0 HIGH Interrupt Cause Mask Offset: 0xca4 (RESERVED if configured for only PCI_0)

Bits	Field Name	Function	Initial Value
11:0	Reserved	Read only.	0x0
12	MasRdErr1Mask	Masks MasRdErr1 to PCI_0.	0x0
13	SlvWrErr1Mask	Masks SlvWrErr1 to PCI_0.	0x0
14	MasWrErr1Mask	Masks MasWrErr1 to PCI_0.	0x0
15	SlvRdErr1Mask	Masks SlvRdErr1 to PCI_0.	0x0

Table 216: PCI_0 HIGH Interrupt Cause Mask Offset: 0xca4 (RESERVED if configured for only PCI_0)

Bits	Field Name	Function	Initial Value
16	AddrErr1Mask	Masks AddrErr1 to PCI_0.	0x0
17	Reserved	Reserved.	0x0
18	MasAbort1Mask	Masks MasAbort1 to PCI_0.	0x0
19	TarAbort1Mask	Masks TarAbort1 to PCI_0.	0x0
20	RetryCtr1Mask	Masks RetryCtr1 to PCI_0.	0x0
31:21	Reserved	Read only.	0x0

Table 217: SErr0* Mask, PCI_0 Events Offset: 0xc28

Bits	Field Name	Function	Initial Value
0	AddrErr0	Mask bit. When set, SErr0* is asserted when the GT-64130 detects a parity error on PCI_0 address lines.	0x0
1	MasWrErr0	Mask bit. When set, SErr0* is asserted when the GT-64130 detects a parity error during a PCI_0 master write operation.	0x0
2	MasRdErr0	Mask bit. When set, SErr0* is asserted when the GT-64130 detects a parity error during a PCI_0 master read operation.	0x0
3	MemErr	Mask bit. When set, SErr0* is asserted when a memory parity error has been detected .	0x0
4	MasAbor0t	Mask bit. When set, SErr0* is asserted when the GT-64130 performs PCI_0 master abort.	0x0
5	TarAbort0	Mask bit. When set, SErr0* is asserted when the GT-64130 detects aPCI_0 target abort.	0x0
31:6	Reserved		0x0

Table 218: SErr1* Mask, PCI_1 Events Offset: 0xca8 (RESERVED if configured for only PCI_0)

Bits	Field Name	Function	Initial Value
0	AddrErr1	Mask bit. When set, SErr1* is asserted when the GT-64130 detects a parity error on PCI_1 address lines.	0x0
1	MasWrErr1	Mask bit. When set, SErr1* is asserted when the GT-64130 detects a parity error during a PCI_1 master write operation.	0x0
2	MasRdErr1	Mask bit. When set, SErr1* is asserted when the GT-64130 detects a parity error during a PCI_1 master read operation.	0x0
3	MemErr	Mask bit. When set, SErr1* is asserted when a memory parity error has been detected.	0x0
4	MasAbort1	Mask bit. When set, SErr1* is asserted when the GT-64130 performs PCI_1 master abort.	0x0
5	TarAbort1	Mask bit. When set, SErr1* is asserted when the GT-64130 detects a PCI_1 target abort.	0x0
31:6	Reserved		0x0

19.17 PCI Configuration

All PCI Configuration registers are located at their standard offset when accessed from their corresponding PCI bus. For example, if a master on PCI_0 performs a PCI configuration cycle on PCI_0's Status and Command register, the register will be located at 0x004. Likewise, if a master on PCI_1 performs a PCI configuration cycle on PCI_1's Status and Command register, the register will be located at 0x004.

On the other hand, if a master on PCI_0 performs a PCI configuration cycle on PCI_1's Status and Command register, the register will be located at 0x084. Likewise, if a master on PCI_1 performs a PCI configuration cycle on PCI_0's Status and Command register, the register will be located at 0x084.

If the CPU masters PCI configuration cycles, PCI_0 configuration registers are located at their standard offsets and PCI_1 configuration registers are located at 0x80 + standard offset.

NOTE: All PCI_1 configuration registers are reserved if the GT-64130 is configured for only PCI_0 operation.

Table 219: PCI_0 Device and Vendor ID, Offset: 0x000 from PCI_0 or CPU; 0x080 from PCI_1

Bits	Field name	Function	Initial Value
15:0	VenID	Read only. Provides the Galileo Technology vender ID number.	0x11ab
31:16	DevID	Read only. Provides the unique GT-64130 PCI device ID number.	0x6320

Table 220: PCI_1 Device and Vendor ID, Offset: 0x080 from PCI_0 or CPU; 0x000 from PCI_1

Bits	Field name	Function	Initial Value
31:0	Various	Same as for PCI_0 Device and Vendor ID.	0x632011ab

Table 221: PCI_0 Status and Command, Offset: 0x004 from PCI_0 or CPU; 0x084 from PCI_1

Bits	Field name	Function	Initial Value
0	IOEn	Controls the GT-64130's response to I/O accesses. 0 - Disable 1 - Enable	0x0
1	MEMEn	Controls the GT-64130's response to memory accesses. 0 - Disable 1 - Enable	0x0
2	MasEn	Controls the GT-64130's ability to act as a master on the PCI bus. 0 - Disable 1 - Enable	0x0
3	Reserved	Read only.	0x0
4	MemWrInv	Controls the GT-64130's ability to generate Memory Write & Invalidate commands on the PCI bus. 0 - Disable 1 - Enable	0x0
5	Reserved	Read only.	0x0

Table 221: PCI_0 Status and Command, Offset: 0x004 from PCI_0 or CPU; 0x084 from PCI_1 (Continued)

Bits	Field name	Function	Initial Value
6	PErrEn	Controls the GT-64130's ability to respond to parity errors on the PCI by asserting the PErr* pin. 0 - Disable 1 - Enable	0x0
7	Reserved	Read only.	0x0
8	SErrEn	Controls the GT-64130's ability to assert the SErr* pin. 0 - Disable 1 - Enable	0x0
20:9	Reserved	Read only.	0x0
21	66MHzEn	66MHz Capable. The GT-64130 PCI interface is capable of running at 66MHz regardless of this bit value.	0x1
22	Reserved	Read only.	0x0
23	TarFastBB	Read only. Indicates that the GT-64130 is capable of accepting fast back-to-back transactions on the PCI bus.	0x1
24	DataParDet	This bit is set by the GT-64130 when it detects a data parity error during master operation.	0x0
26:25	DevSelTim	These pins indicate the GT-64130 's DevSel timing (medium), per the PCI standard.	0x1 Read only.
27	Reserved	Read only.	0x0
28	TarAbort	This bit is set upon Target Abort.	0x0
29	MasAbort	This pin is set upon Master Abort.	0x0
30	SysErr	This pin is set upon System Error.	0x0
31	DetParErr	This pin is set upon detection of Parity error (in master and slave operations).	0x0

NOTE: For bits 24 and 28 through 31, the user cannot set the bit. The user can only clear the bit by writing 1 to it.

Table 222: PCI_1 Status and Command, Offset: 0x084 from PCI_0 or CPU; 0x004 from PCI_1

Bits	Field name	Function	Initial Value
31:0	Various	Same as for PCI_0 Status and Command.	0x2800000

Table 223: PCI_0 Class Code and Revision ID, Offset: 0x008 from PCI_0 or CPU; 0x088 from PCI_1

Bits	Field name	Function	Initial Value
7:0	RevID	Indicates the GT-64130 PCI0 Revision number.	0x01
15:8	Reserved	Read only.	0x0
23:16	SubClass	Indicates the GT-64130 Subclass 0x00 - Host Bridge Device. 0x80 - Memory Device.	Depends on the value sampled at reset on BankSel[0].
31:24	BaseClass	Indicates the GT-64130 Base Class. 0x06 - Bridge Device. 0x05 - Memory Device.	Depends on the value sampled at reset on BankSel[0].

Table 224: PCI_1 Class Code and Revision ID, Offset: 0x088 from PCI_0 or CPU; 0x008 from PCI_1

Bits	Field name	Function	Initial Value
31:0	Various	Same as for PCI_0 Class Code and Revision ID.	

Table 225: PCI_0 BIST, Header Type, Latency Timer, Cache Line, Offset: 0x00c from PCI_0 or CPU; 0x08c from PCI_1

Bits	Field name	Function	Initial Value
7:0	CacheLine	Specifies the GT-64130's cache line size.	0x00
15:8	LatTimer	Specifies in units of PCI bus clocks the value of the latency timer of the GT-64130.	0x00

Table 225: PCI_0 BIST, Header Type, Latency Timer, Cache Line, Offset: 0x00c from PCI_0 or CPU; 0x08c from PCI_1 (Continued)

Bits	Field name	Function	Initial Value
23:16	HeadType	Specifies the layout of bytes 10h through 3fh.	0x00
31:24	BIST	Built In Self Test. Reserved.	0x00

Table 226: PCI_1 BIST, Header Type, Latency Timer, Cache Line, Offset: 0x08c from PCI_0 or CPU; 0x00c from PCI_1

Bits	Field name	Function	Initial Value
31:0	Various	As in PCI_0 BIST, Header Type, Latency Timer, Cache Line.	0x00

The BIST Field is reserved and is hardwired to 0.

Device and Vendor ID (0x000), Class Code and Revision ID (0x008), and Header Type (0x00e) fields are read only from the PCI bus. These fields can be modified and read via the CPU bus.

For more information on these fields, refer to the PCI specification.

Access of PCI masters to SDRAM banks, Devices and internal space is achieved once there is a match between the address presented over the PCI bus and the space defined by the respective Base/Size register pair. The GT-64130 incorporates three Swapped Base Address registers for SCS[1:0]*, SCS[3:2]* and CS[3]* & BootCS*. When the address matches a Swapped Base Address register, the data transferred will undergo the opposite to what is indicated by the ByteSwap bit (bit[0] of 0xc00). By using this mechanism, one could write data directly to SDRAM and read it byte-swapped without CPU processing.

NOTE: For the address presented over the PCI bus, and the space defined by the respective Base/Size register pair, to match the Swapped Base Address register, the address must not match its respective non-Swap Base Address register.

The Size registers could not define a zero size space. In order to enable the system designer to use addresses which are within a certain space without having the GT-64130 respond to these addresses, a Base Address Enable register is incorporated. A disabled space will not trigger device response should the address fall within the space defined by its Base/Size register pair.

Table 227: PCI_0 SCS[1:0]* Base Address, Offset: 0x010 from PCI_0 or CPU; 0x090 from PCI_1

Bits	Field Name	Function	Initial Value
0	MemSpace	Memory Space Indicator Read only.	0x0
2:1	Type	Type Read only.	0x0
3	Prefetch	Prefetch Read only from PCI.	0x1
11:4	Reserved	Read only.	0x0
31:12	Base	Defines the address assignment of SCS[1:0]*. See SCS[1:0]* Bank Size register in Table 177 on page 202. RESERVED if this BAR is disabled through PCI_0 Base Address registers' Enable.	0x00000

Table 228: PCI_1 SCS[1:0]* Base Address, Offset: 0x090 from PCI_0 or CPU; 0x010 from PCI_1

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 SCS[1:0]* Base Address.	0x04

Table 229: PCI_0 SCS[3:2]* Base Address, Offset: 0x014 from PCI_0 or CPU; 0x094 from PCI_1

Bits	Field Name	Function	Initial Value
0	MemSpace	Memory Space Indicator Read only.	0x0
2:1	Type	Type Read only.	0x0
3	Prefetch	Prefetch Read only from PCI.	0x1

Table 229: PCI_0 SCS[3:2]* Base Address, Offset: 0x014 from PCI_0 or CPU; 0x094 from PCI_1 (Continued)

Bits	Field Name	Function	Initial Value
11:4	Reserved	Read only.	0x0
31:12	Base	Defines the address assignment of SCS[3:2]*. See the SCS[3:2]* Bank Size register in Table 179 on page 203. RESERVED if this BAR is disabled through PCI_0 Base Address Registers' Enable.	0x01000

Table 230: PCI_1 SCS[3:2]* Base Address, Offset: 0x094 from PCI_0 or CPU; 0x014 from PCI_1

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 SCS[3:2]* Base Address.	0x01000004

Table 231: PCI_0 CS[2:0]* Base Address, Offset: 0x018 from PCI_0 or CPU; 0x098 from PCI_1

Bits	Field Name	Function	Initial Value
0	MemSpace	Memory Space Indicator Read only.	0x0
2:1	Type	Type Read only.	0x0
3	Prefetch	Prefetch Read only from PCI.	0x0
11:4	Reserved	Read only.	0x0
31:12	Base	Defines the address assignment of CS[2:0]*. See the CS[2:0]* Bank Size register in Table 181 on page 203. RESERVED if this BAR is disabled through PCI_0 Base Address registers' Enable.)	0x1c000

Table 232: PCI_1 CS[2:0]* Base Address, Offset: 0x098 from PCI_0 or CPU; 0x018 from PCI_1

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 CS[2:0]* Base Address.	ox1c000000

Table 233: PCI_0 CS[3]* and BootCS* Base Address, Offset: 0x01c from PCI_0 or CPU; 0x09c from PCI_1

Bits	Field Name	Function	Initial Value
0	MemSpace	Memory Space Indicator Read only.	0x0
2:1	Type	Type Read only.	0x0
3	Prefetch	Prefetch Read only from PCI.	0x0
11:4	Reserved	Read only.	0x0
31:12	Base	Defines the address assignment of CS[3]* and BootCS* (see CS[3]* and BootCS* Bank Size. RESERVED if this BAR is disabled through PCI_0 Base Address registers' Enable.)	0xff000

Table 234: PCI_1 CS[3]* and BootCS* Base Address, Offset: 0x09c from PCI_0 or CPU; 0x01c from PCI_1

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 CS[3]* and BootCS* Base Address.	0xff000000

Table 235: PCI_0 Internal Registers Memory Mapped Base Address, Offset: 0x020 from PCI_0 or CPU; 0x0a0 from PCI_1

Bits	Field Name	Function	Initial Value
0	MemSpace	Memory Space Indicator Read only.	0x0
2:1	Type	Type Read only.	0x0
3	Prefetch	Prefetch Read only.	0x0

Table 235: PCI_0 Internal Registers Memory Mapped Base Address, Offset: 0x020 from PCI_0 or CPU; 0x0a0 from PCI_1 (Continued)

Bits	Field Name	Function	Initial Value
11:4	Reserved	Read only.	0x0
31:12	MemMapBase	Defines the address assignment of the GT-64130's internal registers. RESERVED if this BAR is disabled through PCI_0 Base Address registers' Enable.	0x14000

Table 236: PCI_1 Internal Registers Memory Mapped Base Address, Offset: 0x0a0 from PCI_0 or CPU; 0x020 from PCI_1

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 Internal Registers Memory Mapped Base Address.	0x14000000

Table 237: PCI_0 Internal Registers I/O Mapped Base Address, Offset: 0x024 from PCI_0 or CPU; 0x0a4 from PCI_1

Bits	Field Name	Function	Initial Value
0	MemSpace	IO Space Indicator	0x1
11:1	Reserved	Read only.	0x0
31:12	IOMapBase	Defines the address assignment of the GT-64130's internal registers. RESERVED if this BAR is disabled through PCI_0 Base Address registers' Enable.	0x14000

Table 238: PCI_1 Internal Registers I/O Mapped Base Address, Offset: 0x0a4 from PCI_0 or CPU; 0x024 from PCI_1

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 Internal Registers I/O Mapped Base Address.	0x14000001

Table 239: PCI_0 Subsystem Device and Vendor ID, Offset: 0x02c from PCI_0 or CPU; 0x0ac from PCI_1

Bits	Field name	Function	Initial Value
15:0	VenID	Subsystem Vendor ID	0x0
31:16	DevID	Subsystem Device ID	0x0

Table 240: PCI_1 Subsystem Device and Vendor ID, Offset: 0x0ac from PCI_0 or CPU; 0x02c from PCI_1

Bits	Field name	Function	Initial Value
31:0	Various	Same as for PCI_0 Subsystem Device and Vendor ID.	0x0

Table 241: Expansion ROM Base Address Register, Offset: 0x030 from PCI_0 or CPU; 0x0b0 from PCI_1

Bits	Field Name	Function	Initial Value
0	ERDecEn	Expansion ROM Decode Enable 0 - Disable 1 - Enable	0x0
11:1	Reserved		0x0
31:12	ERBase	Defines the address of the expansion ROM memory space region assigned to the GT-64130. This is where the expansion ROM code appears in system memory when bit 0 of this register contains a value of 1 and bit 1 of this device's Command register contains a value of 1. This register is Reserved in case Expansion ROM was not enabled at Reset (DAdr[5] sampled 0).	0xff000

Table 242: PCI_0 Interrupt Pin and Line, Offset: 0x03c from PCI_0 or CPU; 0x0bc from PCI_1

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-64130. The GT-64130 uses INTA.	0x1
31:16	Reserved	Read only.	0x0

Table 243: PCI_1 Interrupt Pin and Line, Offset: 0x0bc from PCI_0 or CPU; 0x03c from PCI_1

Bits	Field name	Function	Initial Value
31:0	Various	Same as for PCI_0 Interrupt Pin and Line.	0x00000100

19.17.1 Function 1 Configuration Registers

The GT-64130 acts as a two function device. It's PCI slave interface responds to configuration transactions to function number 0 or 1.

Most of function 1 configuration registers are aliased to function 0 registers, except of the 3 swap BARs. To access any of the Swap Base Address Registers, a configuration access addressed to function 1 should be used with the appropriate offset. If an offset other than 0x010, 0x014, or 0x01c is accessed when specifying function 1, the transaction will access the corresponding offset register in function 0. Configuration transactions to any other function number are ignored.

The GT-64130 acts as two function devices regardless of multi-function bit (bit[7] in Header Type). However, this bit value after reset is 0.

In a PC environment, in order for a BIOS to recognize the GT-64130 as a multifunction device (if swap BARs are required in the system), set this bit as well as enable the swap BARs (Base Address Enable register) before BIOS starts. This can be done by programing CPU software or by using GT-64130 Auto-Load option.

Table 244: Function1 PCI_0 Swapped SCS[1:0]* Base Address, Offset: 0x110 from PCI_0 or CPU; 0x190 from PCI_1

Bits	Field Name	Function	Initial Value
0	MemSpace	Memory Space Indicator Read only.	0x0
2:1	Type	Type Read only.	0x0
3	Prefetch	Prefetch Read only from PCI.	0x1
11:4	Reserved	Read only.	0x0
31:12	Base	Defines the address assignment of Swapped SCS[1:0]*. RESERVED if this BAR is disabled through PCI_0 Base Address registers' Enable.	0x0

Table 245: Function 1 PCI_1 Swapped SCS[1:0]* Base Address, Offset: 0x190 from PCI_0 or CPU; 0x110 from PCI_1

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 Swapped SCS[1:0]* Base Address.	0x04

Table 246: Function 1 PCI_0 Swapped SCS[3:2]* Base Address, Offset: 0x114 from PCI_0 or CPU; 0x194 from PCI_1

Bits	Field Name	Function	Initial Value
0	MemSpace	Memory Space Indicator Read only.	0x0
2:1	Type	Type Read only.	0x0
3	Prefetch	Prefetch Read only from PCI.	0x1
11:4	Reserved	Read only.	0x0
31:12	Base	Defines the address assignment of Swapped SCS[3:2]*. RESERVED if this BAR is disabled through PCI_0 Base Address registers' Enable.	0x01000

Table 247: Function 1 PCI_1 Swapped SCS[3:2]* Base Address, Offset: 0x194 from PCI_0 or CPU; 0x114 from PCI_1

Bits	Field Name	Function	Initial Value
31:0	Various	Same as for PCI_0 Swapped SCS[3:2]* Base Address.	0x01000004

Table 248: Function 1 PCI_0 Swapped CS[3]* & BootCS* Base Address, Offset: 0x11c from PCI_0 or CPU; 0x19c from PCI_1

Bits	Field Name	Function	Initial Value
0	MemSpace	Memory Space Indicator Read only.	0x0
2:1	Type	Type Read only.	0x0

Table 248: Function 1 PCI_0 Swapped CS[3]* & BootCS* Base Address, Offset: 0x11c from PCI_0 or CPU; 0x19c from PCI_1 (Continued)

Bits	Field Name	Function	Initial Value
3	Prefetch	Prefetch Read only from PCI.	0x0
11:4	Reserved	Read only.	0x0
31:12	Base	Defines the address assignment of CS[3]* and BootCS*. See CS[3]* and BootCS* Bank Size register in Table 183 on page 204. RESERVED if this BAR is disabled through PCI_0 Base Address registers' Enable.	0xff000

Table 249: Function 1 PCI_1 Swapped CS[3]* & BootCS* Base Address, Offset: 0x19c from PCI_0 or CPU; 0x11c from PCI_1

Bits	Field Name	Function	Initial Value
31:0	Various	As in PCI_0 Swapped CS[3]* and BootCS* Base Address.	0xff000000

NOTE: For more information on these fields, refer to the PCI specification.

19.18 I_2O Support Registers

If I_2O is enabled (i.e., DAdr[8] was sampled '0' at reset), then its related registers can be accessed from the CPU side and the PCI_0 side.

To access registers from the CPU side, the address must be in the 4KBytes of CPU Internal Space Base register space.

PCI_1 has no access to I_2O registers. If PCI_1 address hits first 4KBytes of PCI_1 SCS[1:0]* BAR space, it accesses SDRAM.

To access registers from the PCI_0 side, the address must be in the first 4KBytes of PCI_0 SCS[1:0]* Base register space.

If accessed from PCI_0 side, the address offset is with respect to the PCI_0 SCS[1:0]* Base Address register contents. If accessed from CPU side, the address offset is with respect to the CPU Internal Space Base Register + 0x1c00.

Table 250: Inbound Message Register 0, Offset: 0x10

Bits	Field Name	Function	Initial Value
31:0	InMsg0	Inbound Message Register 0 Read only from CPU. When written, a bit is set in the Inbound Interrupt Cause register and an interrupt is generated to the CPU (if unmasked). First register of two intended for messages from PCI to CPU.	0x0

Table 251: Inbound Message Register 1, Offset: 0x14

Bits	Field Name	Function	Initial Value
31:0	InMsg1	Inbound Message Register 1 Read only from CPU. When written, a bit is set in the Inbound Interrupt Cause register and an interrupt is generated to the CPU (if unmasked). Second register of two intended for messages from PCI to CPU.	0x0

Table 252: Outbound Message Register 0, Offset: 0x18

Bits	Field Name	Function	Initial Value
31:0	OutMsg0	Outbound Message Register 0 Read only from PCI. When written, a bit is set in the Outbound Interrupt Cause register and an interrupt is generated to the PCI (if unmasked). First register of two intended for messages from CPU to PCI.	0x0

Table 253: Outbound Message Register 1, Offset: 0x1c

Bits	Field Name	Function	Initial Value
31:0	OutMsg1	Outbound Message Register 1 Read only from PCI. When written, a bit is set in the Outbound Interrupt Cause register and an interrupt is generated to the PCI (if unmasked). Second register of two intended for messages from CPU to PCI.	0x0

Table 254: Inbound Doorbell Register, Offset: 0x20

Bits	Field Name	Function	Initial Value
31:0	InDoor	Inbound Doorbell Register Setting a bit in this register to 1 by PCI causes a CPU interrupt, if not masked by Inbound Interrupt Mask register. Writing 1 to this bit by a CPU clears the bit (and deassert the interrupt).	0x0

Table 255: Inbound Interrupt Cause Register, Offset: 0x24

Bits	Field Name	Function	Initial Value
0	InMsg0Int	Inbound Message 0 Interrupt This bit is set when Inbound Message 0 register is written. CPU writes it with 1 to clear it.[1]	0x0
1	InMsg1Int	Inbound Message 1 Interrupt This bit is set when Inbound Message 1 register is written. CPU writes it with 1 to clear it.[2]	0x0
2	InDoorInt	Inbound Doorbell Interrupt Read only. This bit is set when at least one bit of Inbound Doorbell register is set.	0x0
3	Reserved		0x0
4	InPQInt	Inbound Post Queue Interrupt This bit is set when Inbound Post Queue gets written. CPU writes it with 1 to clear it.	0x0

Table 255: Inbound Interrupt Cause Register, Offset: 0x24 (Continued)

Bits	Field Name	Function	Initial Value
5	OutFQOvr	Outbound Free Queue Overflow Interrupt This bit is set when Outbound Free Queue is full. CPU writes it with 1 to clear it.	0x0
31:6	Reserved		0x0

1. Unlike Intel's i960RP, bits [8:6] and bit 3 are reserved since GT-64130 does not support Index, APIC, or NMI mechanisms.

2. An interrupt to CPU is generated if any of the bits 0,1,2,4, or 5 is set to 1 given that its corresponding entry in the Inbound Interrupt Mask Register is NOT set.

Table 256: Inbound Interrupt Mask Register, Offset: 0x28

Bits	Field Name	Function	Initial Value
0	InMsg0IntMsk	Inbound Message 0 Interrupt Mask	0x0
1	InMsg1IntMsk	Inbound Message 1 Interrupt Mask	0x0
2	InDoorIntMsk	Inbound Doorbell Interrupt Mask	0x0
3	Reserved		0x0
4	InPQIntMsk	Inbound Post Queue Interrupt Mask	0x0
5	OutFQOvrMsk	Outbound Free Queue Overflow Interrupt Mask When set, no interrupt to CPU is generated for set OutFQOvr bit in Inbound Interrupt Cause register.	0x0
31:6	Reserved		0x0

Table 257: Outbound Doorbell Register, Offset: 0x2c

Bits	Field Name	Function	Initial Value
31:0	OutDoor	Outbound Doorbell Register Setting a bit in this register to 1 by a CPU causes a PCI interrupt, if not masked by Outbound Interrupt Mask register.[1] Writing 1 to this bit by PCI clears the bit (and deassert the interrupt).	0x0

1. Unlike Intel's i960RP, there are No Reserved bits for PCI interrupts INTA#, INTB#, INTC#, INTD#.

Table 258: Outbound Interrupt Cause Register, Offset: 0x30

Bits	Field Name	Function	Initial Value
0	OutMsg0Int	Outbound Message 0 Interrupt This bit is set when Outbound Message 0 register is written. PCI writes it with 1 to clear it. For CPU, it is read only.	0x0
1	OutMsg1Int	Outbound Message 1 Interrupt This bit is set when Outbound Message 1 register is written. PCI writes it with 1 to clear it. For CPU, it is read Only.[1]	0x0
2	OutDoorInt	Outbound Doorbell Interrupt This bit is set when at least one bit of Outbound Doorbell register is set. It is read only.	0x0
3	OutPQInt	Outbound Post Queue Interrupt This bit is set as long as Outbound Post Queue is not empty. It is read only.	0x0
31:4	Reserved		0x0

1. An interrupt to PCI is generated if any of the bits 0,1,2, or 3 is set to 1 given that its corresponding entry in the Outbound Interrupt Mask Register is NOT set.

Table 259: Outbound Interrupt Mask Register, Offset: 0x34

Bits	Field Name	Function	Initial Value
0	OutMsg0IntMsk	Outbound Message 0 Interrupt Mask	0x0
1	OutMsg1IntMsk	Outbound Message 1 Interrupt Mask	0x0
2	OutDoorIntMsk	Outbound Doorbell Interrupt Mask	0x0
3	OutPQIntMsk	Outbound Post Queue Interrupt Mask When set, no interrupt to PCI is generated for set OutPQInt bit in Outbound Interrupt Cause register.	0x0
31:4	Reserved		0x0

Table 260: Inbound Queue Port Virtual Register, Offset: 0x40

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 Inbound Post Queue. A read from this port results in a read from Inbound Free Queue. Reserved from CPU side.	0x0

Table 261: Outbound Queue Port Virtual Register, Offset: 0x44

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 Outbound Free Queue. A read from this port results in a read from Outbound Post Queue. Reserved from CPU side.	0x0

Table 262: Queue Control Register, Offset: 0x50

Bits	Field Name	Function	Initial Value
0	CirQEn	Circular Queue Enable If 0, any PCI write to Queue is ignored. Upon a PCI read from Queue, 0xffffffff is returned. Reserved from PCI side.	0x0
5:1	CirQSize	Circular Queue Size 00001 - 16 Kbytes 00010 - 32 Kbytes 00100 - 64 Kbytes 01000 - 128 Kbytes 10000 - 256 Kbytes Reserved from PCI side.	0x1
31:6	Reserved		0x0

Table 263: Queue Base Address Register, Offset: 0x54

Bits	Field Name	Function	Initial Value
19:0	Reserved		0x0
31:20	QBAR	Queue Base Address Register Reserved from PCI side.	0x0

Table 264: Inbound Free Head Pointer Register, Offset: 0x60

Bits	Field Name	Function	Initial Value
1:0	Reserved		0x0
19:2	InFHPtr	Inbound Free Head Pointer[1] Reserved from PCI side.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

1. This register is maintained by CPU software. It is reserved for PCI accesses.

Table 265: Inbound Free Tail Pointer Register, Offset: 0x64

Bits	Field Name	Function	Initial Value
1:0	Reserved		0x0
19:2	InFTPtr	Inbound Free Tail Pointer[1] Reserved from PCI side.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

1. This register is incremented by GT-64130 after PCI read from Inbound port. It is reserved for PCI accesses.

Table 266: Inbound Post Head Pointer Register, Offset: 0x68

Bits	Field Name	Function	Initial Value
1:0	Reserved		0x0
19:2	InPHPtr	Inbound Post Head Pointer[1] Reserved from PCI side.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

1. This register is incremented by GT-64130 after PCI write to Inbound port. It is reserved for PCI accesses.

Table 267: Inbound Post Tail Pointer Register, Offset: 0x6c

Bits	Field Name	Function	Initial Value
1:0	Reserved		0x0
19:2	InPTPtr	Inbound Post Tail Pointer[1] Reserved from PCI side.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

1. This register is maintained by CPU software. It is reserved for PCI accesses.

Table 268: Outbound Free Head Pointer Register, Offset: 0x70

Bits	Field Name	Function	Initial Value
1:0	Reserved		0x0
19:2	OutFHPtr	Outbound Free Head Pointer[1] Reserved from PCI side.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

1. This register is incremented by GT-64130 after PCI write to Outbound port. It is reserved for PCI accesses.

Table 269: Outbound Free Tail Pointer Register, Offset: 0x74

Bits	Field Name	Function	Initial Value
1:0	Reserved		0x0
19:2	OutFTPtr	Outbound Free Tail Pointer[1] Reserved from PCI side.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

1. This register is maintained by CPU software. It is reserved for PCI accesses.

Table 270: Outbound Post Head Pointer Register, Offset: 0x78

Bits	Field Name	Function	Initial Value
1:0	Reserved		0x0
19:2	OutPHPtr	Outbound Post Head Pointer[1] Reserved from PCI side.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

1. This register is maintained by CPU software. It is reserved for PCI accesses.

Table 271: Outbound Post Tail Pointer Register, Offset: 0x7c

Bits	Field Name	Function	Initial Value
1:0	Reserved		0x0
19:2	OutPTPtr	Outbound Post Tail Pointer[1] Reserved from PCI side.	0x0
31:20	QBAR	Queue Base Address Register Read only.	0x0

1. This register is incremented by GT-64130 after PCI read from Outbound port. It is reserved for PCI accesses.

20. GT-64130 PINOUT TABLE, 388 PIN BGA

NOTE: The table is sorted by ball number.

Table 272: Pinout Table

Ball#	Signal Name	Ball#	Signal Name	Ball#	Signal Name
A01–A26		***B01–B26***		***C01–C26***	
A01	GND	B01	DH[25]	C01	DH[23]
A02	GND	B02	GND	C02	DH[24]
A03	DH[26]	B03	DH[28]	C03	GND
A04	DH[30]	B04	TS*	C04	DH[27]
A05	L2DBG*	B05	L2BG*	C05	DH[29]
A06	A/DL[1]	B06	A/DL[3]	C06	QsAEn2
A07	A/DL[5]	B07	A/DL[7]	C07	A/DL[0]
A08	A/DL[9]	B08	A/DL[10]	C08	A/DL[4]
A09	A/DL[11]	B09	A/DL[13]	C09	A/DL[8]
A10	A/DL[15]	B10	A/DL[17]	C10	A/DL[12]
A11	A/DL[19]	B11	A/DL[21]	C11	A/DL[16]
A12	A/DL[22]	B12	A/DL[24]	C12	A/DL[20]
A13	A/DL[26]	B13	A/DL[28]	C13	A/DL[23]
A14	A/DL[30]]	B14	DP[0]	C14	A/DL[25]
A15	DP[2]	B15	DP[3]	C15	A/DL[29]
A16	DP[4]	B16	DP[6]	C16	DP[1]
A17	L2BR*	B17	BG*	C17	DP[5]
A18	ARTRY*	B18	TT[1]	C18	DBG*
A19	TSIZ[0]	B19	TSIZ[2]	C19	TT[3]
A20	TBST*	B20	L2HIT*	C20	Interrupt*
A21	QsAEn	B21	Req1*	C21	TA*
A22	Gnt1*	B22	PAD1[1]	C22	PAD1[3]
A23	PAD1[0]	B23	PAD1[6]	C23	PAD1[7]
A24	PAD1[5]	B24	PAD1[4]	C24	GND
A25	CBE1*[0]	B25	GND	C25	PAD1[9]
A26	GND	B26	GND	C26	PAD1[11]

Table 272: Pinout Table (Continued)

Ball#	Signal Name	Ball#	Signal Name	Ball#	Signal Name
D01–D26		***E01–E04, E23–E26***		***H01–H04, H23–H26***	
D01	DH[20]	E01	DH[16]	H01	DH[6]
D02	DH[22]	E02	DH[19]	H02	DH[8]
D03	DH[21]	E03	DH[17]	H03	DH[5]
D04	GND	E04	DH[18]	H04	GND
D05	DH[31]	E23	PAD1[14]	H23	Frame1*/Req64*
D06	VDD	E24	PAD1[8]	H24	Trdy1*
D07	A/DL[2]	E25	Par1/Par64	H25	PAD1[17]
D08	A/DL[6]	E26	CBE1*[1]	H26	PAD1[18]
D09	GND	***F01–F04, F23–F26***		***J01–J04, J23–J26***	
D10	A/DL[14]	F01	DH[12]	J01	DH[3]
D11	VDD	F02	DH[15]	J02	DH[4]
D12	A/DL[18]	F03	DH[13]	J03	DH[1]
D13	A/DL[27]	F04	VDD	J04	DH[7]
D14	GND	F23	VDD	J23	GND
D15	A/DL[31]	F24	PAD1[12]	J24	PAD1[19]
D16	VDD	F25	DevSel1*/Ack64*	J25	PAD1[22]
D17	DP[7]	F26	Perr1*	J26	PAD1[16]
D18	TSIZ[1]	***G01–G04, G23–G26***		***K01–K04, K23–K26***	
D19	GND	G01	DH[10]	K01	DH[0]
D20	AACK*	G02	DH[11]	K02	DH[2]
D21	VDD	G03	DH[9]	K03	JTDO
D22	PAD1[2]	G04	DH[14]	K04	JTDI
D23	GND	G23	Stop1*	K23	PAD1[21]
D24	PAD1[10]	G24	Serr1*	K24	PAD1[23]
D25	PAD1[13]	G25	CBE1*[2]	K25	IdSel1
D26	PAD1[15]	G26	Irdy1*	K26	PAD1[20]

Table 272: Pinout Table (Continued)

Ball#	Signal Name	Ball#	Signal Name	Ball#	Signal Name
L01–L04, L11–L16, L23–L26		***M23–M26***		***P11–P16, P23–P26***	
L01	JTCLK	M23	CBE1*[3]	P11	GND
L02	JTMS	M24	PAD1[26]	P12	GND
L03	AD[61]	M25	PAD1[30]	P13	GND
L04	VDD	M26	PAD1[24]	P14	GND
L11	GND	***N01–N04, N11–N16, N23–N26***		P15	GND
L12	GND	N01	AD[58]	P16	GND
L13	GND	N02	AD[60]	P23	GND
L14	GND	N03	AD[54]	P24	PAD1[29]
L15	GND	N04	GND	P25	PAD0[30]
L16	GND	N11	GND	P26	PClk0
L23	VDD	N12	GND	***R01–R04, R11–R16, R23–R26***	
L24	VREF1	N13	GND	R01	AD[51]
L25	PAD1[25]	N14	GND	R02	AD[53]
L26	PAD1[27]	N15	GND	R03	AD[49]
M01–M04, M11–M16		N16	GND	R04	AD[47]
M01	AD[62]	N23	PAD1[28]	R11	GND
M02	AD[63]	N24	PAD1[31]	R12	GND
M03	AD[57]	N25	Rst*	R13	GND
M04	AD[59]	N26	PClk1	R14	GND
M11	GND	***P01–P04***		R15	GND
M12	GND	P01	TClk	R16	GND
M13	GND	P02	AD[56]	R23	PAD0[29]
M14	GND	P03	AD[52]	R24	PAD0[28]
M15	GND	P04	AD[55]	R25	PAD0[25]
M16	GND			R26	PAD0[24]

Table 272: Pinout Table (Continued)

Ball#	Signal Name	Ball#	Signal Name	Ball#	Signal Name
T01–T04, T11–T16, T23–T26		***V01–V04, V23–V26***		***AA01–AA04, AA23–AA26***	
T01	AD[48]	V01	AD[40]/BootCS*	AA01	ADP[6]
T02	AD[50]	V02	AD[42]	AA02	AD[32]/ DMAAck*[0]
T03	AD[45]	V03	AD[37]/CS*[1]	AA03	ADP[2]/EOT[2]
T04	VDD	V04	GND	AA04	VDD
T11	GND	V23	PAD0[18]	AA23	VDD
T12	GND	V24	PAD0[20]	AA24	DevSel0*
T13	GND	V25	PAD0[23]	AA25	Lock0*
T14	GND	V26	PAD0[22]	AA26	Trdy0*
T15	GND	***W01–W04, W23–W26***		***AB01–AB04, AB23–AB26***	
T16	GND	W01	AD[38]/CS*[2]	AB01	ADP[3]/EOT[3]
T23	VDD	W02	AD[39]/CS*[3]	AB02	ADP[4]/BankSel[1]
T24	PAD0[31]	W03	AD[33]/ DMAAck*[1]	AB03	AD[30]
T25	CBE0*[3]	W04	AD[35]/ DMAAck*[3]	AB04	ADP[0]/EOT[0]
T26	PAD0[26]	W23	GND	AB23	Par0
U01–U04, U23–U26		W24	PAD0[16]	AB24	Serr0*
U01	AD[44]	W25	PAD0[19]	AB25	CBE0*[1]
U02	AD[46]	W26	PAD0[17]	AB26	Perr0*
U03	AD[41]/DevRW*	***Y01–Y04, Y23–Y26***		***AC01–AC08***	
U04	AD[43]	Y01	AD[34]/ DMAAck*[2]	AC01	AD[31]
U23	IdSel0	Y02	AD[36]/CS*[0]	AC02	ADP[1]/EOT[1]
U24	PAD0[27]	Y03	ADP[5]/DAdr[11]	AC03	AD[28]
U25	PAD0[21]	Y04	ADP[7]	AC04	GND
U26	VREF0	Y23	Stop0*	AC05	AD[19]
		Y24	CBE0*[2]	AC06	VDD
		Y25	Irdy0*	AC07	AD[15]
		Y26	Frame0*	AC08	GND

Table 272: Pinout Table (Continued)

Ball#	Signal Name	Ball#	Signal Name	Ball#	Signal Name
AC09–AC26		***AD11–AD26***		***AE13–AE26***	
AC09	AD[8]	AD11	SDQM*[3]	AE13	SDQM*[2]
AC10	AD[0]	AD12	SCAS*	AE14	SCS*[0]
AC11	VDD	AD13	SCS*[3]	AE15	SRAS*
AC12	SDQM*[1]	AD14	BypsOE*/MGNT*	AE16	DMAReq*[3]
AC13	GND	AD15	ALE	AE17	DMAReq*[1]
AC14	SCS*[1]	AD16	DMAReq*[2]	AE18	DAdr[9]/Wr*[6]
AC15	Ready*	AD17	DAdr[8]/Wr*[5]	AE19	DAdr[6]/Wr*[3]
AC16	VDD	AD18	DAdr[4]/Wr*[1]	AE20	DAdr[3]/Wr*[0]
AC17	DAdr[10]/Wr*[7]	AD19	DAdr[0]/BAdr[0]	AE21	Int0*
AC18	GND	AD20	PAD0[3]	AE22	PAD0[2]
AC19	DAdr[2]/BAdr[2]	AD21	PAD0[0]	AE23	PAD0[7]
AC20	Req0*	AD22	PAD0[4]	AE24	CBE0*[0]
AC21	VDD	AD23	PAD0[11]	AE25	GND
AC22	PAD0[6]	AD24	GND	AE26	PAD0[9]
AC23	GND	AD25	PAD0[8]	***AF01–AF14***	
AC24	PAD0[13]	AD26	PAD0[15]	AF01	GND
AC25	PAD0[14]	***AE01–AE12***		AF02	AD[26]
AC26	PAD0[12]	AE01	GND	AF03	AD[24]
AD01–AD10		AE02	GND	AF04	AD[21]
AD01	AD[27]	AE03	AD[25]	AF05	AD[17]
AD02	AD[29]	AE04	AD[23]	AF06	AD[13]
AD03	GND	AE05	AD[20]	AF07	AD[11]
AD04	AD[22]	AE06	AD[16]	AF08	AD[7]
AD05	AD[18]	AE07	AD[12]	AF09	AD[4]
AD06	AD[14]	AE08	AD[9]	AF10	AD[1]
AD07	AD[10]	AE09	AD[5]	AF11	SDQM*[6]
AD08	AD[6]	AE10	AD[3]	AF12	SDQM*[4]
AD09	AD[2]	AE11	DWr*	AF13	SDQM*[0]
AD10	SDQM*[7]	AE12	SDQM*[5]	AF14	SCS*[2]

Table 272: Pinout Table (Continued)

Ball#	Signal Name	Ball#	Signal Name	Ball#	Signal Name
AF15–AF18		***AF19–AF22***		***AF23–AF26***	
AF15	DMAReq*[0]/ MREQ*	AF19	DAdr[5]/Wr*[2]	AF23	PAD0[5]
AF16	CSTiming*	AF20	DAdr[1]/BAdr[1]	AF24	PAD0[10]
AF17	BankSel[0]	AF21	Gnt0*	AF25	GND
AF18	DAdr[7]/Wr*[4]	AF22	PAD0[1]	AF26	GND

21. DC CHARACTERISTICS

Table 273: Absolute Maximum Ratings

Symbol	Parameter	Min.	Max.	Unit
V_{CC}	Supply Voltage	-0.3	4.0	V
V_i	Input Voltage	-0.3	5.5	V
V_o	Output Voltage	-0.3	5.5	V
I_o	Output Current		24	mA
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
E_{SD}			2000	V

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 274: Recommended Operating Conditions

Symbol	Parameter	Min.	Typ.	Max.	Unit
V_{CC}	Supply Voltage	3.15	3.3	3.45	V
V_i	Input Voltage	0		5.5	V
V_o	Output Voltage	0		Vdd	V
T_c	Operating Case Temperature	0		90	C

Table 275: PIN Capacitance

Symbol	Parameter	Min.	Typ.	Max.	Unit
C_{in}	Input Capacitance		7.2		pF
C_{out}	Output Capacitance		7.2		pF

21.1 DC Electrical Characteristics Over Operating Range

Table 276: DC Electrical Characteristics Over Operating Range

Symbol	Parameter	Test Condition	Min.	Max.	Unit	Loading
V_{ih}	Input HIGH level	Guaranteed Logic HIGH level	2.0	5.5	V	
V_{il}	Input LOW level	Guaranteed Logic LOW level	-0.5	0.8	V	
V_{oh}	Output HIGH Voltage: DWr*, SDQM[7:0], SCAS*, SCS[3:0]*, SRAS*, BankSel[0], DAdr[10:3]/Wr[7:0]*, DAdr[2:0]/ BAdr[2:0], MGNT*/BypsOE*, DMAReq[3:1]*, DMAReq[0]*/ MREQ*	IoH = 16 mA	2.4		V	50 pF
V_{oh}	Output HIGH Voltage: ALE, ADP[7:6], ADP[5]/DAdr[11], ADP[4]/BankSel[1], ADP[3:0]/ EOT[3:0]*	IoH = 12 mA	2.4		V	50 pF
V_{oh}	Output HIGH Voltage: AD[63:42], AD[41]/DevRW*, AD[40]/BootCS*, AD[39:36]/ CS[3:0]*, AD[35:32]/ DMAAck[3:0]*, AD[31:0], A/DL[0-31],DH[0-31], DP[0-7],AACK*,TA*, BG*, DBG*, Interrupt* L2BG*,L2DBG*,QsAEn, QsAEn2	IoH = 8 mA	2.4		V	50 pF
V_{oh}	Output HIGH Voltage: CSTiming*	IoH = 8 mA	2.4		V	30 pF
V_{ol}	Output LOW Voltage: DWr*, SDQM[7:0], SCAS*, SCS[3:0]*, SRAS*, BankSel[0], DAdr[10:3]/Wr[7:0]*, DAdr[2:0]/ BAdr[2:0], MGNT*/BypsOE*, DMAReq[3:1]*, DMAReq[0]*/ MREQ*	IoL = 16 mA		0.4	V	50 pF
V_{ol}	Output LOW Voltage: ScDOE*, ALE, ADP[7:6], ADP[5]/ DAdr[11], ADP[4]/BankSel[1], ADP[3:0]/EOT[3:0]*	IoL = 12 mA		0.4	V	50 pF

Table 276: DC Electrical Characteristics Over Operating Range (Continued)

Symbol	Parameter	Test Condition	Min.	Max.	Unit	Loading
V_{ol}	Output LOW Voltage: AD[63:42], AD[41]/DevRW*, AD[40]/BootCS*, AD[39:36]/ CS[3:0]*, AD[35:32]/ DMAAck[3:0]*, AD[31:0], A/DL[0-31],DH[0-31], DP[0-7],AACK*,TA*, BG*, DBG*, Interrupt* L2BG*,L2DBG*,QsAEn, QsAEn2	IoL = 8 mA		0.4	V	50 pF
V_{ol}	Output LOW Voltage: CSTiming*	IoL = 8 mA		0.4	V	30 pF
I_{ih}	Input HIGH Current			+-1	uA	
I_{il}	Input LOW Current			+-1	uA	
I_{ozh}	High Impedance Output Current			+-1	uA	
I_{ozl}	High Impedance Output Current			+-1	uA	
Vh	Input Hysteresis		TBD	TBD	mV	
I_{cc}	Operating Current	Add = 3.45V f = 75MHz		470	mA	
	PAD0[31:0], PAD1[31:0], Frame0*, Frame1*/Req64*, Irdy0*, Irdy1*, Trdy0*, Trdy1*, DevSel0*, DevSel1*/Ack64*, Stop0*, Stop1*, Perr0*, Perr1*, Par0, Par1/Par64, CBE0[3:0]*, CBE1[3:0]*, Gnt0*, Gnt1*, Req0*, Req1*, IdSel0, IdSel1, Lock0*, Serr0*, Serr1*, Int0*	See PCI Specification Rev. 2.1				

21.2 Thermal Data

Table 277 shows the package thermal data for the GT-64130.

Galileo Technology recommends the use of heatsink for most systems, especially those with little or no airflow. Using a heatsink with the commercial grade device is especially important.

Use an adequate airflow, layout, and other means to meet the recommed operating conditions listed in Table 277.

Table 277: BGA Thermal Data

Airflow	Definition	Value		
		0 m/s	1 m/s	2 m/s
θ_{ja}	Thermal resistance: junction to ambient.	22.9 c/w	21.2 c/w	19.2 c/w
θ_{jc}	Thermal resistance: junction to case (not air-flow dependent)	0.5 c/w		
θ_{ca}	Thermal resistance: case to ambient)	22.4 c/w	20.7 c/w	18.7 c/w

22. AC TIMING

Table 278: AC Timing Measurement Formulas

Commercial	Industrial
TCase= 90^{o}C; VCC= +3.3V, +/- 5%	TAmbient= -40-85^{o}C; VCC= +3.3V, +/- 5%

Table 279: Commercial AC Timing

All Delays, Setup, and Hold times are referred to TClk RISING edge, unless stated otherwise.

		66 Mhz		75 Mhz			
Signals	Description	Min.	Max.	Min.	Max.	Unit	Loading
Clk							
TClk	Pulse Width High					ns	
TClk	Pulse Width Low					ns	
TClk	Clock Period	15		13.3		ns	
TClk	Rise Time					V/ns	
TClk	Fall Time					V/ns	
Rst*	Active	10		10		TClk	
CPU Interface							
TS*, ARTRY*, L2HIT*, A/DL[0-31], DH[0-31], TT[1], TT[3], TBST*, TSIZ[0-2], L2BR*	Setup	4		3		ns	
TS*, ARTRY*, L2HIT*, A/DL[0-31], DH[0-31], TT[1], TT[3], TBST*, TSIZ[0-2], L2BR*	Hold	1		1		ns	
A/DL[0-31],DH[0-31], DP[0-7], AACK*, TA*, QsAEn, QsAEn2, BG*, DBG*, L2BG*, L2DBG*, Interrupt*	Output Delay	2	8	2	7	ns	50pF

Table 279: Commercial AC Timing (Continued)

All Delays, Setup, and Hold times are referred to TClk RISING edge, unless stated otherwise.

Signals	Description	66 Mhz		75 Mhz		Unit	Loading
		Min.	Max.	Min.	Max.		
PCI Interface							
Pclk0, Pclk1	Clock Period	15		15		ns	
All Inputs, 64 bit	Setup	9		7		ns	
All Inputs, 32 bit	Setup	7		5			
Req*, 64 and 32 bit	Output Delay	2	9	2	8	ns	
All Inputs, 64 and 32 bit	Hold	0		0		ns	
All Outputs except Req*, 64 and 32 bit	Output Delay	2	8	2	7	ns	
Memory Interface							
AD[63:0], SCS[3:0], SDQM[7:0]	Output Delay	2	9	2	8	ns	50pF
AD[63:0]	Setup	4		3		ns	
AD[63:0]	Hold	1		1		ns	
DAdr[11:0]	Setup (Input Reset Programming)	5		5		ns	
DAdr[11:0]	Hold (Input Reset Programming)	1		1		ns	
DAdr[11:0]	Output Delay	2	7	2	6.5	ns	50pF
DAdr[2:0]	Output Delay (Device Burst)	2	7	2	6.5	ns	50pF
DAdr[10:3]	Output Delay from TClk Falling (Device Write)	2	7	2	6.5	ns	50pF
ADP[7:0]	Output Delay (ECC)	2	9	2	8	ns	50pF
ADP[3:0]/ EOT[3:0]*	Setup (EOT)	7		5		ns	
ADP[3:0]/ EOT[3:0]*	Hold (EOT)	1		1		ns	

Table 279: Commercial AC Timing (Continued)

All Delays, Setup, and Hold times are referred to TClk RISING edge, unless stated otherwise.

Signals	Description	66 Mhz		75 Mhz		Unit	Loading
		Min.	Max.	Min.	Max.		
Memory Interface (Continued)							
ADP[1]	Output Delay (ALE)	2	8	2	7	ns	30pF
ADP[1]	Output Delay, TClk Falling (ALE)	2	8	2	7	ns	30pF
ADP[2]	Output Delay (BefOE)	2	8	2	7	ns	30pF
ADP[3]	Output Delay (DWr*)	2	9	2	8	ns	50pF
ADP[5:4]	Output Delay (64Mbit SDRAM Addr.)	2	7	2	6.5	ns	50pF
ADP[7:6]	Output Delay (SRAS* & SCAS*)	2	7	2	6.5	ns	50pF
Ready*	Setup	7		5		ns	
Ready*	Hold	1		1		ns	
MGnt*/ByPsOE*	Output Delay (UMA Grant)	2	9	2	8	ns	30pF
MGnt*/ByPsOE*	Output Delay (Bypass OE)	2	11	2	10	ns	30pF
SRAS*	Output Delay	2	7	2	6.5	ns	50pF
SCAS*	Output Delay	2	7	2	6.5	ns	50pF
ALE	Output Delay (ALE)	2	8	2	7	ns	30pF
ALE	Output Delay, TClk FALLING (ALE)	2	8	2	7	ns	30pF
DWr*	Output Delay	2	7	2	6.5	ns	50pF
CSTiming*	Output Delay	2	8	2	7	ns	30pF
DMAReq[0]*/ MREQ*/SRAS*	Setup (DMAReq[0]*/ MREQ*)	7		5		ns	

Table 279: Commercial AC Timing (Continued)

All Delays, Setup, and Hold times are referred to TClk RISING edge, unless stated otherwise.

Signals	Description	66 Mhz		75 Mhz		Unit	Loading
		Min.	Max.	Min.	Max.		
Memory Interface (Continued)							
DMAReq[0]*/ MREQ*/SRAS*	Hold (DMAReq[0]*/ MREQ*)	1		1		ns	
DMAReq[0]*/ MREQ*/SRAS*	Output Delay (SRAS*)	2	9	2	8	ns	50pF
DMAReq[1]*/Bank-Sel[1]*	Setup (DMAReq[1]*)	7		5		ns	
DMAReq[1]*/Bank-Sel[1]*	Hold (DMAReq[1]*)	1		1		ns	
DMAReq[1]*/Bank-Sel[1]*	Output Delay (BankSel[1]*)	2	7	2	6.5	ns	50pF
DMAReq[2]*/ DAdr[11]	Setup (DMAReq[[2]*)	7		5		ns	
DMAReq[2]*/ DAdr[11]	Hold (DMAReq[2]*)	1		1		ns	
DMAReq[2]*/ DAdr[11]	Output Delay (DAdr[11])	2	7	2	6.5	ns	50pF
DMAReq[3]*/ SCAS*/EOT[0]*/ TREQ*	Setup (DMAReq[3]*/ EOT[0]*)	7		5		ns	
DMAReq[3]*/ SCAS*/EOT[0]*/ TREQ*	Hold (DMAReq[3]*/ EOT[0]*)	1		1		ns	
DMAReq[3]*/ SCAS*/EOT[0]*/ TREQ*	Output Delay (SCAS*/TREQ*)	2	9	2	8	ns	50pF

Table 280: Industrial AC Timing

Signals	Description	66 Mhz		Unit	Loading
		Min.	Max.		
Clk					
TClk	Pulse Width High			ns	
TClk	Pulse Width Low			ns	
TClk	Clock Period	15		ns	
TClk	Rise Time			V/ns	
TClk	Fall Time			V/ns	
Rst*	Active	10		TClk	
CPU Interface					
TS*, ARTRY*, L2HIT*, A/DL[0-31], DH[0-31], TT[1], TT[3], TBST*, TSIZ[0-2], L2BR*	Setup	4		ns	
TS*, ARTRY*, L2HIT*, A/DL[0-31], DH[0-31], TT[1], TT[3], TBST*, TSIZ[0-2], L2BR*	Hold	1.2		ns	
A/DL[0-31],DH[0-31], DP[0-7], AACK*, TA*, QsAEn, QsAEn2, BG*, DBG*, L2BG*, L2DBG*, Interrupt*	Output Delay	1.5	8	ns	50pF
PCI Interface					
Pclk0, Pclk1	Clock Period	15		ns	
All Inputs, 64 bit	Setup	9		ns	
All Inputs, 32 bit	Setup	7			
Req*, 64 and 32 bit	Output Delay	1.5	9	ns	
All Inputs, 64 and 32 bit	Hold	0.2		ns	
All Outputs except Req*, 64 and 32 bit	Output Delay	2	8	ns	
Memory Interface					
AD[63:0], SCS[3:0], SDQM[7:0]	Output Delay	1.5	9	ns	50pF
AD[63:0]	Setup	4		ns	
AD[63:0]	Hold	1.2		ns	
DAdr[11:0]	Setup (Input Reset Programming)	5		ns	

Table 280: Industrial AC Timing

Signals	Description	66 Mhz		Unit	Loading
		Min.	Max.		
Memory Interface (Continued)					
DAdr[11:0]	Hold (Input Reset Programming)	1.2		ns	
DAdr[11:0]	Output Delay	1.5	7	ns	50pF
DAdr[2:0]	Output Delay (Device Burst)	1.5	7	ns	50pF
DAdr[10:3]	Output Delay from TClk Falling (Device Write)	1.5	7	ns	50pF
ADP[7:0]	Output Delay (ECC)	1.5	9	ns	50pF
ADP[3:0]/EOT[3:0]*	Setup (EOT)	7		ns	
ADP[3:0]/EOT[3:0]*	Hold (EOT)	1.2		ns	
ADP[1]	Output Delay (ALE)	1.5	8	ns	30pF
ADP[1]	Output Delay, TClk Falling (ALE)	1.5	8	ns	30pF
ADP[2]	Output Delay (BefOE)	1.5	8	ns	30pF
ADP[3]	Output Delay (DWr*)	1.5	9	ns	50pF
ADP[5:4]	Output Delay (64Mbit SDRAM Addr.)	1.5	7	ns	50pF
ADP[7:6]	Output Delay (SRAS* & SCAS*)	1.5	7	ns	50pF
Ready*	Setup	7		ns	
Ready*	Hold	1.2		ns	
MGnt*/ByPsOE*	Output Delay (UMA Grant)	1.5	9	ns	30pF
MGnt*/ByPsOE*	Output Delay (Bypass OE)	1.5	11	ns	30pF
SRAS*	Output Delay	1.5	7	ns	50pF
SCAS*	Output Delay	1.5	7	ns	50pF
ALE	Output Delay (ALE)	1.5	8	ns	30pF

Table 280: Industrial AC Timing

Signals	Description	66 Mhz		Unit	Loading
		Min.	Max.		
Memory Interface (Continued)					
ALE	Output Delay, TClk FALLING (ALE)	1.5	8	ns	30pF
DWr*	Output Delay	1.5	7	ns	50pF
CSTiming*	Output Delay	1.5	8	ns	30pF
DMAReq[0]*/MREQ*/SRAS*	Setup (DMAReq[0]*/ MREQ*)	7		ns	
DMAReq[0]*/MREQ*/SRAS*	Hold (DMAReq[0]*/ MREQ*)	1.2		ns	
DMAReq[0]*/MREQ*/SRAS*	Output Delay (SRAS*)	1.5	9	ns	50pF
DMAReq[1]*/BankSel[1]*	Setup (DMAReq[1]*)	7		ns	
DMAReq[1]*/BankSel[1]*	Hold (DMAReq[1]*)	1.2		ns	
DMAReq[1]*/BankSel[1]*	Output Delay (BankSel[1]*)	1.5	7	ns	50pF
DMAReq[2]*/DAdr[11]	Setup (DMAReq[[2]*)	7		ns	
DMAReq[2]*/DAdr[11]	Hold (DMAReq[2]*)	1.2		ns	
DMAReq[2]*/DAdr[11]	Output Delay (DAdr[11])	1.5	7	ns	50pF
DMAReq[3]*/SCAS*/EOT[0]*/TREQ*	Setup (DMAReq[3]*/ EOT[0]*)	7		ns	
DMAReq[3]*/SCAS*/EOT[0]*/TREQ*	Hold (DMAReq[3]*/ EOT[0]*)	1.2		ns	
DMAReq[3]*/SCAS*/EOT[0]*/TREQ*	Output Delay (SCAS*/TREQ*)	1.5	9	ns	50pF

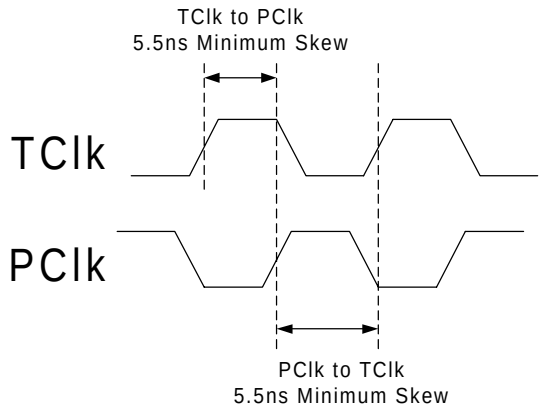
22.1 TClk/PClk Restrictions

TClk cycle must be smaller than PClk cycle by at least 1ns (T **tclk** < T**pclk** + 1ns). This restriction applies to all sync modes.

There is one exception to this restriction. TClk and PClk can run at the same frequency if the following conditions are met:

- The two clocks are synchronized (derived from the same clock source).
- If running at sync mode 1, a minimum skew of 5.5ns must be observed between rising edge of TClk and PClk, as shown in Figure 38. Galileo Technology recommends using an inverted TClk as PClk in order to guarantee this skew.
- If running at sync mode 2 or 3, a maximum skew of 2ns between rising edge of TClk and PClk must be met.

Figure 38: TClk = PClk Skew Requirement



In addition to the above restriction, there are few sync modes specific restrictions, summarized in Table 281.

Table 281: TClk/PClk Restrictions

Sync Mode	PClk Frequency Range	Restrictions
0,4	from DC up to TClk	T**pclk** > T**tclk** + 1.5ns
1	from TClk/2 up to TClk	T**pclk** < 2T **tclk** -1ns, unless running with **synchronized** Tpclk = 2Ttclk and minimum skew of 5.5ns between PClk rise and TClk rise is guaranteed. For example, if T **tclk** = 15ns (66MHz), T**pclk** should be smaller than 29ns (unless running with synchronized clocks).
2,3	from TClk/2 up to TClk	**TClk and PClk are synchronized**, and a maximum skew of 2ns between PClk rise and TClk rise is guaranteed.

Table 281: TClk/PClk Restrictions

Sync Mode	PClk Frequency Range	Restrictions
5	from TClk/3 up to TClk/2	T**pclk** < 3T **tclk** - 1ns, unless running with **synchronized** Tpclk = 3T tclk and minimum skew of 5.5ns between PClk rise and TClk rise is guaranteed. For example, if T **tclk** = 13.3ns (75MHz), T**pclk** should be smaller than 39ns (unless running with synchronized clocks).
6,7	from TClk/3 up to TClk/2	**TClk and PClk are synchronized** , and a maximum skew of 2ns between PClk rise and TClk rise is guaranteed.

The sync mode can be programed by the CPU, the PCI or during autoload. For sync mode information, see Section 19.15 “PCI Internal” on page 200.

22.2 Additional Delay Due to Capactive Loading

Some applications may require additional capacitive loading on different output pins of the GT-64130. For example, when using multiple GT-64130s connected to the same A/DL bus, the 50pF load specification may be exceeded.

This additional loading affects the output delays of the signals, depending on the drive strength of the output driver. The follow section describes how to calculate the affects of additional loading on the output drivers.

22.2.1 Calculating the Maximum Delay Due to Loading

The basic equation for calculating the maximum delay is:

Tmax = [Atypa + (Btyp * Cr)] * 1.6

where

Tmax is the maximum delay in nanoseconds.
Atypa must be calculated by the designer as shown in Section 22.2.1.1.
Btyp is a parameter according to the specific output buffer from Table 282.
Cr is the capacitance required.

22.2.1.1 Calculating Atypa

To calculate Atypa use the given values in the AC Timing Parameters table. Start with the equation:

Tspec = [Atypa + (Btyp * Cds)] * 1.6

and solving for Atypa,

Atypa = (Tspec/1.6) - (Btyp * Cds)

where

Tspec is the maximum delay parameter from the AC Timing Parameters from Table 279.
Btyp is a parameter according to the specific output buffer from Table 282.

C is the capacitance parameter from the AC Timing Parameters Table 279.

NOTE: 1.6 is the worst case derating factor.

22.2.2 Calculating the Minimum Delay Due to Loading

The basic equation for calculating the maximum delay is:

Tmin = [Atypb + (Btyp * Cr)] * 0.7

where

Tmin is the maximum delay in nanoseconds.
Atypb must be calculated by the designer as shown in Section 22.2.2.1
Btyp is a parameter according to the specific output buffer from Table 282.
Cr is the capacitance required.

22.2.2.1 Calculating Atypb

Atypa can be calculated by using the given values in the AC Timing Parameters table. We start with the equation:

Tspec = [Atypb + (Btyp * Cds)] * 0.7

and solving for Atypb,

Atypb = (Tspec/0.7) - (Btyp * Cds)

where

Tspec is the maximum delay parameter from the AC Timing Parameters Table 279.
Btyp is a parameter according to the specific output buffer from Table 282.
Cds is the capacitance parameter from the AC Timing Parameters Table 279.

NOTE: 0.7 is the worst case derating factor.

22.2.3 Btyp Values

Table 282 lists the Btyp values for the different output buffers of the GT-64130. See Section 21.1 “DC Electrical Characteristics Over Operating Range” on page 247 for the corresponding pin and output driver.

Table 282: Btyp Values

Output Driver	Low to High Btyp Value	High to Low Btype Value
4mA	0.06	0.077
8mA	0.031	0.039
12mA	0.021	0.028
16mA	0.018	0.022
All PCI Outputs	0.015	0.018

22.2.4 Example Calculation of Tmax

The following is an example of how to calculate the maximum delay on the AD[0] line for a 75pF load.

From the AC Timing Parameters Table, for a 50pF load, the maximum output delay on the AD[0] line is specified as 8ns. Looking at Table 282, Btyp for 8mA drivers is = 0.031 (Low to High transition).

Substituting these values into

Atypa = (Tspec/1.6) - (Btyp * Cds)

gives Atypa = 3.45.

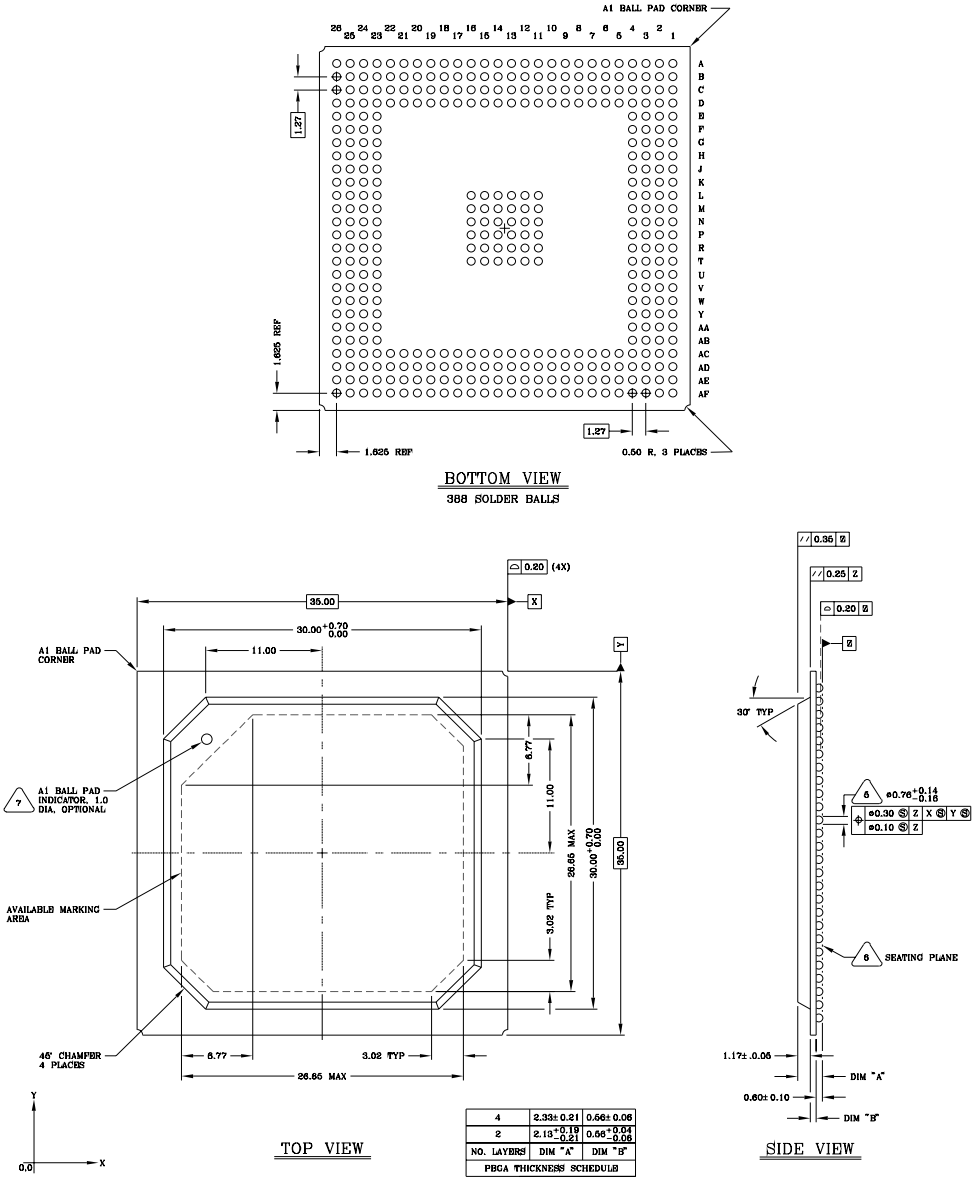
Substituting Atypa of 3.45, Btyp of 0.031 and Cr of 75pF in

Tmax = [Atypa + (Btyp * Cr)] * 1.6

gives Tmax = 9.24. This means that the maximum output delay of AD[0] with a 75pF load is 9.24ns.

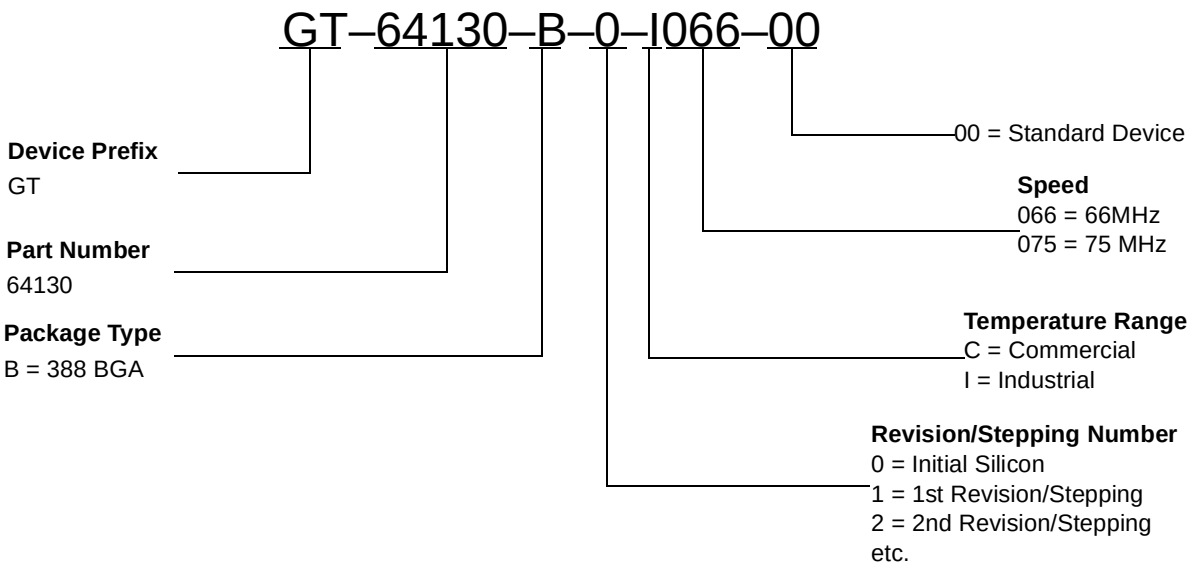
23. 388 PBGA PACKAGE MECHANICAL INFORMATION

Figure 39: Package Information



24. GT–64130 PART NUMBERING

Figure 40: Sample Part Number



24.1 Standard Part Number

The standard part number for the GT–64130 is the GT–64130–B–X.

Without the –XYYY–ZZ suffix, this part number indicates that is the commercial temperature grade, 75MHz version. In other words, the GT–64130–B–X is the same as the GT–64130–B–X–C075–00, although it will not be marked as such.

24.2 Valid Part Numbers

The following part numbers are the only valid part numbers that can be used when ordering the GT–64130:

- GT–64130–B–X: Commercial temperature, 75MHz
- GT–64130–B–X–C066–00: Commercial temperature, 66MHz
- GT–64130–B–X–I066–00: Industrial temperature, 66MHz

25. REVISION HISTORY

Table 283: Document History

Document Type	Rev. Number	Date	Comments
Product Preview	0.1	07/02/1998	Preliminary Version
Product Preview	0.2	11.02/1998	Preliminary Version
Product Preview	0.4	12/05/1998	Add TClk/PClk ratio restriction (Section 2.1 “Pin Assignment Table” on page 15, Section 6.8 “PCI Bus/Device Bus/CPU Clock Synchronization” on page 108, Section 6.13.3, Section 22.1 “TClk/PClk Restrictions” on page 257.) Added PClk1 information, Corrected MPC860 Pin List Table (Section 2.1.) CS[2:0] High Decodes up to 0x1EF1.FFFF by default, not 0x1DFF.FFFF (Table 3.) Correct SDRAM address decode tables (Section 5.1.4.) Correct flow through bit description (Section 5.4.0.2.) Correct SDRAM duplicate signals table (Section 5.10.) 12. Update memory interface restrictions (Section 5.11.) Clarify MaxBurst programing (Section 6.3.2). Correct SDRAM connection app. note (Section 12.1) Correct DMA to PCI override description (Section 8.2.18 and Section 19.12.) Correct Device Parameters registers initial values (Section 19.10.) PCI Sync Mode Information (Section 19.15.) Correct CPU to PCI override bits (Section 19.3) Remove Internal space endianess (Section 19.3 and Section 11.)
Datasheet	1.0	APR 14, 1999	EOT bits are no longer written as being pulled low. Supports the PowerPC MPC604e CPU. Added Rst* and TClk to Figure 1: “Pin Information” on page 14.

Table 283: Document History

Document Type	Rev. Number	Date	Comments
Datasheet	1.0	APR 14, 1999	Added the following note in Section 3.1.1 “CPU Side Decoding Process” on page 30 and Section 19. “Register Tables” on page 160: **NOTE:** When referring to an address in this section, the high number is the most significant bit and the lowest number is the least significant bit. For example, bit [31:0] means that the most significant bit is 31 and the least significant bit is 0. This is opposite of the PowerPC convention in which the lowest bit number is the most significant bit and the highest bit number is the least significant bit. New piplining information in Section 4.2 “PowerPC Address/Data Buses Multiplex” on page 43. Removed Section 4.16 Fast Mode Support. Not supported by the GT-64130. Two new memory controller restrictions in Section 5.10 “Programming the ADP lines for other Functions” on page 91. Added Section 13. “JTAG Application Notes” on page 145 to include information about disabling JTAG. Added Section 6.3.5 “PCI Target Byte Swapping” on page 102 Changed Table 176 on page 202 from RAS to SCS. This change also applies to other instances when RAS was used. The text now refers to the SCS bit instead of RAS. In section Section 6.6.3 “Expansion ROM Functionality” on page 107, changed SCS[3]/BootCS* to CS[3]/BootCS*. Revised table in Section 2.2 “603e/860 Pins Multiplex Table” on page 25. Revised waveform in Figure 24: “Waveform Showing Device Write Parameters” on page 84. For pin Frame1*/Req64*,DAdr[2], changed function for setting 10 to reserved, see Table 51 on page 139. Added note in Section 14.1 “Endian Background” on page 146 about endian definition document on the Galileo Technology Website at http://www.GalileoT.com/library/syslib.htm.

Table 283: Document History

Document Type	Rev. Number	Date	Comments
Datasheet (Continued)	1.0	APR 14, 1999	Added information about configuring the GT-64130 to work in an MPC860 configuration, see Section 17. “Using the GT-64130 in MPC860 Configuration” on page 156. New θ_{ja} and θ_{jc} thermal data figures in Table 277 on page 249.Bit 9 in Table 87 on page 174 is now reserved. New AC timing specifications for commercial and industrial grade parts in Section 22. “AC Timing” on page 250. Revised TClk/PClk restrictions specific to certain sync modes in Table 281 on page 257.
Datasheet Revision	1.1	DEC 15, 1999	Added notes to the pin descriptions for TA* and AACK* in Table 1 “Pin Assignments” on page 15 explaining that these pins MUST be pulled-up through resistors to VCC. **NOTE:** Galileo Technology recommends using 4.7KOhm resistors. In Table 1 “Pin Assignments” on page 15, revised information for PClk0 and PClk1. The PClk cycle must be higher than TClk cycle by at least 1ns. In Table 2 “603e/860 Pins Multiplex” on page 25, the 860 setting for TSIZ[0] must be pulled down. Added new disabling device decoder information to Section 3.3 “Disabling the Device Decoders” on page 34. Added information about PowerQUICC II (MPC8260) CPU support in Section 4.18 “Power-QUICC II Support” on page 60. Changed DMAReq[3]* to DMAReq[0] in Section 4.16 “32-bit Bus Support” on page 55. Revised footnote on page 67. New aggressive pre-fetch restriction added to Section 5.11 “Memory Controller Restrictions” on page 92. This new restriction is also added to Section 6.3.3 “PCI Target Read Prefetching” on page 101. Added two new DMA restrictions to Section 8.9 “DMA Restrictions” on page 134. Added note to Section 8.1.4 “Pointer to the Next Record Register” on page 122 stating that the next record pointer must be 16 bytes aligned. This means bits [3:0] must be set to 0. This note is also added to Table 155 to Table 158 on page 191.

Table 283: Document History

Document Type	Rev. Number	Date	Comments
Datasheet Revision (continued)	1.1	DEC 15, 1999	Added detail in Section 9. “Timer/Counters” on page 136 that The timer/counter’s count frequency is equal to the TCLk. New note in Section 11. “Reset Configuration” on page 139 stating that Rst* must be de-asserted for at least 10 PClk cycles before any CPU transactions are generated. An addition to Table 51 on page 140. Pin DAdr[1] is reserved and must be pulled low. In other words, DADr[1] must be pulled through a 4.7KOhm resistor to GND on reset. Added Table 65 “PCI_0 as 32-bit PCI Only” on page 151 about connecting the GT-64130 to a single 32-bit PCI (i.e. no PCI_1). This also includes adding minimum system configuration information to Section 18.1 “Minimal System Configuration” on page 157. Corrected initial values for Table 78 “CS[3]* & BootCS* Low Decode Address, Offset: 0x038” on page 172 and Table 97 “CS[3]* & BootCS* Address Remap, Offset: 0x0e8” on page 176. The correct reset value is 0x7f8. New footnote in Table 173 “PCI_0 Command, Offset: 0xc00” on page 200 explaining that bits [12:10] are not supported in a 64-bit PCI configuration Corrected initial value explanation for bits [31:16] in Table 223 “PCI_0 Class Code and Revision ID, Offset: 0x008 from PCI_0 or CPU; 0x088 from PCI_1” on page 221. The initial value depends on the value sampled at reset on BankSel[0]. Revised operating current of 470 mA in Table 276on page 248. Corrected title in Table 277 “BGA Thermal Data” on page 249. Incorrectly labeled PQFP in previous revision. Revised PClk/TClk restrictions in Section 22.1 “TClk/PClk Restrictions” on page 257. New part numbering information in Section 24. “GT–64130 Part Numbering” on page 262.