MARVELL®

GT-64260A Design Guide

Doc. No. MV-S300165-00, Rev. A

May 21, 2002

MOVING FORWARD FASTER™

Document Status	
Advanced Information	This datasheet contains design specifications for initial product development. Specifications may change without notice. Contact Marvell Field Application Engineers for more information.
Preliminary Information	This datasheet contains preliminary data, and a revision of this document will be published at a later date. Specifications may change without notice. Contact Marvell Field Application Engineers for more information.
Final Information	This datasheet contains specifications on a product that is in final release. Specifications may change without notice. Contact Marvell Field Application Engineers for more information.
Revision Code:	
Preliminary	Technical Publication: 0.x

Marvell
700 First Avenue
Sunnyvale, CA 94089
Phone: (408) 222 2500
Sales Fax: (408) 752 9029
Email: commsales@marvell.com

Table of Contents

List of Tables

List of Figures

Section 1. Introduction

This design guide provides information for designing a system using the GT-64260A. This guide is intended for internal and external reference and is subject to future changes and modifications.

Use this document as an addition to the GT-64260A datasheet and evaluation/reference design schematics.

1.1 Related Documentation

The following documents are referenced by this design guide or provide additional information about working with the GT-64260A. See the Marvell® website at http://www.marvell.com to access this material.

- GT-64260A Datasheet, Doc. No. MV-S100413-00
- EV-64260A-BP-xxxxx Reference Platform Schematics
- PCI 2.2 Local Bus Specification
- AN-66: Initializing Ethernet Ports
- AN-67: Powering Up /Powering Down Galileo Technology Devices with Multiple Power Supplies of Different Voltages, Doc. No. MV-S300069-00
- AN-72: Operating the MPSC as a UART, Doc. No. MV-S300072-00
- AN-82: SDRAM Clocking Schemes in the GT-642xx/A Devices, Doc. No. MV-S300108-00
- BRG Settings Tool
- PCI Specification 2.2 - http://www.pcisig.com/specifications/conventional_pci

Section 2. GT-64260A Overview

The GT-64260A is a bridge from the PowerPC processor to the PCI bus, as well as a high-speed memory controller for external ROM and external peripherals. In addition, the GT-64260A integrates three 10/100 Mbps Ethernet ports and two MPSC controllers.

The GT-64260A provides a single-chip solution for designers building systems for a PowerPC 64-bit bus CPU. It has the following interfaces:

- A 64-bit interface to the CPU bus.
- A 64-bit interface to SDRAM.
- A 32-bit interface to devices (various types of memory and I/O devices).
- Two 64-bit PCI interfaces.
- Three RMII/MII interfaces.
- Two MPSC communication interfaces.

Figure 1 shows the GT-64260A interfaces.

Figure 1: GT-64260A Interfaces

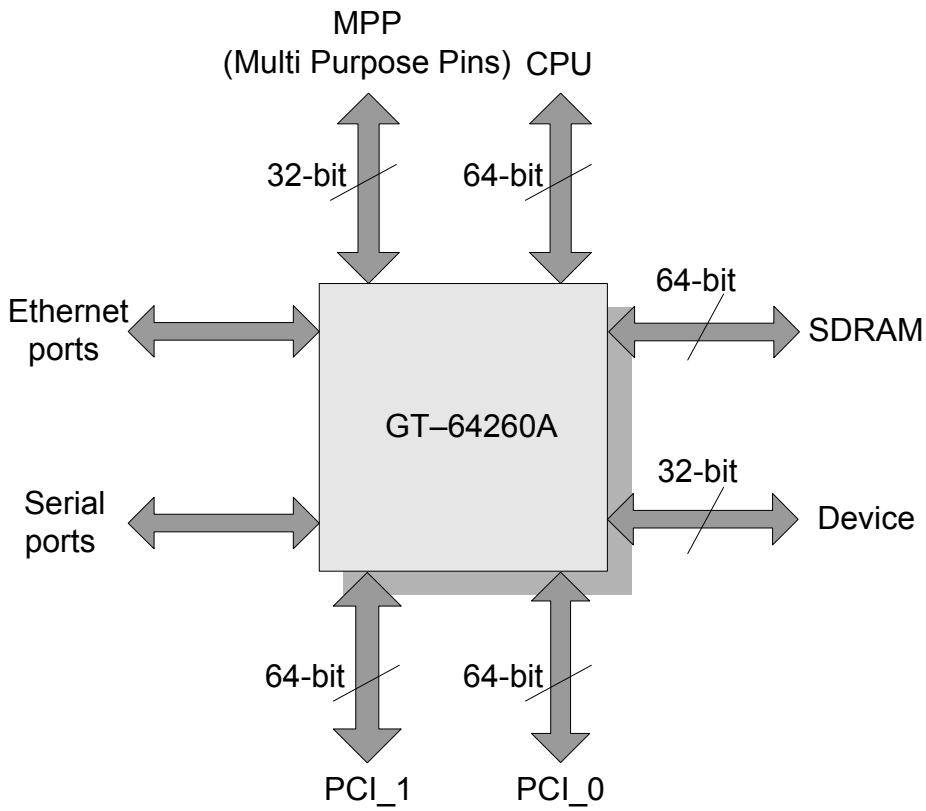
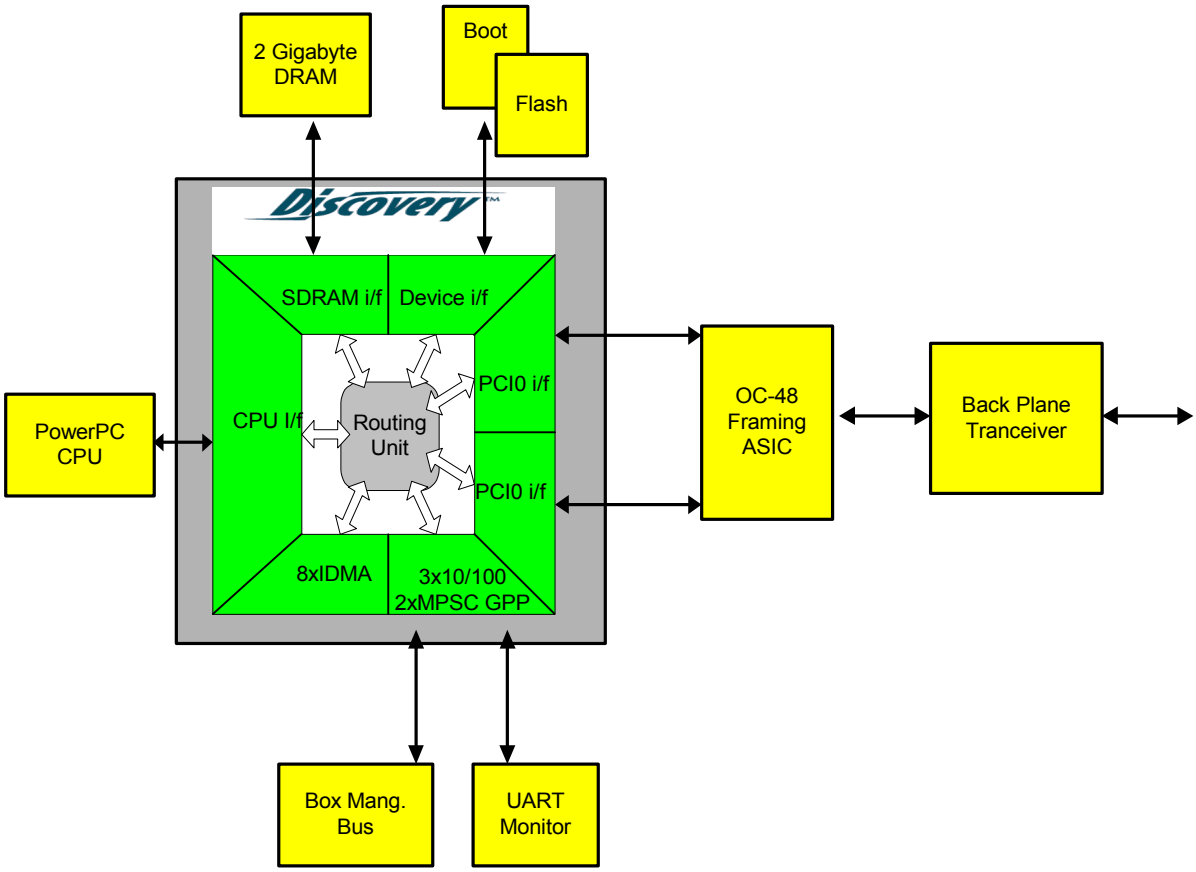


Figure 2: Typical GT-64260A System Configuration



Section 3. CPU Interface Functional Overview

The GT-64260A supports PowerPC 64-bit bus CPUs. (See Section 3.5 "Specific CPUs Aspects" .) These include the following:

- Motorola MPC603e/604e
- Motorola MPC740/750/755
- Motorola PowerQUICC II (MPC8260)
- Motorola MPC7400/7410/745x
- IBM PPC603e
- IBM PPC750/750cx/e/750FX
- Any 64-bit 60x or MPX compatible CPU

The CPU interface can work as a slave interface, responding to CPU transactions, or as a master interface, generating transactions on the CPU bus. The master interface is used for PowerPC snoop generation. It also allows for access to MPC8260 local memory or GT-to-GT transfers in a multi-GT-64260A configuration.

3.1 CPU Pinout Description

The GT-64260A provides all the pins needed to interface between the PowerPC processor and other devices (such as SDRAM, ROM, PCI, etc.). Generally, there is a point-to-point connection between the GT-64260A and the CPU. In other cases, it depends on the system architecture, such as multi-GT-64260A, multiple CPU, or external arbiter. The following table describes the pin information and details of the GT-64260A CPU interface.

Table 1: CPU Interface Pin Information

Pin Name	Input/ Output	PowerPC CPU Pin Connection	Required External Resistor	Description
A[0-31], AP[0-4]	T/S I/O	For address connection to CPUs that support 36-bit addressing, see 3.5.3 "MPC745x Extended Pins" on page 22. For all other CPUs with 32-bit address, connect to A[0-31], AP[0-4].	10K-Ohm Pull-up	CPU address bus and address parity bus. To work with address parity in the system, set the CPU Configuration register's APValid bit [26], at offset 0x000 to '1'.
DH[0-31], DL[0-31], DP[0-7]	T/S I/O	Some CPUs use D[0-63] for the entire data bus. In this case, use the following connection: • DH[0-31] = D[0-31] • DL[0-31]=D[32-63]	The GT-64260A includes internal Pull-ups.	CPU data bus and data parity bus. To work with data parity in the system, set the CPU Configuration register's DPValid bit [19], at offset 0x000 to '1'.

Table 1: CPU Interface Pin Information (Continued)

Pin Name	Input/ Output	PowerPC CPU Pin Connection	Required External Resistor	Description
SysClk	I	See Section 19. "Clocks" on page 143.	When the CPU interface is configured to run with TClk instead of SysClk, SysClk is not used and must be tied to GND.	CPU interface clock. The CPU clock configuration is selected by AD[5] value at reset de-assertion. 0 = SysClk asynchronous to TClk. 1 = CPU interface is running with TClk.
SysRst*	I	See Section 20. "Reset" on page 144.		The GT-64260A main reset pin. When in the reset state, all output pins (except for SDRAM address and control pins) are put into tri-state.
TS*	T/S I/O	TS*	10K-Ohm Pull-up	Address tenure start
TSIZ[0:2]	T/S I/O	TSIZ[0:2]	10K-Ohm Pull-up	Transfer size
TBST*	T/S I/O	TBST*	10K-Ohm Pull-up	Transfer burst
TT[0:4]	T/S I/O	TT[0:4]	10K-Ohm Pull-up	Transfer type
TA*	T/S I/O	TA*	In multi-GT mode, requires 10K-Ohm Pull-up.	Transfer acknowledge
AACK*	T/S I/O	AACK*	In multi-GT mode, requires 10K-Ohm Pull-up.	Address acknowledge
ARTRY*	T/S I	ARTRY*.	10K-Ohm Pull-up	Address retry. Not sampled on the second cycle after the AACK* assertion.
ABB*	T/S I/O	When working with PowerPC CPUs that use the ABB* pin as input, ABB* must be connected. Otherwise, ABB* must be pulled up.	10K-Ohm Pull-up	Address bus busy
DBB*	T/S I/O	When working with PowerPC CPUs that use the DBB* pin as input, DBB* must be connected. Otherwise, DBB* must be pulled up.	10K-Ohm Pull-up	Data bus is busy

Table 1: CPU Interface Pin Information (Continued)

Pin Name	Input/ Output	PowerPC CPU Pin Connection	Required External Resistor	Description
DTI[0-2]	T/S O	Only applicable on CPUs that support MPX bus. When using the MPC745x, connect to DTI[1-3] of the MPC745x. DTI[0] of the MPC745x must be pulled low. When using MPC741x, connect to DTI[0-2].	MPC740x/741x platforms: DTI[0]/DBWO* must be pulled up. DTI[1:2] must be pulled Low. MPC75x/PPC75x pin compatible to MPC740x/741x platforms: DTI[0]/DBWO* must be pulled up. DTI[1]/ARTRY* must be pulled up or connected to HRESET*. (See the MPC75x user manual) DTI[2] pulled down. For all other configurations 10K-Ohm pull-down.	Data transfer index.
BR0*/ GT_BG*	I	When using the GT-64260A internal arbiter, connect to the primary CPU BR* pin. When using an external arbiter, connect to the arbiter GT_BG* pin. **NOTE:** In single CPU systems with the internal arbiter enabled, must be used as BR0*.	To avoid unstable states at reset, a 10K-Ohm Pull-up is recommended.	
BG0*	T/S O	When using the GT-64260A internal arbiter, connect to the primary CPU BG* pin. When using an external arbiter, this pin can be left as not connected (NC). **NOTE:** In single CPU systems with the internal arbiter enabled, must be used as BG0*.	To avoid unstable states at reset or when an external arbiter is used, a 10K-Ohm Pull-up is recommended.	

Table 1: CPU Interface Pin Information (Continued)

Pin Name	Input/ Output	PowerPC CPU Pin Connection	Required External Resistor	Description
DBG0*	T/S O	When using the GT-64260A internal arbiter, connect the primary CPU DBG* pin. When using an external arbiter, this pin can be left as not connected (NC). **NOTE**: In single CPU systems with the internal arbiter enabled, must be used as CPU DBG0*.	To avoid unstable states at reset or when an external arbiter is used, a 10K-Ohm Pull-up is recommended.	
BR1*/ GT_DBG*	I	When using single CPU systems, this pin can be left as not connected (NC). When using the GT-64260A internal arbiter, connect to the secondary CPU BR* pin. When using an external arbiter, connect to the arbiter GT_DBG* pin.	To avoid unstable states at reset or in a single CPU configuration, a 10K-Ohm Pull-up is recommended.	
DBG1*	T/S O	When using an external arbiter or single CPU systems, this pin can be left as not connected (NC). When using the GT-64260A internal arbiter, connect to the secondary CPU DBG* pin.	To avoid unstable states at reset or when external arbiter is used, a 10K-Ohm Pull-up is recommended.	
BG1*/ GT_BR*	T/S O	When using single CPU systems, this pin can be left as not connected. When using the GT-64260A internal arbiter, connect to the secondary CPU BG* pin. When using an external arbiter, connect to the arbiter GT_BR* pin.	To avoid unstable states at reset or when external arbiter is used, a 10K-Ohm Pull-up is recommended.	

3.2 60x Bus Mode

The GT-64260A can act as master and slave on the 60x bus. In this mode, the GT-64260A 60x internal arbiter supports three masters on the bus; two external 60x compatible masters, and an internal 60x master. The GT-64260A is configured to 60x bus mode by having the AD[7:6] signals sampled to b'00' at reset de-assertion. The CPU bus configuration can be read in the CPU Mode register's (Offset: 0x120) CPUType bits [7:4].

On the 60x bus, the internal 60x master supports snoop and master capability for data transfers between the GT-64260A and other agents on the 60x bus, such as another GT-64260A device.

3.3 MPX Bus Mode

Note

The GT-64260A does not support multi-GT mode in MPX bus mode.

The GT-64260A can act as master and slave on the MPX bus. In this mode, the GT-64260A 60x internal arbiter supports two masters on the bus, one external MPX compatible master and the internal CPU bus master for snoop transaction only. The GT-64260A is configured to MPX bus mode by AD[7:6] signals sampled to b'01' at reset de-assertion. The CPU bus configuration can be read in the CPU Mode Register (Offset: 0x120) bits CPU-Type (bits 7:4).

Table 2 describes the MPX bus features that are supported by the GT-64260A.

Table 2: GT-64260A Supported Features in MPX Bus Mode

Features	Description
Address streaming	The CPU initiates a new address tenure the cycle after AACK* assertion without a dead cycle between the two address tenures.
Data streaming	If the data bus is driven by the same agent in both data tenures, no dead cycle is required between two consecutive data tenures.
16 byte burst	Load/store of AltiVec uncached operands.
Read out of order completion	The CPU read response data of pipelined transactions in any order. The DTI[0-2] pins indicates data tenure ID. • 000 = data of the oldest pending read. • 001 = the second oldest pending read. • 010 = the third oldest pending read. **NOTE:** To enable the read out of order completion, the CPU Configuration register's `RdOOO` bit [16] and `Pipeline` bit [13] must be set to '1', at offset 0x000.

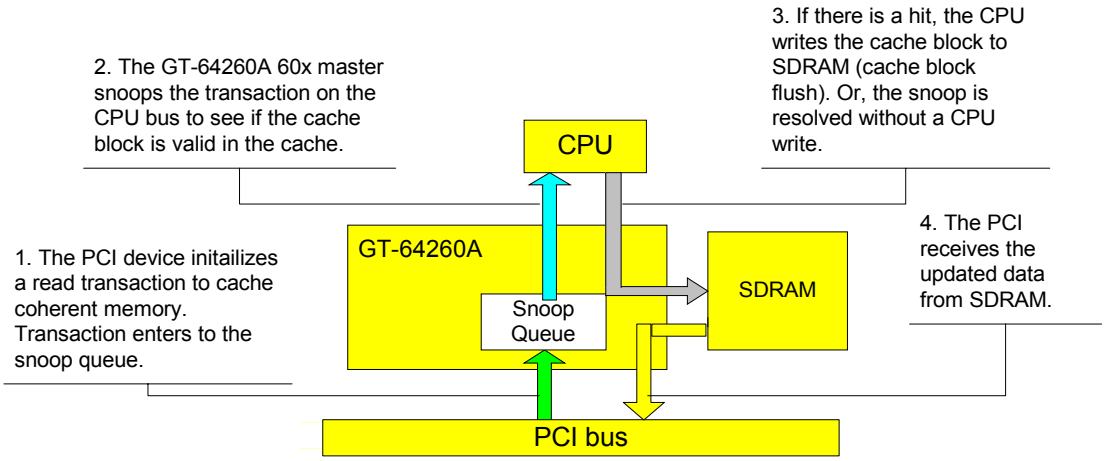
3.4 Cache Coherency

The GT-64260A supports full cache coherency between the SDRAM and CPU caches.

Any access to the SDRAM (from PCI or IDMA) may result in a snoop transaction driven by the GT-64260A on the CPU bus. The SDRAM access to a cache coherent region is always suspended until the snoop is resolved. In case of a HIT in a modified line in CPU cache, the SDRAM access might be suspended until the line write-back to SDRAM is completed.

Figure 3 describes the transaction flow for PCI reads from cache coherent regions.

Figure 3: PCI Reads from Cache Coherent Regions



The GT-64260A supports up to four SDRAM address windows in which IDMA cache coherency is maintained. The address windows do not correlate to specific chip selects and may cross CS boundaries.

The GT-64260A also supports eight configurational address ranges (four for each PCI interface) that help maintain PCI cache coherency.

Four pairs of base/top registers define the IDMA address regions.

Table 3: IDMA Address Base/Top Registers

Register	Offset
Snoop Base Address 0	0x380
Snoop Top Address 0	0x388
Snoop Base Address 1	0x390
Snoop Top Address 1	0x398
Snoop Base Address 2	0x3A0
Snoop Top Address 2	0x3A8
Snoop Base Address 3	0x3B0
Snoop Top Address 3	0x3B8

Each channel has a programmable bit per source, destination, and next descriptor pointer to enable/disable snoops in its Control (High) register's `SrcSnoopEn`, `DestSnoopEn` and `NextSnoopEn` bits. If these bits (or each one of these bits) are set, each IDMA engine transaction address is compared against the four cache coherency regions. If an address hits one of these regions, the DRAM access results in a snoop action, based on cache policy (WB/WT) as defined by the snoop regions registers Additionally, the CPU Master Control register's `CleanBlock` and `FlushBlock` bits [13:12] at offset: 0x160 must be set to the appropriate value depending on the CPU type.

The pairs of base/top registers define the PCI address regions as shown in the table below.

Table 4: PCI Address Base/Top Registers

Register	PCI_0 Offset	PCI_1 Offset
PCI Snoop Control Base 0 (Low)	0x1F00	0x1F80
PCI Snoop Control Base 0 (High)	0x1F04	0x1F84
PCI Snoop Control Top 0	0x1F08	0x1F88
PCI Snoop Control Base 1 (Low)	0x1F10	0x1F90
PCI Snoop Control Base 1 (High)	0x1F14	0x1F94
PCI Snoop Control Top 1	0x1F18	0x1F98
PCI Snoop Control Base 2 (Low)	0x1F20	0x1FA0
PCI Snoop Control Base 2 (High)	0x1F24	0x1FA4
PCI Snoop Control Top 2	0x1F28	0x1FA8
PCI Snoop Control Base 3 (Low)	0x1F30	0x1FB0
PCI Snoop Control Base 3 (High)	0x1F34	0x1FB4
PCI Snoop Control Top 3	0x1F38	0x1FB8

Each PCI transaction address is compared against the four cache coherency regions. If an address hits one of these regions, the DRAM access results in a snoop action, based on cache policy (WB/WT) and as defined by the snoop region registers. Additionally, the CPU Master Control registers `CleanBlock` and `FlushBlock` bits [13:12] at offset 0x160 must be set to the appropriate value depending on the CPU type.

3.4.1 Cache Coherency Initialization Sequence

The cache coherency initialization sequence is as follows:

1. The CPU Configuration register's `AACK Delay` bit [11], at offset 0x000, must be set to '1'. See the latest GT-64260A errata and restrictions document for more information.
2. In the CPU Master Control at offset 0x160, set the `CleanBlock` and `FlushBlock` bits [13:12] to the correct value, depending on the CPU type. See the GT-64260A datasheet's "CPU Interface" section for more information.
3. Configure cache coherent windows for the desired interface with the PCI cache coherency registers (0x1F00 - 0x1F38) and the IDMA cache coherency registers (0x380 - 0x3B8).
4. Confirm that the Snoop Control Base (Low) register's `Snoop` bits [13:12] at offset 0x1F00–0x1F30 are set to the correct value, depends on the CPU cache policy (WB or WT).

NOTE: The cache coherency will not be enabled when these bits are set to No snoop (0x0).

5. When using the PCI access control registers (base smaller than top) 0x1E00 - 0x1E78, the MBurst bits to the cache coherent memory must be set to 0x0 (four 64-bit words).

NOTE: Never configure the communication interface to work with cache coherent regions (see GT-64260A errata and restriction document).

It is the designer's responsibility to ensure that the CPU is configured correctly to support the cache coherency. For example, make sure that the W, I, M, G bits in the block attributes or page table entry indicate the correct setting of the processor cache policy.

3.5 Specific CPUs Aspects

3.5.1 MPC745x Burst to Boot Address

The MPC745x instruction queue holds as many as 12 instructions and loads as many as four instructions per cycle. After reset the MPC745x fetches instructions from the boot device (or any other cache-inhibited memory), in 60x-bus mode, the bus access is a 32 byte transaction (even though only the required 16 bytes are transmitted to the instruction queue). In MPX bus mode, a cache-inhibited instruction fetch performs a 16 byte transaction on the bus.

Since the MPC745x's first transaction after boot is 32/16 byte read and the GT-64260A does not support burst longer than 8 byte from 8-bits or 16-bits wide devices, implement one of the following solutions.

- Use 32-bit wide boot device.
- Use the Serial ROM Initialization at reset to copy 8-bit flash device to another 32-bit device (e.g. SRAM).

For example, the GT-64260A with the MPC745x evaluation platform (EV-64260ABP-MPC7450) uses an 8-bit wide flash device and a 32-bits wide SRAM device to boot the CPU. Before the CPU reset de-assertion, the 8-bit flash is copied to the 32-bits SRAM device by the I2C interface

Note

For more information on the serial ROM initialization, see the GT-64260A datasheet's "Reset Configuration" section.

System Initialization Sequence

The system initialization sequence is as follows:

1. On the device bus, implement one 8-bit flash memory (used as the boot device connected to one of the CS* signals) and a 32-bit wide SRAM on the boot chip select (BootCS*).
2. Treat the 8-bit flash memory as the boot device. This means the boot image must be burned in the flash memory.
3. Burn an I^2C EEPROM with a data sequence that initiates a DMA transfer from the 8-bit boot device to the 32-bit SRAM. See Appendix A. "I2C EEPROM Example" on page 147.
4. Configure the GT-64260A to serial ROM initialization enabled (AD[0] set to '1').

Note

When booting from an I^2C EEPROM, the CPU must be kept at reset as long as the initialization process takes place. This is accomplished by using the InitAck pin, driven by the GT-64260A during the initialization, on one of the MPP pins. The MPP pin that functions as InitAck pin must be pulled high and configured to function as InitAct at the I^2C EEPROM file.

If this procedure is correctly executed, the PPC745x boots from the 32-bit SRAM, instead of from the 8-bit flash device.

Note

Make sure that all the Reset configuration strapping required by the GT-64260A with serial ROM initialization are pulled to the correct values (the same value configured in the serial initialization). For more information, see the GT-64260A datasheet's "Reset Configuration" section.

3.5.2 MPC7410, MPC745x and PPC750CX/e Bus Voltage

The MPC7410, MPC745x and PPC750CX/e only supports 1.8V and 2.5V bus voltages (except for the 3.3V version of the MPC7410). Alternatively, the GT-64260A supports 2.5V and 3.3V CPU interface voltages.

Therefore, when using one of these CPUs, it must be configured to work in 2.5V on the CPU bus by configuring the CPU's BVSEL pin to the correct value at reset. In addition, the GT-64260A's CPU interface must be configured to 2.5V at reset by setting AD31 to '0'.

Note

AD31 must be pulled to the correct value at reset even when the serial ROM is enabled. This means the CPU interface voltage value (AD[31]) and the serial initialization value (AD[0]) must be the same.

3.5.3 MPC745x Extended Pins

The MPC745x CPU supports a 32-bit addressing mode and a 36-bit extended addressing modes. When extended physical addressing is disabled, the MPC745x drives the four most significant bits to zeroes.

Note

The four most significant bits are still sampled and must be actively pulled to zero if they are not being used in a system.

The MPC745x Address and address parity buses must be connected to the GT-64260A as follows:

- Pull MPC745x A[0:3] to b'0000'.
- Connect MPC745x A[4:35] to GT-64260A A[0:31].
- Pull MPC745x AP[0] to b'1'.
- Connect MPC745x AP[1:4] to GT-64260A AP[0:3].

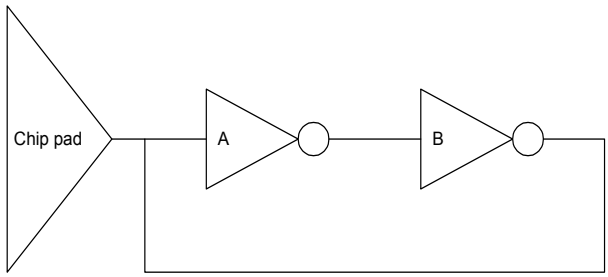
Additionally, the MPC745x microprocessor supports a 4-bit DTI index, with a maximum value of DTI[0:3] = b'1111'. Therefore, the DTI bus must be connected to the GT-64260A as follows:

- Pull MPC745x DTI[0] to b'0'.
- Connect MPC745x DTI[1:3] to GT-64260A DTI[0:2].

3.5.4 PPC750FX Level Protection

The PPC750FX CPU implements signal 'keepers' on some of its IO pads. The keeper can be viewed as two cross-coupled inverters. (See Figure 4). Inverter B is extremely weak. It can supply a maximum of ~50uA.

Figure 4: PPC750FX CPU Keeper



Since the 'keepers' are not biased to drive the IOs one way or the other, any control signals that must be in a particular state leaving HRESET* have to be either driven to that state or pulled there by external resistors.

The designer can connect two 'keepers' on the same signals (i.e. multiple PPC750FX systems), since they will always drive the same value. Theoretically, two keepers can end up in contention during power up. However, since this would be unstable, any noise event would knock it to a '1' or '0'.

The bus keeper can supply up to ~50uA. This means if there is a Pull-up (e.g., 10K-Ohm) and the last value was Low ('0'), the Pull-up will start pulling the signal high but the maximum current is 50 uA =>.

V(signal) = V(io) - [R(pull up) * I(maximum)] = 2.5 - [10000 * 0.00005]

V(signal) = 2V.

Note

IBM recommends using a 5K Ohm Pull-up that pulls the signal to 2.25V.

When the signal reaches the 'keeper' threshold voltage, the keeper drives the signals high and the signal voltage reaches the V(IO) value.

A similar calculation can be used for the pull down with keeper. In this case with 10K Ohm V(signal) = 0.5V (or 0.25V for 5K Ohm). When the signal reaches to the 'keeper' threshold voltage, the keeper starts driving the signals low and the signal voltage reaches the GND value.

Note

For more information, see the IBM datasheet.

3.5.5 PPC750FX Data Bus Parking

The GT-64260A internal 60x arbiter supports data bus parking.

See the GT-64260A errata and restriction document errata "FEr #CPU-1 Multiple data bus masters with IBM PPC750CX/e CPUs". This errata is also applicable to PPC750FX DD1.x Revisions. The dd2.0 Revision and higher will support data bus parking.

Note

PPC750FX first production revision is DD2.1. It is the revision that supports the data bus parking. Previous revisions are defined as prototypes.

3.6 Multi-GT or Multi-slave Modes

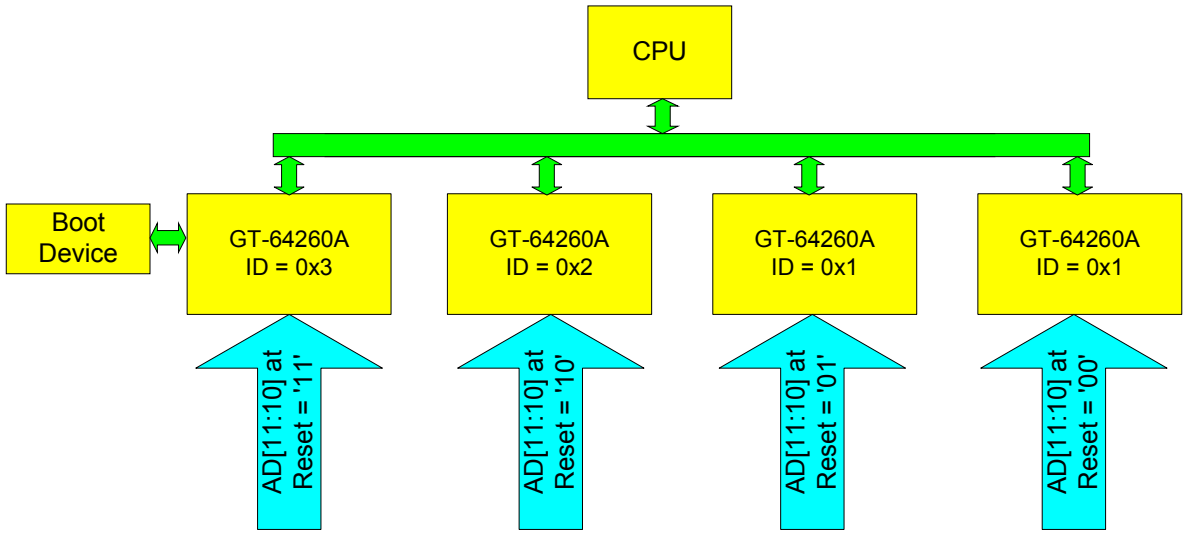
It is possible to connect up to four GT-64260A or other 60x-bus compliant slave devices to the 60x bus without the need for any glue logic.This capability (referred to as "Multi-GT" mode) adds significant system design flexibility. Multiple GT-64260A is enabled by setting AD[9] to ’1’ at reset.

Figure 5 describes the Multi-GT system architecture.

Note

Each GT-64260A in the figure can be removed or replaced by another 60x compatible slave.

Figure 5: Multi-GT System Architecture



Note

In multi-GT configuration, the AACK* and TA* pins function as sustained tri-state outputs requiring Pull-up resistors (see Table 1, “CPU Interface Pin Information,” on page 14). All TA* outputs from the GT-64260A devices must be tied together to drive the CPU TA* input. All AACK* outputs from the GT-64260A devices must be tied together to drive the CPU AACK* input.

When multi-GT is enabled, after reset de-assertion the GT-64260A uses a reduced address-decoding scheme as long as the CPU Configuration register’s `MultiGTDec` bit [18], at offset 0x000, is set to ‘1’. In this mode, each GT-64260A device has a two bit ID that is sampled at reset on AD[11:10] pins. Each device responds to the transaction address that matches its ID, as shown in Table 5:

Table 5: Multi-GT Device ID

ID	Multi-GT Address ID
00	A[5-6] = 00
01	A[5-6] = 01
10	A[5-6] = 10
11	A[5-6] = 11

Note

The boot GT-64260A ID must be programmed to ’11’.

The GT-64260A translates the CPU transactions as shown in Table 6.

Table 6: Multi-GT Mode Transaction Translation

AD[4]	Transaction Type	Transaction Decode
'0'	READ	Read from internal register when A[20-31] defining the specific register offset.
'1'	READ	Read access from boot device (BootCS*).
'x' (don't care)	WRITE	Write to internal register when A[20-31] defining the specific register offset.

For example, the boot GT-64260A (ID='11') responds to a read transaction from address 0xFFF00100. This transaction is translated to a read from the boot device at offset 0x100. A write transaction to address 0x00000000 gets a response from the GT-64260A configured as ID='00'. This transaction is translated to a write to its internal register offset at 0x000.

For more information on the multi-GT initialization sequence and system considerations see the following:

- The GT-64260A datasheet's "CPU Interface" section
- AN-91 Multi-GT Mode in the GT-6426xx

Notes

- When multi-GT is enabled at reset, the NoMatch counter is only applicable to the GT-64260A boot.
- In multi-GT mode AACK* and TA* pins have different AC timings. For details, see the GT-64260A datasheet's "AC Timing" section and the GT-64260A Documentation Updates and Changes document.

3.7 CPU Bus Multiple Masters

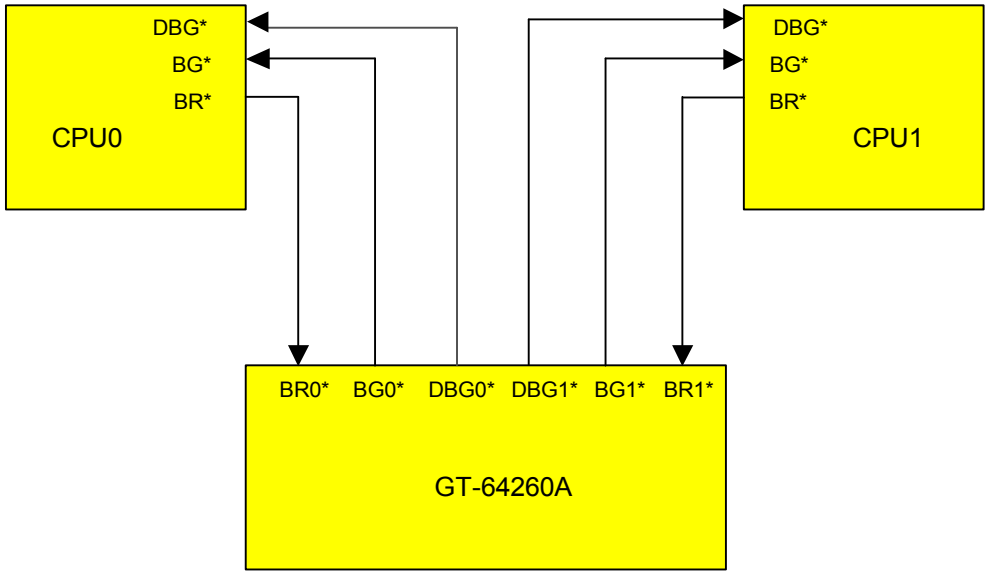
The PowerPC bus protocol supports separate arbitration on address and data busses. The arbitration must be implemented to support multiple masters on the CPU bus. The "multiple masters" configuration is usually used in a Symmetric Multiple Processing (SMP) system. For an SMP system to be fully functional, it requires additional HW implementations.

3.7.1 PowerPC Bus Arbitration

The GT-64260A supports both external arbiter and internal arbiter configurations. The arbiter configuration is sampled at reset on the AD8 pin (see the GT-64260A datasheet's "Reset Configuration" section).

If the internal arbiter is enabled, the BR0* pin is used as the CPU bus request input and BG0* and DBG0* are used as the CPU bus grant and data bus grant outputs. In this configuration, the arbiter bus requests are BR0*, BR1*, and CPU interface internal request (for bus mastering or snooping). Arbiter outputs are BG0* and DBG0* to one master, BG1* and DBG1* for second bus master, and internal bus grants and data bus grants for the internal CPU interface. The arbiter works in a fixed round robin scheme. Figure 6 shows the connection of two CPUs to the internal 60x arbiter.

Figure 6: 2 CPUs Connection Through Internal 60x Arbiter



Note

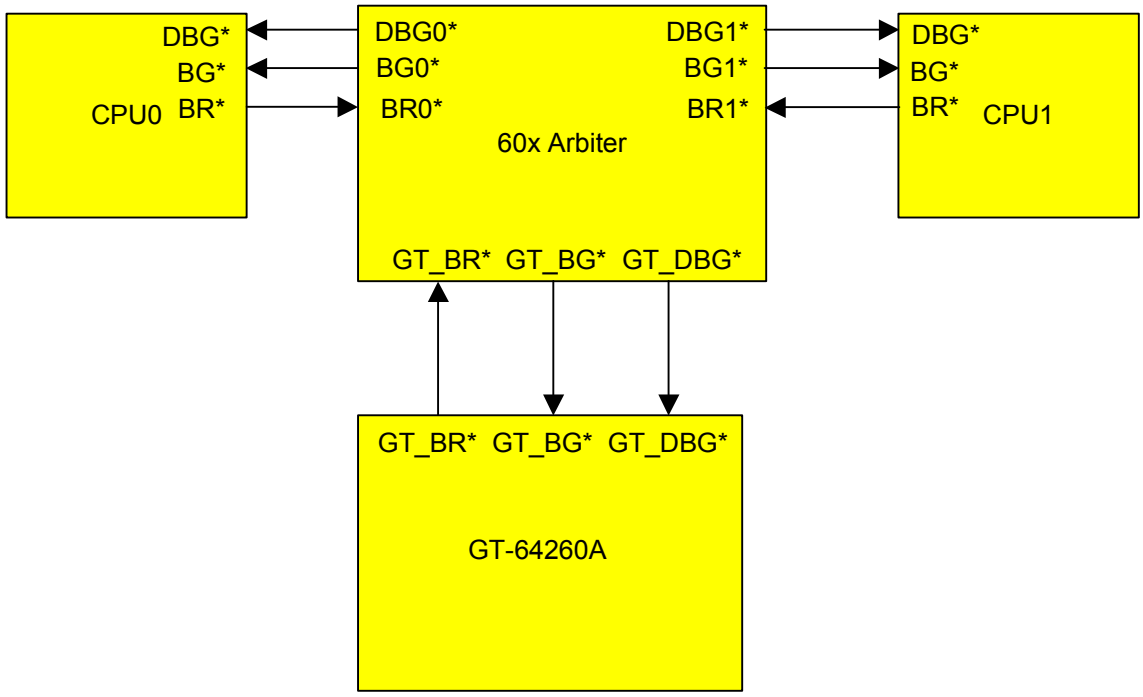
By default, the BR1* input is masked, enabling CPU0 to boot first. To enable CPU1 to access the bus, set the CPU Master Control register's `MaskBR1` bit [9], at offset 0x160 to '0'.

If the internal arbiter is disabled, an external arbiter must be used. In this mode:

- The BG1*/GT_BR* pin is used as the GT-64260A bus request output.
- The BR0*/GT_BG* pin is used as the GT-64260A bus grant.
- The BR1*/GT_DBG* pin is used as the GT-64260A data bus grant.

Figure 7 describes the connection of two CPUs and a GT-64260A to an external arbiter:

Figure 7: Two CPUs Connected Through an External Arbiter



Notes

- When using the IBM PPC750CX/e and some of the 750FX versions (see 3.5.5 "PPC750FX Data Bus Parking" on page 24) with multiple masters on the data bus an external 60x bus must be used. For more information, see the GT-64260A Restrictions and Errata document.
- For multiple CPU timing requirements, see Section 14.4.2 "Timing Simulation" on page 95.

3.7.2 Symmetric Multiple Processing Systems Requirements

Symmetric Multiple Processing (SMP) systems provide increased computational power in an industry-accepted manner that can be supported by numerous operating systems as well as middleware, such as database systems.

Boot Sequence

After reset pins de-assertion, each CPU begins executing a location in ROM. This is the start of the system firmware execution that eventually provides the interfaces to the operating system.

One of the first things that firmware does is establish one of the processors as the master. The master is a single processor that continues with the rest of the booting process. All of the other processors are placed in a stopped state. A processor in this stopped state does nothing that affects the state of the system and will remain in this state until accessed by an external event, such as an Inter-Processor Interrupts (IPIs).

The GT-64260A achieves the same goal by masking the BR1* pin from the internal arbiter. This causes the arbiter to think that only CPU0 needs the bus and it does not grant the bus to CPU1. To enable CPU1 arbitration, set the CPU Master Control register's `MaskBR1` bit [9], at offset: 0x160 to '0'.

Clock Synchronization

In a Symmetric Multiprocessor (SMP) system, the operating system needs the ability to synchronize the clocks on all the processors. To do this, the operating system must also have the ability to stop all clocks at the same time. The TBEN pin provided on the PowerPC CPUs can be used to implement this clock control function.

Instead of using a dedicated pin for this purpose, the GT-64260A uses an MPP pin to enable this synchronization. The MPP must be configured to function as a general purpose output and must be connected to all of the CPUs' TBEN pin.

Note

For more information, see Section 8. "Multi-Purpose Pin Interface Functional Overview" on page 74 and the corresponding section in the GT-64260A datasheet.

Inter-processor Communication

The processors communicate with each other through inter-processor interrupts (IPIs). IPIs can effectively schedule and control threads over multiple processors.

The GT-64260A supports one dedicated interrupt output pin to the CPU (Interrupt*). When using more than one CPU, the PCI0/1 interrupt or MPP output pins can be connected to the CPUs.

Note

When using the MPP interrupts (Int*[3:0]), the interrupt cause can be driven by only one interrupt cause register - main interrupt cause register Low (offset 0xC18) or main interrupt cause register High (offset 0xC68). See section Section 11. "Interrupt Controller Functional Overview" on page 82 or the GT-64260A datasheet's "Interrupt Controller" section.

The IPI is implemented by using the MPP interface in loops (CPU0 to CPU1, CPU1 to CPU0 etc.). Figure 8 describes the interrupt pins connectivity in SMP systems with a single GT-64260A and two CPUs.

Figure 8: Interrupt Pins' Connectivity

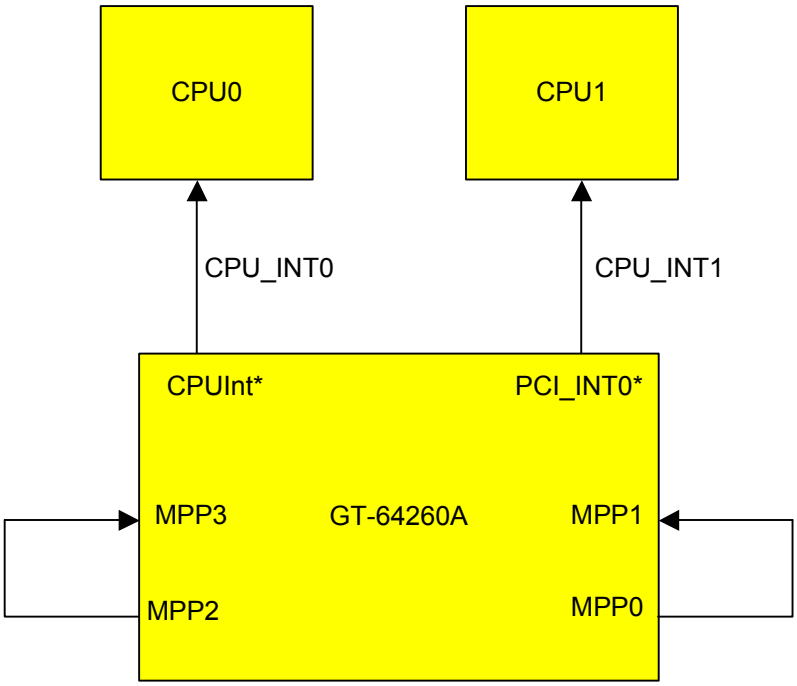
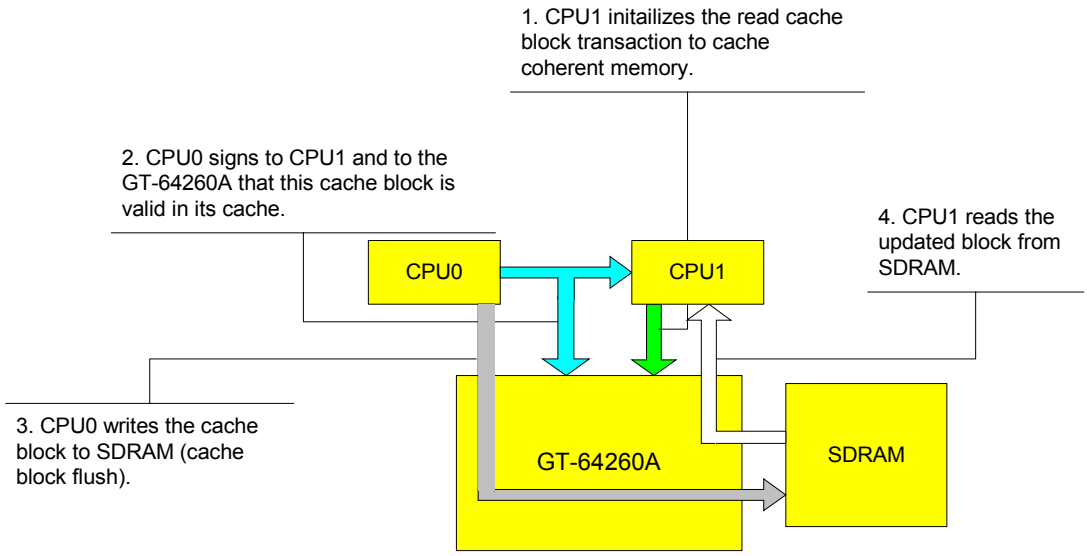


Figure 9 describes the CPU to CPU cache coherency data flow:

Figure 9: CPU to CPU Cache Coherency Data Flow



3.8 PowerPC COP/JTG Interface

The Power PC's common on-chip processor (COP) function allows a remote computer system (typically a PC with dedicated hardware and debugging software) to access and control the internal operations of the processor. The COP interface connects primarily through the JTAG port of the processor, with some additional status monitoring pins. The COP port must be able to independently assert HRESET or TRST to control the processor.

Figure 10 and Figure 11 illustrate the different HRESET and TRST connections that IBM and Motorola recommend.

Figure 10: IBM RISCWatch™ JTAG to HRESET, TRST, and SRESET pin Connector

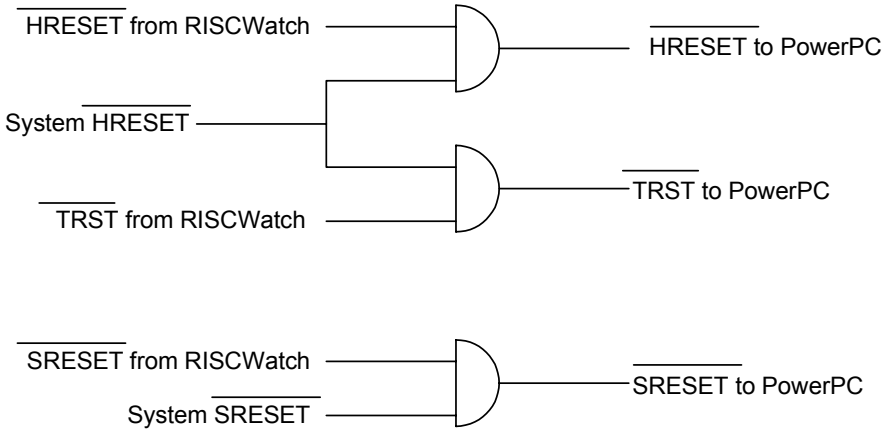
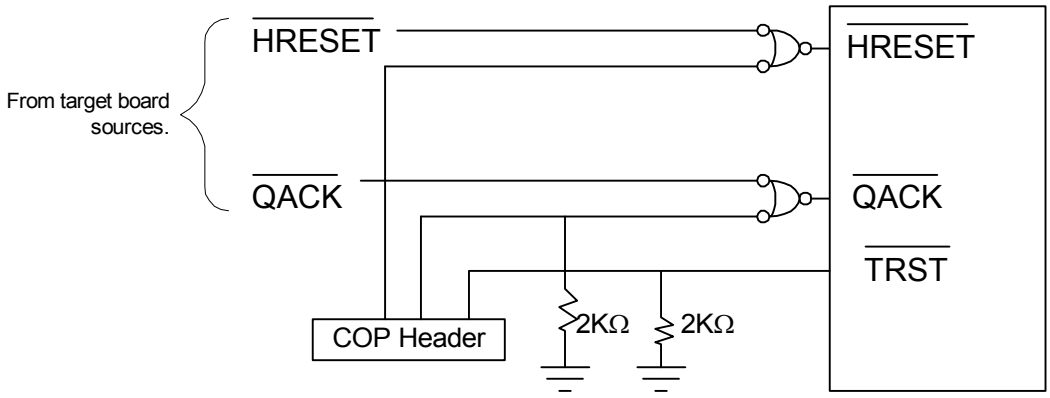
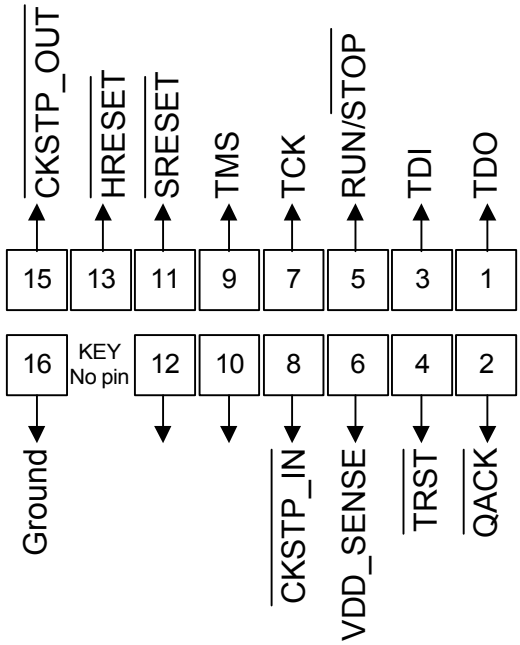


Figure 11: Motorola JTAG to HRESET and TRST pin Connector



It is recommended to implement both connections on the board since different ICE tools require different connections. Figure 12 describes the JTAG/COP 16-pins connectors.

Figure 12: JTAG/COP 16 Pin Connectors



Top View

Pins 10, 12, and 14 are no-connects.
Pin 14 is not physically present.

Note

The JTAG pins for the COP interface must not be chained with other devices in the system.

Section 4. SDRAM Interface Functional Overview

The GT-64260A SDRAM controller supports 16/64/128/256/512 MB density SDRAM and registered SDRAM at 133 MHz. It also supports up to four banks of SDRAM and can address up to 4 GB (1 GB per bank – physical SCS*).

4.1 Pinout Description

Table 7: SDRAM Interface Pinout Description

Pin Name	Input/ Output	SDRAM Device or DIMM Connector	Required External Resistor	Description
SDClkOut/ SDClkIn	IO	This pin functionality is sampled at reset on AD23. See Section 19. "Clocks" on page 143 for more information on the SDRAM clock scheme.	No.	The SDRAM controller can be programmed to drive the SDRAM clock or to sample the data with the SDRAM clock to overcome board layout issues at high frequencies.
SRAS*	t/s O	When using DRAM DIMMs, connect to the RAS* pin. When using SDRAM, devices must be connected to RAS* pin of each device.	10K-Ohm pull up is required only in UMA mode.	SDRAM Row Address Select. This pin is tri-stated only in UMA mode.
SCAS*	t/s O	When using DRAM DIMMs, connect to the CAS* pin. When using SDRAM, devices must be connected to CAS* pin of each device.	10K-Ohm pull up is required only in UMA mode.	SDRAM Column Address Select. This pin is tri-stated only in UMA mode.
DWr*	t/s O	When using DRAM DIMMs, connect to the DWr* pin. When using SDRAM devices, must be connected to DWr * pin of each device.	10K-Ohm pull up is required only in UMA mode.	SDRAM Write. This pin is tri-stated in UMA mode only.

Table 7: SDRAM Interface Pinout Description (Continued)

Pin Name	Input/ Output	SDRAM Device or DIMM Connector	Required External Resistor	Description
DAdr[12:0]	t/s O	When using DRAM DIMMs, connect to the DAdr pins. When using SDRAM devices, must be connected to the DAdr pins of each device. When using 16 Mb devices, use only Dadr[10:0]. When using 64 or 128 Mbit devices, use only Dadr[11:0]. When using 256 or 512 Mb devices, use only Dadr[12:0].	10K-Ohm pull up is required only in UMA mode.	SDRAM Address **NOTE:** For more information on the address connection, see 4.2 "Memory Connection" on page 35). This pin is tri-stated only in UMA mode.
BankSel[1:0]	t/s O	When using DRAM DIMMs, connect to the BankSel pins. When using SDRAM, devices must be connected to BankSel pins of each device. When using 16 Mb devices, use only BankSel [0]. When using 64, 128, 256 or 512 Mbit devices, only use BankSel [1:0].	10K-Ohm pull up is required only in UMA mode.	SDRAM Bank Select. **NOTE:** For more information on the address connection, see 4.2 "Memory Connection" on page 35) These pins are tri-stated only in UMA mode.
SCS[3:0]*	t/s O	Each SCS* pin must select 64-bits wide data + 8-bits ECC (if used). When using DRAM DIMM, connect to the physical bank (CS0* and CS2*) or (CS1* and CS3*) pins, because each one selects only 32-bit data width. When using SDRAM devices, must be connected to CS* pin of each device. All devices connected to the same SCS* pin must be connected in parallel to achieve 64-bit wide data.	10K-Ohm pull up is required only in UMA mode.	SDRAM Chip Select These pins are tri-stated only in UMA mode.
SDQM[7:0]*	t/s O	When using DRAM DIMMs, connect to the SDQM pins. When using SDRAMs, devices must be connected to the SDQM pin of each device.	In UMA mode the master must drive these pins.	SDRAM Data Mask.

Table 7: SDRAM Interface Pinout Description (Continued)

Pin Name	Input/ Output	SDRAM Device or DIMM Connector	Required External Resistor	Description
SData[63:0]	t/s I/O	When using DRAM DIMMs, connect to the data pins. When using SDRAM devices, must be connected to data pins of each device in parallel to achieve 64-bits data width.	In UMA mode the master must drive these pins.	SDRAM Data
ECC[7:0]	t/s I/O	When using DRAM DIMMs, connect to the DP pin. When using SDRAM devices, must be connected to ECC bank device. **NOTE:** Since the SDRAM controllers do not write partials when ECC is enabled, the ECC bank device can be connected to any of the SDQM pins.	In UMA mode the master must drive these pins.	SDRAM ECC

Note

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

4.2 Memory Connection

The GT-64260A supports 16, 64, 128, 256 and 512 MB SDRAM devices. Each SDRAM physical bank (SCS[3:0]) can be built from different SDRAM devices. The DRAM density is configured via the DRAM Bank Parameter registers.

See Table 8 for details on how the various DRAM device sizes differ in the usage of DAdr[12:0] and BankSel[1:0] lines.

Table 8: SDRAM Interface Pinout Description

SDRAM	Addressing	x4	x8	x16	x32
16 Mb (2 virtual banks)	Row	A0 - A10	A0 - A10	NA	NA
	Column	A0 - A9	A0 - A8	NA	NA
64 Mb (4 virtual banks)	Row	A0 - A11	A0 - A11	A0 - A11	A0 - A10
	Column	A0 - A9	A0 - A8	A0 - A7	A0 - A7

Table 8: SDRAM Interface Pinout Description (Continued)

SDRAM	Addressing	x4	x8	x16	x32
128 Mb (4 virtual banks)	Row	A0 - A11	A0 - A11	A0 - A11	A0 - A11
	Column	A0 - A9, A11	A0 - A9	A0 - A8	A0 - A7
256 Mb (4 virtual banks)	Row	A0 - A12	A0 - A12	A0 - A12	A0 - A12
	Column	A0 - A9, A11	A0 - A9	A0 - A8	A0 - A7
512 Mb (4 virtual banks)	Row	A0 - A12	A0 - A12	A0 - A12	NA
	Column	A0 - A9, A11, A12	A0 - A9, A11	A0 - A9	NA

Table 8 describes different address pins during the RAS* and CAS* cycles. This indicates that when using a "wider" (more data lines) device, less devices are needed, in total, to achieve the 64-bit data width. For example, to use a 64 MB SDRAM device with a 8-bits wide configuration, connect eight devices in parallel to get 64-bits data width ➔ 64 MB-memory space. Or, when using the same device in 16-bits wide configuration, connect four devices in parallel in order to get 64-bits data width ➔ 32 MB-memory space. The example shows that the widest SDRAM device provides the smallest memory space (for a given SDRAM density) because less devices are used.

Table 9 describes the SDRAM memory space for various SDRAM devices:

Table 9: SDRAM Memory Space

Device Density	Device Width	Memory Space per SCS*	Total Memory Space SCS*[3:0]
16 Mb	X4	16 Mbit x 16 devices = **32 MB**	32 MB x 4 = **128 MB**
16 Mb	X8	16 Mbit x 8 devices = **16 MB**	16 MB x 4 = **64 MB**
64 Mb	X4	64 MB x 16 devices = **128 MB**	128 MB x 4 = **512 MB**
64 Mb	X8	64 MB x 8 devices = **64 MB**	64 MB x 4 = **256 MB**
64 Mb	X16	64 MB x 4 devices = **32 MB**	32 MB x 4 = **128 MB**
64 Mb	X32	64 MB x 2 devices = **16 MB**	16 MB x 4 = **64 MB**
128 Mb	X4	128 MB x 16 devices = **256 MB**	256 MB x 4 = **1024 MB (1 GB)**
128 Mb	X8	128 MB x 8 devices = **128 MB**	128 MB x 4 = **256 MB**
128 Mb	X16	128 MB x 4 devices = **64 MB**	64 MB x 4 = **128 MB**
128 Mb	X32	128 MB x 2 devices = **32 MB**	32 MB x 4 = **64 MB**
256 Mb	X4	256 MB x 16 devices = **512 MB**	512 MB x 4 = **2048 MB (2 GB)**
256 Mb	X8	256 MB x 8 devices = **256 MB**	256 MB x 4 = **1024 MB (1 GB)**
256 Mb	X16	256 MB x 4 devices = **128 MB**	128 MB x 4 = **512 MB**
256 Mb	X32	256 MB x 2 devices = **64 MB**	64 MB x 4 = **128 MB**
512 Mb	X4	512 MB x 16 devices = **1024 MB**	1024 MB x 4 = **4096 MB (4 GB)**
512 Mb	X8	512 MB x 8 devices = **512 MB**	512 MB x 4 = **2048 MB (2 GB)**

Table 9: SDRAM Memory Space (Continued)

Device Density	Device Width	Memory Space per SCS*	Total Memory Space SCS*[3:0]
512 Mb	X16	512 Mb x 4 devices = **256 MB**	256 MB x 4 = **1024 MB (1 GB)**

Figure 13 shows the SDRAM connection for regular SDRAM or heavy load modes. It uses four, 16-bit wide devices connected in parallel to achieve a 64-bit data path.

Figure 13: SDRAM Connection for Regular SDRAM/Heavy Load Mode

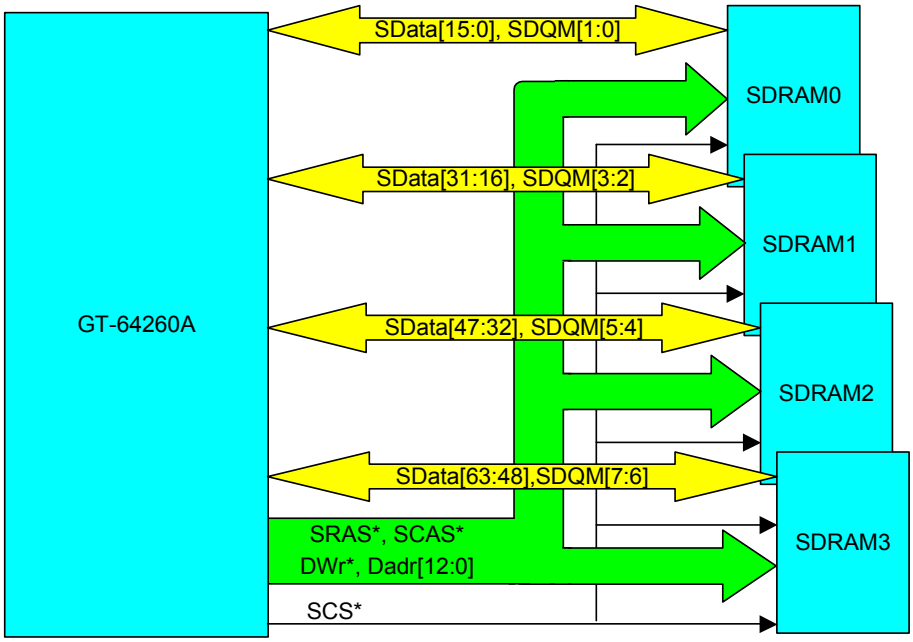
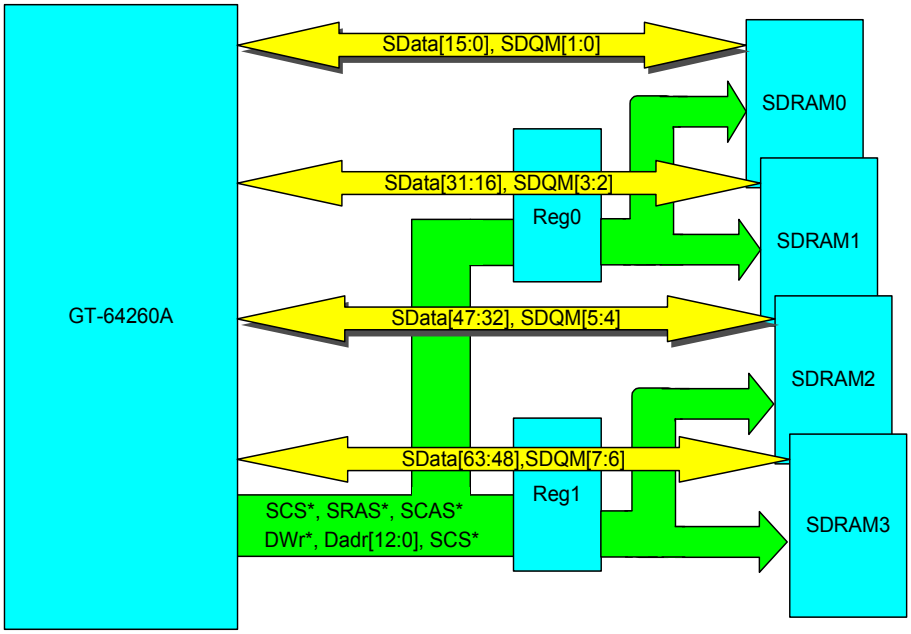


Figure 14 describes the SDRAM connection for registered SDRAM mode. It uses four, 16-bit wide devices connected in parallel to achieve a 64-bit data path.

Figure 14: SDRAM Connection for Registered SDRAM Mode



Both examples use one SCS* pin (one physical SDRAM bank). When using more than one SDRAM device, all of the pins must be connected in parallel to physical SDRAM banks, except for the SCS* pin. Different physical banks must be connected to different SCS* pins.

4.2.1 Connecting ECC Memory Bank

The SDRAM ECC memory bank must be connected in parallel to the SDRAM data memory. This means connection to the same DAdr, BA, RAS, CAS, DWr and CS pins. The ECC bank can be connected to any one of the SDQM pins. However, it is recommended to use the closest one to the ECC bank for higher signal integrity.

When working with ECC enabled, all transactions on the SDRAM interface are 64-bits wide. This is true for partial transactions or bursts, since the GT-64260A uses the Read Modify Write mechanism. This means all SDQMs are asserted on each transaction. On partial reads (smaller than 8B), the SDRAM controller reads all 64-bits of data and 8-bit ECC (72-bit wide) to check the ECC.

On partial writes, the GT-64260A reads all 64-bits of data and 8-bit ECCs to check the ECC. The GT-64260A then modifies the data bytes (less than 8B) and the ECC bank writes the updated data back.

On 8 bytes writes and bursts, the SDRAM controller writes all 64-bits of data and the corresponding 8-bit ECC without using the Read Modify Write mechanism.

Figure 10 shows the ECC bank selection for each ECC bank per SCS* with various SDRAM configurations. The guidelines for the ECC bank device selection are as follows:

- ECC bank capacity must be at least 1:8 of SDRAM capacity.
- ECC bank device type must be compatible to the SDRAM type. This means in the same group defined by the SDRAM Bankx Parameters register's `SDType` bits [15:14], at offsets 0x44c, 0x450, 0x454, 0x458.

Table 10: ECC Bank Selection

Device Density	Device Width	ECC Devices per SCS
16 Mb	X4	2 x 16 Mb (4-bit wide)
16 Mb	X8	1 x 16 Mb (8-bit wide)
64 Mb	X4	1 x 128 Mb (8-bit wide)
64 Mb	X8	1 x 64 Mb (8-bit wide)
64 Mb	X16	1 x 64 Mb (8-bit wide) – 50% used
64 Mb	X32	1 x 64 Mb (8-bit wide) – 25% used
128 Mb	X4	2 x 128 Mb (4-bit wide)
128 Mb	X8	1 x 128 Mb (8-bit wide)
128 Mb	X16	1 x 64 Mb (8-bit wide)
128 Mb	X32	1 x 64 Mb (8-bit wide) – 50% used
256 Mb	X4	1 x 512 Mb (8-bit wide)
256 Mb	X8	1 x 256 Mb (8-bit wide)
256 Mb	X16	1 x 256 Mb (8-bit wide) – 50% used
256 Mb	X32	1 x 256 Mb (8-bit wide) – 25% used
512 Mb	X4	2 x 512 Mb (4-bit wide)
512 Mb	X8	1 x 512 Mb (8-bit wide)
512 Mb	X16	1 x 256 Mb (8-bit wide)

4.3 SDRAM Address Control

The Address Control Register, at offset 0x47C, is a four-bit register that determines how address bits driven by the CPU, PCI, or DMA to the SDRAM controller are translated to row and column address bits on DAdr[12:0] and BankSel[1:0]. This flexibility allows the setting of the specific address decode setting which gives the software a better chance of virtual banks interleaving and enhances overall system performance.

Note

The row and column address translation is different for 16 Mb, 64/128Mb, or 256/512 MB SDRAMs. For more information, see the GT-64260A datasheet's "SDRAM Density" section.

4.4 SDRAM Initialization

The DRAM controller executes the SDRAM initialization sequence as soon as the GT-64260A goes out of reset (SysRst* de-assert). The DRAM controller performs a MRS (Mode Register Setting) cycle based on the default DRAM parameters (CL = 3, burst length = 4, burst order = linear). If the serial ROM initialization is enabled, the DRAM controller postpones the above DRAM initialization sequence until the serial ROM initialization completes.

To achieve better performance, the software can change CL to '2' if the DRAM device is capable of this CAS latency. Changing the CL requires setting the mode register of the SDRAM device. To update the mode register on the SDRAM, use the following procedure:

1. Write the new configuration data to the SDRAM Timing Parameters register (Offset: 0x4B4).
2. Write the value 0x3 (mode register command enable) to the SDRAM Operation Mode register's `SDRAMOp` bits [2:0], at offset 0x474.
3. Read the SDRAM Operation Mode register. This read guarantees that the following step is executed after the register value is updated.
4. Perform dummy word (32-bit) writes to an SDRAM bank. This eventually causes the required cycle to be driven to the selected DRAM bank.
5. Poll the SDRAM Operation Mode register's `Active` bit [31], at offset 0x474, until it is sampled '1'.This indicates that the MRS cycle is done.
6. Write a 0x0 value to the `SDRAMOp` bits. This value returns the register to Normal SDRAM Mode.
7. Read the SDRAM Operation Mode register. This read guarantees the execution of the following access to the DRAM, after the register value is updated.

For an example of this code, see Appendix B. "SDRAM Mode Register/Code" on page 149.

4.5 ECC Support

Note

For more information about the ECC operation and error report, see the GT-64260A datasheet's "SDRAM ECC" section.

The GT-64260A supports a force ECC capability. When the SDRAM ECC Control register's `ForceECC` bit [8], at offset: 0x494, is set to '1', any write to the SDRAM also writes to the ECC bank the data defined in the ForceECC field.

Considerations when using the force ECC feature:

- Use this feature for ECC initialization, only. Do not use it in regular operation modes.
- When using force ECC, only 64-bit transaction are allowed.
- When initializing the ECC, only run the code from cacheable memory space or from the ROM/RAM devices. Do not run the code from SDRAM.

If the SDRAM controller detects an ECC error, the controller can generate an interrupt to the CPU. Writing of 0x0 to the ErrType bits [1:0] in the SDRAM Error Address register clears the interrupt.

Note

The error type can hold a value of 0x3. This means that the SDRAM controller detected at least two ECC error events, when one of them is a single-bit error event and the other is double-bit error event. In this

case, the ECC error address register latches the first event address. The Interrupt Service Routine (ISR) can easily detect if the first event is the latched event with one or two errors by clearing the ECC error address register and then reading the latched address. If the address is latched again, the first event was two errors since the SDRAM controller did not fix the error. If the address is not latched, the first event was one ECC error but the SDRAM controller fixed it.

4.5.1 ECC Bank Initialization

To work with the ECC support, the ECC bank must be initialized.

Use one of the following procedures to initialize the ECC bank.

SDRAM Size Is Known

Note

Useful when SDRAM devices are assembled on board.

1. When running code from the boot device, configure the address space of the SDRAM interface to fit the memory size and configuration (all physical banks must be initialized). For more information on the address decode settings, see the GT-64260A datasheet's "Address Space Decoding" section.
2. Set to '0' the SDRAM ECC Control register's `ErrProp` bit [9] and the `ThrEcc` bits [23:16], at offset 0x494. This prevents the GT-64260A from generating an interrupt or propagating the ECC error to other interfaces.
3. In the SDRAM Timing Parameters register, at offset 0x4b4, set the `ECCEn` bit [13] to '1' and the `RdSample` bit [14] to '0'.
4. Perform 64-bit or burst write transactions to all of the memory space. Use IDMA engines to shorten the initialization sequence (e.g., copy the SDRAM to itself and use a data transfer limit of 128B).
5. To clear the ECC error report, write 0x0 to the SDRAM Error Address register's `ErrType` bits [1:0], at offset: 0x490.
6. Set to a value other than '0' (depending on the SW architecture) the `ErrProp` bit [9] and the `ThrEcc` bits [23:16] in the SDRAM ECC Control register. This enables the ECC interrupt to be generated and/or propagates the error to other interfaces.

SDRAM Size Is Flexible and Can Be Changed

Note

Useful when SDRAM devices are assembled on DIMM connectors.

1. When running code from the boot device, configure an address space in the SDRAM interface needed for the first operation of the operating system. For more information on the address decode settings, see the GT-64260A datasheet's "Address Space Decoding" section.
2. Set the SDRAM ECC Control register's `ErrProp` bit [9] and the `ThrEcc` bits [23:16], at offset 0x494 to '0'. This prevents the GT-64260A from generating an interrupt or propagating the ECC error to other interfaces.
3. In the SDRAM Timing Parameters register, at offset 0x4b4, set the `ECCEn` bit [13] to '1' and the `RdSample` bit [14] to '0'.
4. Perform 64-bit or burst write transactions to the memory space defined in step 1. It is recommended to use IDMA engines to shorten the initialization sequence (e.g., copy the SDRAM to itself and use a Data Transfer Limit of 128B).

5. To clear the ECC error report, write 0x0 to the SDRAM Error Address register's `ErrType` bits [1:0], at offset: 0x490.
6. Use the I^2C interface to detect the memory size, type, and number of physical banks. It is also possible to detect the memory size by opening the SDRAM memory space to the maximum (each SCS* at a time) and the check memory aliasing. To use this method, the detection code must not use the SDRAM (instruction and data) and the ECC must be disabled.
7. Perform 64-bit or burst write transactions to all of the memory space. It is recommended to use IDMA engines to shorten the initialization sequence (e.g., copy the SDRAM to itself and use a data transfer limit of 128B).
8. Set to a value other than '0' (depending on the SW architecture) the `ErrProp` bit [9] and the `ThrEcc` bits [23:16] in the SDRAM ECC Control register. This enables the ECC interrupt to be generated and/or propagates the error to other interfaces.

For example, the GT-64260A evaluation platform (EV-64260A-BP) DINK32 monitor ECC initialization sequence for various SDRAM sizes is as follows. The example code for the ECC initialization is found in Appendix C.

1. While running from the boot device, enable the ECC by setting the `ECCEn` bit [13] to '1'.
2. Use the IDMA engines, with DTL of 8B, to copy the boot code to the SDRAM from the boot device. It is important to use DTL = 8 Bytes to prevent the GT-64260A from performing a Read Modify Write to the SDRAM before the ECC initialization. The DTL cannot be set to larger values since the boot device is 8-bits wide.
3. Confirm that there is enough memory space with initialized ECC for the stack.
4. Run code from SDRAM to detect all the memory space (detection though the I^2C interface).
5. Initialize the ECC bank to all the memory space by using the IDMA with DTL of 8B.

Note

The Error Address register's bits [31:2], at offset 0x490, are cleared by reading the register. The register will not latch a new address before it is cleared.

4.6 Memory Banks and Pages

The GT-64260A supports both physical banks (SCS[3:0]*) and virtual banks (BankSel[1:0]).

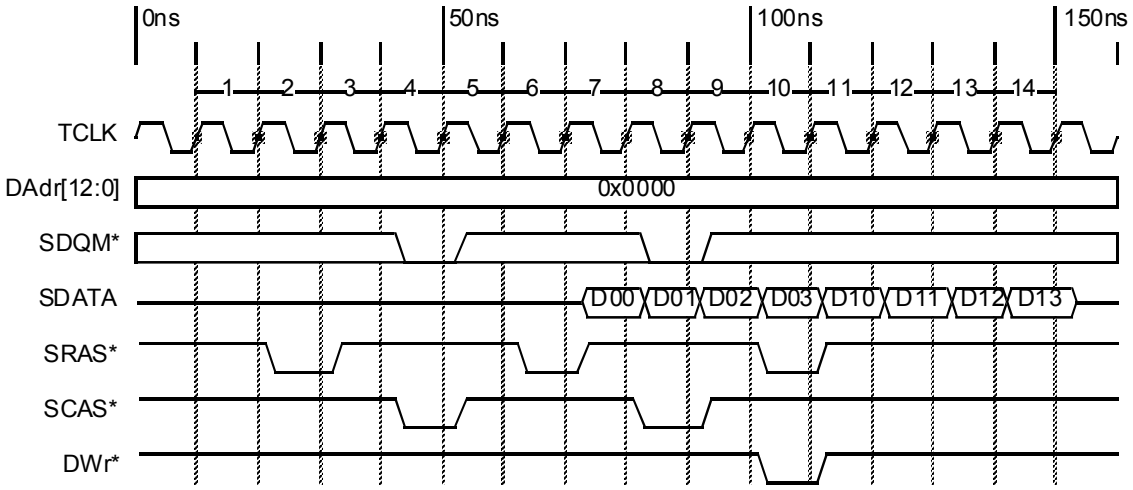
Interleaving is supported between all 16 banks (4 physical * 4 virtual). The physical interleaving is enabled by setting the SDRAM Configuration register's `PhInterEn` bit [15], at offset: 0x448 to '0'. The virtual interleaving is enabled by setting `VinterEn` bit [14] in the same register to '0'. Both interleaving methods are enabled by default after reset de-assertion.

Note

The GT-64260A only supports two virtual banks interleaving with 16 Mbit SDRAM devices since it has only a single bank select pin (BankSel[0]).

The figure below shows the bank interleaving between two reads from different virtual banks.

Figure 15: Two Read Interleaving from Different Virtual Banks

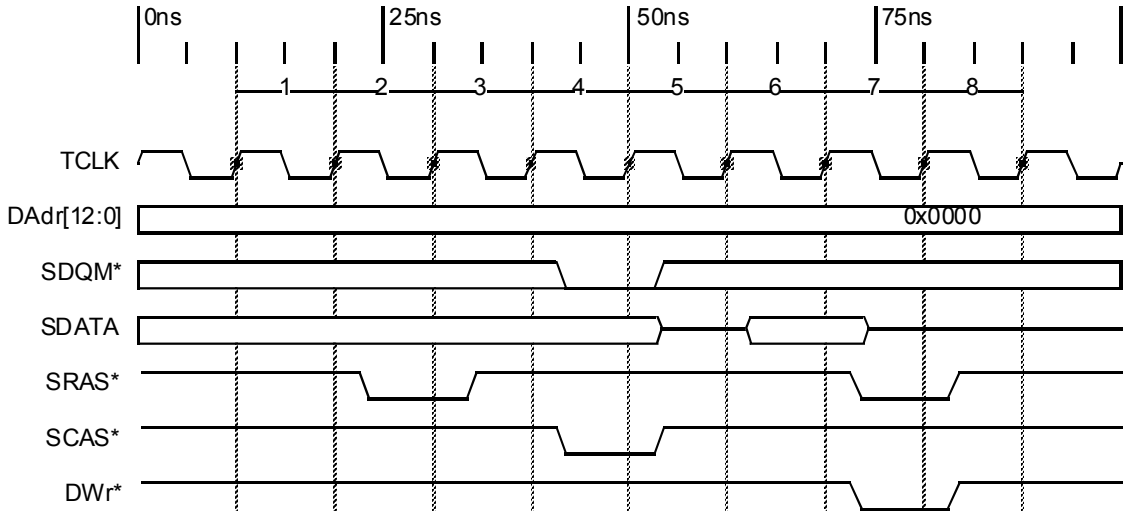


- Cycle 2: Bank0 - Row activating.
- Cycle 4: Column cycle (command).
- Cycle 6: Row activating.
- Cycle 7: Fetch first data (CL = 3 cycles).
- Cycle 8: Column cycle (command).
- Cycle 10: Precharge (close bank).
- Cycle 11: Fetch first data (CL = 3 cycles).

In addition, the SDRAM controller supports open pages. This means DRAM pages can stay open. The controller supports up to 16 open pages - one per each virtual bank. Each ROW address in the SDRAM defines a page and the COLUMN selects the data in the current open page. When a page is kept open at the end of a burst (no pre-charge cycle) and if the next cycle to the same virtual bank hits the same page (same row address), there is no need for a new activate cycle. To enable open pages, set the SDRAM Bankx Parameters register's `OpenP` bits [3:0], at offset 0x44c, 0x450, 0x454 and 0x458 to '1'.

The figure below shows a single read access to a non-open page when the open pages are disabled.

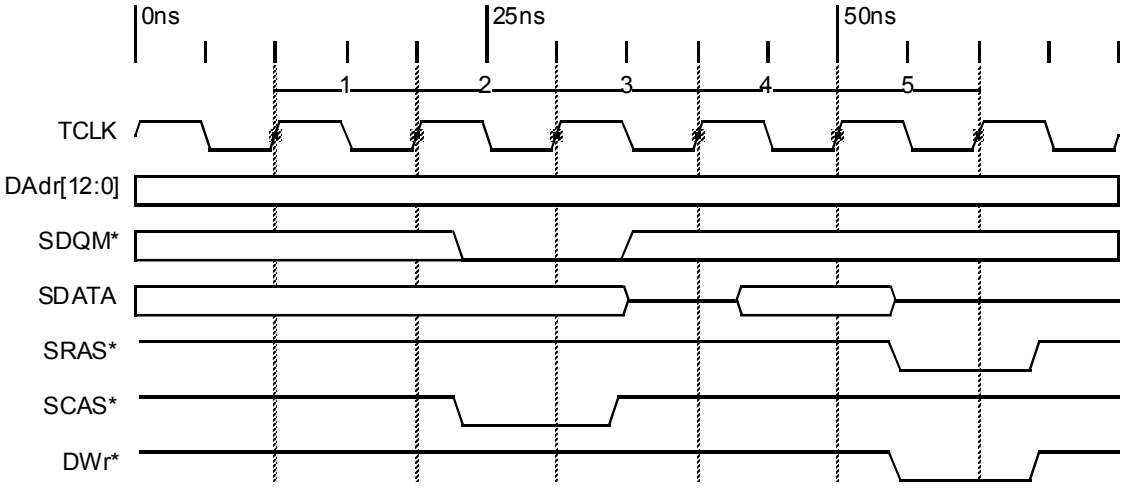
Figure 16: Single Read Access to Non-open Page



- Cycle 2: Row activating.
- Cycle 4: Column cycle (command).
- Cycle 6: Fetch first data – can be 2 or 3 cycles (depends on the CL parameters).
- Cycle 7: Precharge cycle (close the bank).

Figure 17 shows a single read access to an open page.

Figure 17: Single Read Access to Open Page



- Cycle 2: Column cycle (command).
- Cycle 4: Fetch first data – can be 2 or 3 cycles (depends on the CL parameters).

When open pages are enabled, a bank row is kept open until one of the following events occurs:

- An access occurs to the same bank but to a different row address. In this case, the DRAM controller pre-charges, to close the page, and opens a new one (the new row address).
- The access is smaller than the DRAM burst length. The DRAM controller needs to terminate the burst in the middle using early precharge. (See Figure 17.)
- The refresh counter expires. The DRAM controller closes all of the open pages and performs a refresh to all banks.

Section 5. PCI Interface Functional Overview

The GT-64260A supports two 64-bit PCI interfaces, compliant with the PCI specification, Rev. 2.2. Each PCI interface can be individually configured to a 32- or 64-bit configuration.

The GT-64260A PCI0 interface supports ComactPCI Hot Swap. For more information on the CompactPCI Hot-Swap feature, see the GT-64260A datasheet's "Hot Swap" section.

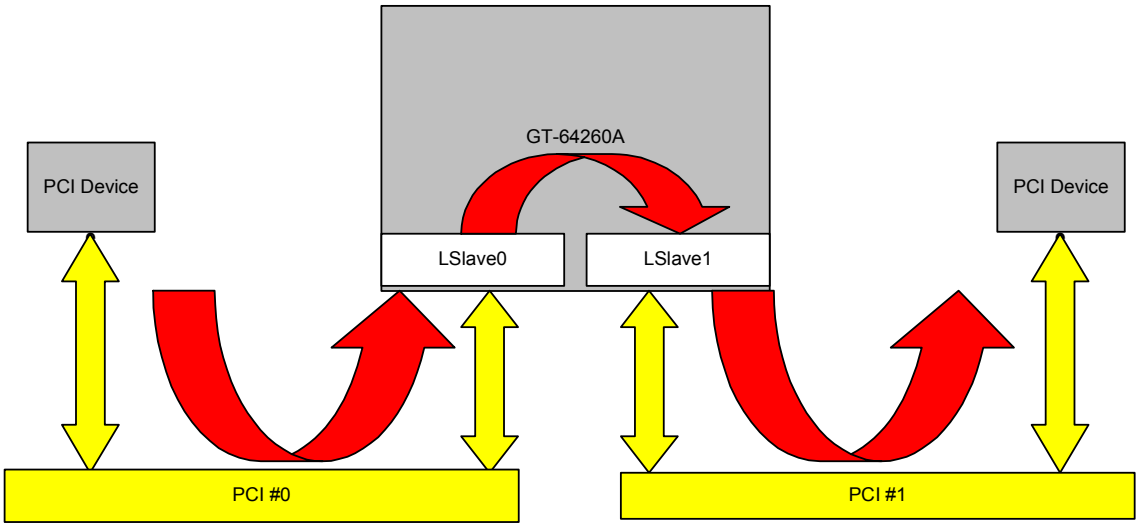
Notes

- When the PCI interfaces are in the reset state, all output pins are in tri-state condition and all open drain pins are floated.
- If not using the GT-64260A in a hot-swap board, the REQ64* pin must be connected to 64EN*.
- For pin and connection information, see the GT-64260A datasheet's "Hot Swap" section.

5.1 P2P Capability

The GT-64260A supports memory Dual Address Cycle (DAC) and Single Address Cycle (SAC), IO, and configuration transactions from one PCI interface to the other PCI interface. The figure below describes a typical P2P system configuration.

Figure 18: Typical P2P System Configuration



5.1.1 Memory and I/O P2P Transactions

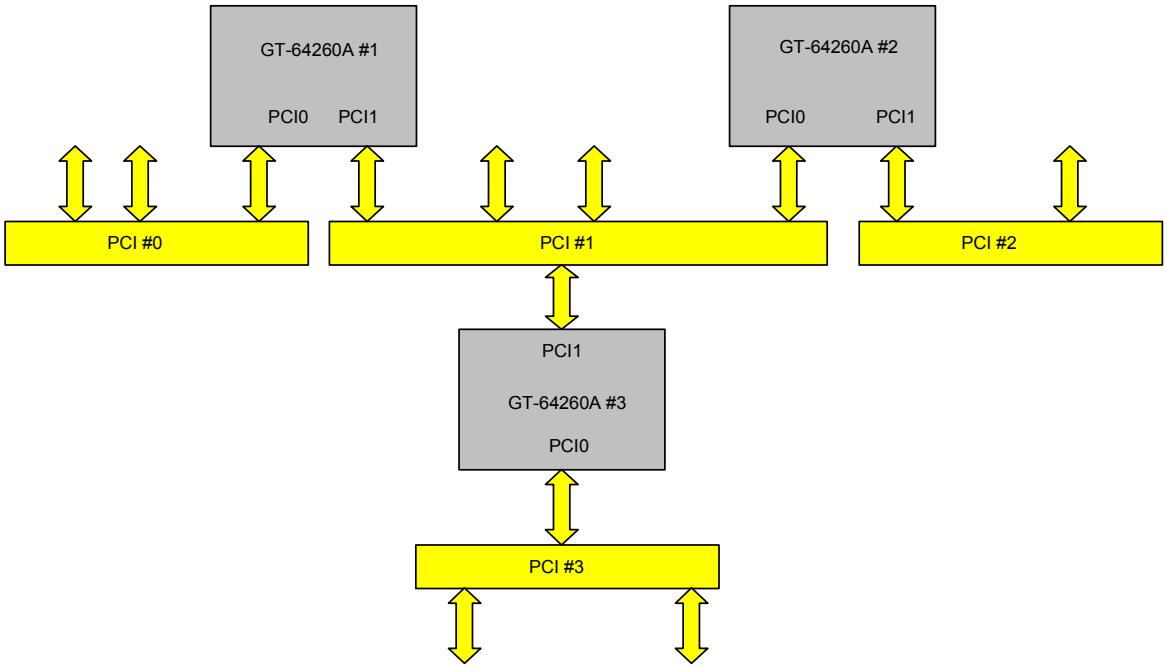
To support access between the two PCI interfaces, the GT-64260A contains two 32-bit Memory BARs and one 32-bit I/O BAR. It also supports two additional Memory BARs for 64-bit addressing (DAC). A PCI address hit in one of the P2P BARs results in transferring the transaction to the other PCI interface memory space.

5.1.2 Configuring P2P Transactions

Each PCI interface responds to a type 1 configuration transaction according to the settings of the PCI P2P Configuration register's `2ndBusL` bits [7:0] and `2ndBusH` bits [15:8], at offsets 0x1d14 and 0x1d94. These fields specify the range of the buses that reside on the PCI interface.

Figure 19 shows an example of a system configuration.

Figure 19: I/O P2P Transaction Example



For this example, the system configuration of the PCI P2P Configuration register must be initialized as follows.

Table 11: PCI P2P Configuration Register Initialization Example

	GT-64260A #1		GT-64260A #2		GT-64260A #3	
	PCI0	PCI1	PCI0	PCI1	PCI0	PCI1
2nd BusL	1	0	2	0	0	3
2nd BusH	3	0	2	3	2	3

Table 11: PCI P2P Configuration Register Initialization Example

	GT-64260A #1		GT-64260A #2		GT-64260A #3	
	PCI0	PCI1	PCI0	PCI1	PCI0	PCI1
BusNum	0	1	1	2	3	1

- 2ndBusL - Secondary PCI Interface Bus Range Lower Boundary
- 2ndBusH - Secondary PCI Interface Bus Range Upper Boundary
- BusNum - The PCI bus number to which the PCI interface is connected.

5.2 PCI Arbitration

The GT-64260A supports either external arbiter or internal arbiter configurations through the PCI Arbiter Control register's EN bit [31], at offsets 0x1d00 and 0x1d80. If the bit is set to '1', the internal PCI bus arbiter is enabled. Each PCI interface can be configured separately to a different PCI arbiter configuration.

5.2.1 Internal PCI Arbiter

When the internal arbiter is enabled the GT-64260A PCI arbiter REQ*/GNT* pins are multiplexed on the MPP pins. Each internal PCI arbiter (PCI0 and PCI1) supports up to six external PCI devices and the GT-64260A PCI device (seven PCI devices per PCI interface).

Figure 12 shows the internal PCI arbiter multiplexing.

Note

The multiplexing for PCI0 and PCI1 interfaces are the same.

Table 12: Internal PCI Arbiter in Multiplexing

PCI Pins	Optional Multiplexing on MPP Pins			
REQ0*	MPP1	MPP17		
GNT0*	MPP0	MPP16		
REQ1*	MPP3	MPP19		
GNT1*	MPP2	MPP18		
REQ2*	MPP5	MPP21		
GNT2*	MPP4	MPP20		
REQ3*	MPP7	MPP23	MPP15	MPP31
GNT3*	MPP6	MPP22	MPP14	MPP30
REQ4*	MPP9	MPP25	MPP13	MPP29
GNT4*	MPP8	MPP24	MPP12	MPP28

Table 12: Internal PCI Arbiter in Multiplexing (Continued)

PCI Pins	Optional Multiplexing on MPP Pins			
REQ5*	MPP11	MPP27		
GNT5*	MPP10	MPP26		

Since the internal PCI arbiter is disabled by default (the MPP pins function as general-purpose inputs), changing the configuration can only be done by the CPU or through serial ROM initialization. The CPU or serial initialization must configure the MPP interface to their appropriate functionality through the MPP Control registers, at offsets 0xf000, 0xf004, 0xf008, and 0xf00c, and then enable in the Arbiter Control register's `EN` bit [31], at offsets 0x1d00 and 0x1d80.

When the internal PCI arbiter is enabled, the REQ* and GNT* pins to the GT-64260A PCI master unit are internally connected. Pull down the GNT* pin and leave the REQ* pin unconnected in the GT-64260A PCI interfaces (PCI0 and PCI1).

For more information on the internal PCI arbiter, see the GT-64260A datasheet's "Internal PCI Arbiter" section.

Notes

- Since the MPP default configuration is general-purpose input, pull ups must be set on all the GNT* and REQ* pins on the MPP interface.
- If more than one master is requesting the bus and the PCI Arbiter Control register's `BDEn` bit [1] and `BV` bits [6:3] are set to '1', at offsets 0x1d00 and 0x1d80, a lock situation may occur. A broken value implies that if a frame was not asserted one cycle after GNT* was asserted the arbiter can grant the bus to another master. In the configuration cycle, the GT-64260A uses address stepping (drives the address and command for one cycle before asserting frame). This means that the configuration cycle might never go to the bus.

5.3 Delayed Read

The GT-64260A uses delay reads for all read transactions. The only way to avoid delayed read is in the following settings:

- For memory read: Read line and read multiple transactions configure the corresponding PCI region to all aggressive prefetch bits disabled, delayed read disabled, and prefetch enabled. The PCI Access Control Base (Low) register, at offsets 0x1e00 and 0x1e80:
  - `RdMulPrefetch` and `RdLinePrefetch` bits [18:17] set to '00'.
  - `DReadEn` and `PrefetchEn` bits [13:12] set to '00'.
- For configuration read: Disable the slave sync barrier in the PCI Command register's `SBDis` bit [13], at offsets 0x1c00 and 0x1c80.
- All IO reads are treated as delayed read.

5.4 32-bit PCI System

The GT-64260A can be configured to work in a 32-bit PCI bus mode. Each PCI interface (PCI0 and PCI1) can be individually configured to 32- or 64-bit modes.

If the REQ64* pin is sampled high at PRst* de-assertion, the PCI interface is configured to 32-bit mode. Since the GT-64260A drives the dedicated 64-bit mode pins (e.g., AD[63:32], CBE[7:4], REQ64*, ACK64* and PAR64*) in 32-bit mode, these bits must not be connected to any other device that might be driving them (e.g., two GT-64260A connected as 64-bit PCI but configured in a 32-bit mode).

5.5 Cache Coherency

The GT-64260A supports cache coherency between PCI and SDRAM. Each PCI transaction to a cache coherent region generates a snoop transaction on the CPU bus.

Note

A read transaction from the WT cache region does not generate a snoop transaction on the CPU bus.

There are up to four configurational address ranges in which cache coherency is maintained. The PCI Snoop Base and Top Address registers define the cache coherent address windows. See 3.4 "Cache Coherency" on page 18.

5.6 Message Signaled Interrupt (MSI)

The optional Message Signaled Interrupt (MSI) feature enables a device to request service by writing a system specified message to a system specified address. Devices that do not support interrupt pins may implement messages to increase performance without adding additional pins.

The MSI is defined in the PCI specification. For more information, see the PCI specification, Section 6.8 "Message Signaled Interrupts" in the PCI specification and the GT-64260A datasheet's "Message Signaled Interrupts" section.

Note

In the following initialization procedure, the first offset number is for PCI_0, and the second is for PCI_1.

The MSI initialization procedure is as follows:

1. Set the PCI Mode register's `MSI` bit [10] to '1', at offsets 0xD00 and 0xD80.
2. Set the PCI MSI Message Control register's `MSIEn` bit [16] to '1', at offsets 0x50 and 0xd0.
3. If DAC (64 bit addressing mode) is needed, the PCI MSI Message Control register's `Addr64` bit [23] must be set to '1'. Also, the PCI Command register's `MDACEn` bit [17] must be set to '1', at offsets 0xc00 and 0xc80.
4. Set the message address in the PCI MSI Message Address registers, at offsets 0x54 and 0xd4. If DAC is needed, set the PCI MSI Message Upper Address registers, at offsets 0x58 and 0xd8. The GT-64260A initiates a write transaction to this address as soon as an interrupt is pending.
5. Set the control data in the PCI MSI Data Control register (at offset 0x5c and 0xdc). This register specifies the data for the write transaction above.
6. Unmask the interrupt, so that the MSI signals in the PCI Interrupt Mask (Low) or PCI Interrupt Mask (high) registers, at offsets 0xc24, 0xca4, 0xc64, and 0xce4.
7. Unmask the interrupt, so that the MSI must signal in the interrupting unit. For more information on the interrupt controller, see Section 11. "Interrupt Controller Functional Overview" on page 82.

Section 6. Device Interface Functional Overview

The device controller supports up to five banks of devices. Each bank's supported memory space can be separately programmed in 1 MB granularity up to 512 MB of address space, resulting in a total device space of 2.5 GB. For pinout information of the device interface, see the GT-64260A datasheet's "Pin Information" section.

6.1 Device Connection

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

To connect more than five devices, use an external logic. The external logic can use the CS*[3:0], BootCS*, and the address pins to generate additional Device_CS* pins. For example, use CS*0 to select two devices. In this case, the address can be used to determine which device is selected:

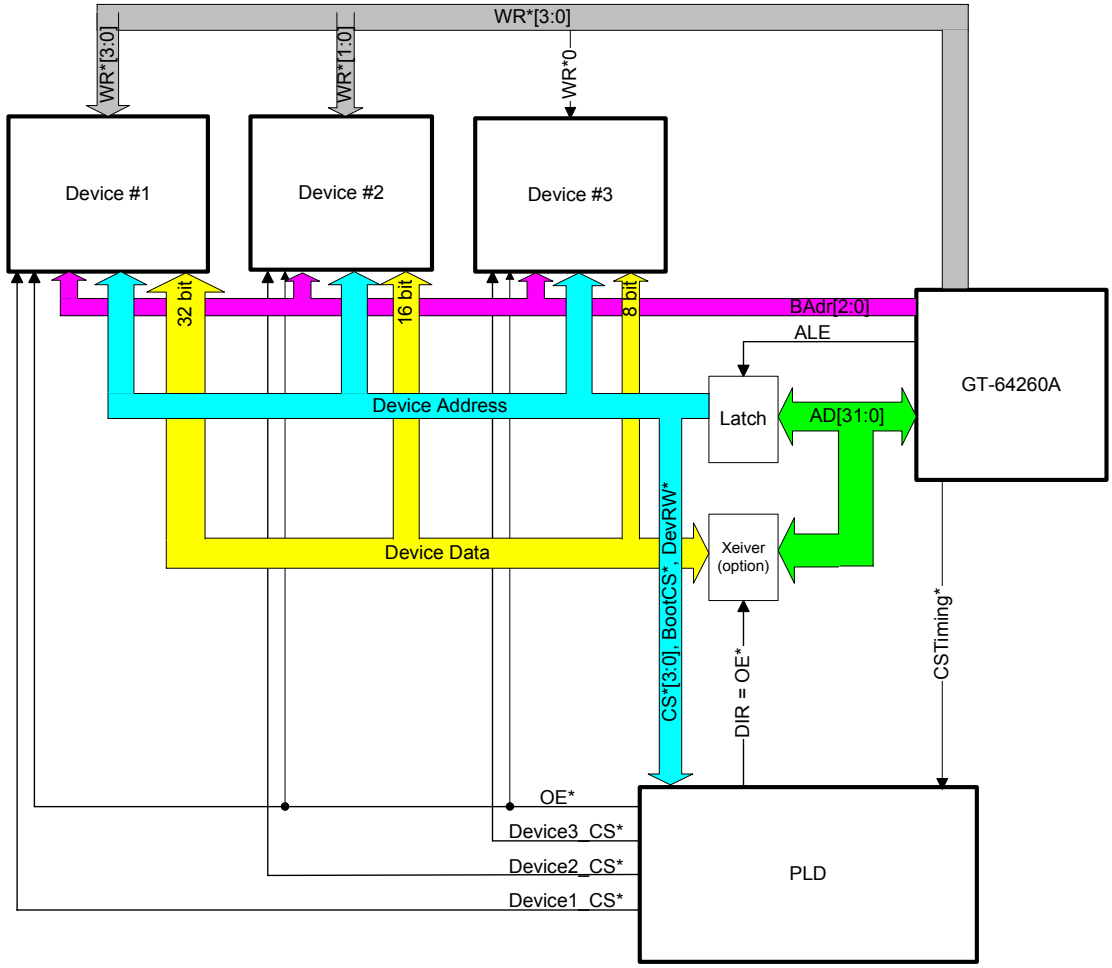
- Device0_CS* = Address[30] 'OR' (CSTiming* 'OR' CS*0)
- Device1_CS* = 'NOT'(Address[30]) 'OR' (CSTiming* 'OR' CS*0)

Figure 20 shows an example of how three devices are connected through the GT-64260A device interface connection.

Note

Since CSTiming* is tri-stated at reset assertion and CS pins qualification may be incorrect, connect a pull-up resistor on the CSTiming* pin.

Figure 20: Three Device Connection Example



Notes

- Since the AD bus is used to configure the GT-64260A at reset, the latch and/or transceivers must NOT implement "bus hold".
- The CS* pins must be qualified with CSTiming* to generate the specific device chip select (e.g., Device_CS* = [CSTiming* 'OR' CS*0]). Also, the DevRW* pin must be qualified with CSTiming* to generate a read or write cycle indication. This pin must be used as an output enable pin and not replace the WR*[3:0] pins. The CSTiming* pin is active for the entire device access time specified in the device timing parameters register.
- Since it is in High-Z for two cycles after reset de-assertion, pull up the CSTiming* pin. This may cause an erroneous qualification of the Device_CS* pins. Alternatively, it is possible to mask the CSTiming* with a delayed SysRst* pin for two cycles.

The GT-64260A device controller supports 8-, 16-, or 32-bit wide devices. The device width is specified in the Device Bank Parameters register's bits `DevWidth` [21:20], at offsets 0x45c, 0x460, 0x464, 0x468, and 0x46c. The boot device width is sampled at reset on `AD` bits[15:14] pins and sets the corresponding bits in the Boot Device Parameters register, at offset 0x46c. For more information on the reset settings, see Section 20, "Reset", on page 144.

6.2 8-bit Device

The GT-64260A device controller supports 8-bit wide devices connected on the AD[7:0] bus. The device controller support up to 8 byte bursts to/from 8-bit devices. The burst address is supported by a dedicated three bit BAdr [2:0] bus that must be connected directly to the device address bus.

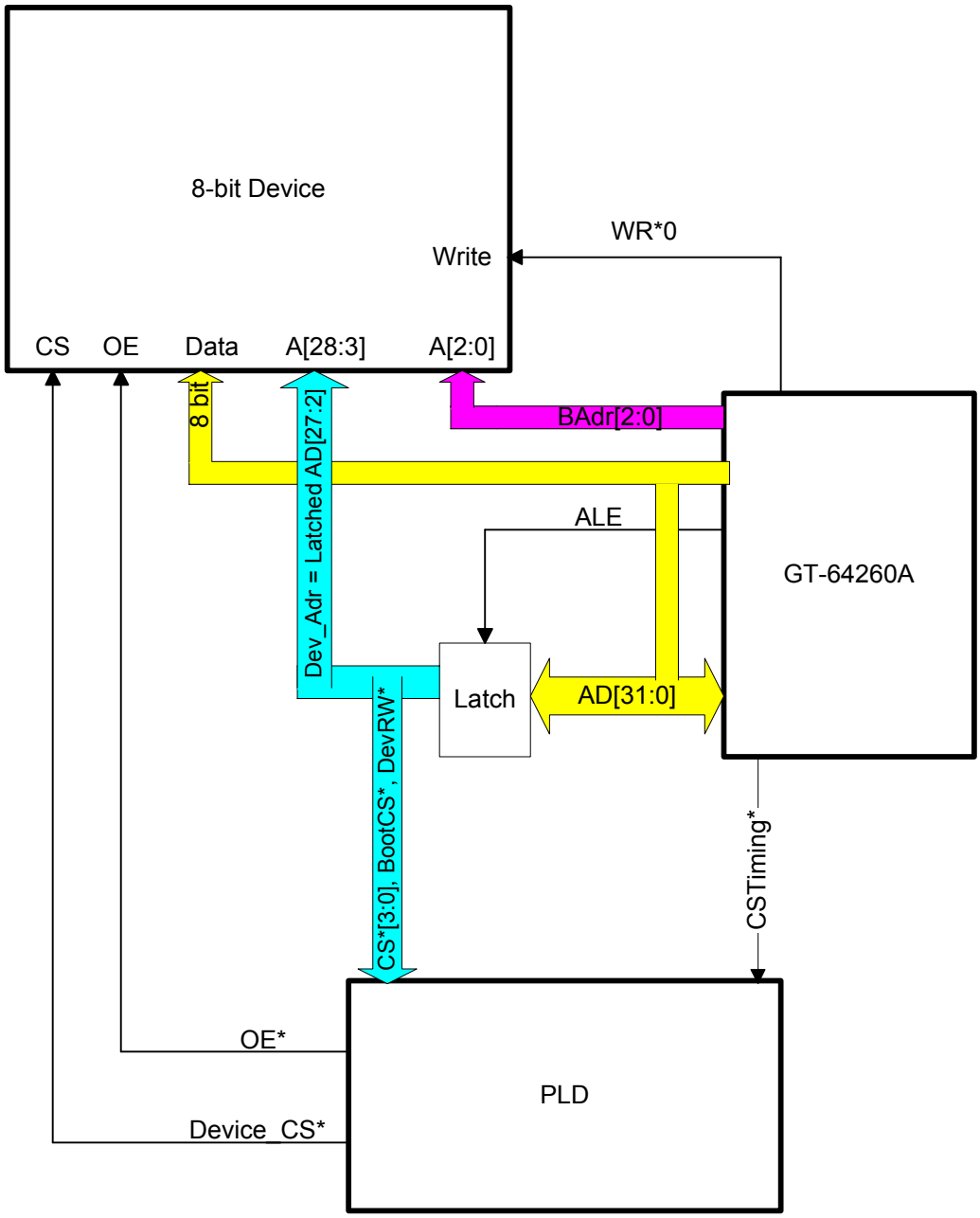
Figure 21 shows the connection of an 8-bit wide device to the GT-64260A. For more information on the 8-bit device connection, see the GT-64260A datasheet's "Interfacing With 8/16/32-Bit Devices" section.

Note

The device controller does not support bursts longer than 8B. Any attempt to support longer bursts causes the GT-64260A to assert an interrupt. For more information on the interrupt assertion, see Section 11, "Interrupt Controller Functional Overview", on page 82. The following restrictions are also implied.

- The CPU must configure the memory space of an 8-bit device as uncacheable space.
- Since the IDMA engines do not support a data transfer limit smaller than 8B, the only DTL allowed from the IDMA engine to an 8-bit device is 8B.
- When using MPC745x CPU, the boot device must be 32-bit wide. For more information, see Section 3.5.1, "MPC745x Burst to Boot Address", on page 21.
- It is only possible to read 8-bit devices from a PCI non-prefetchable region.

Figure 21: 8-bit Device Connection Example



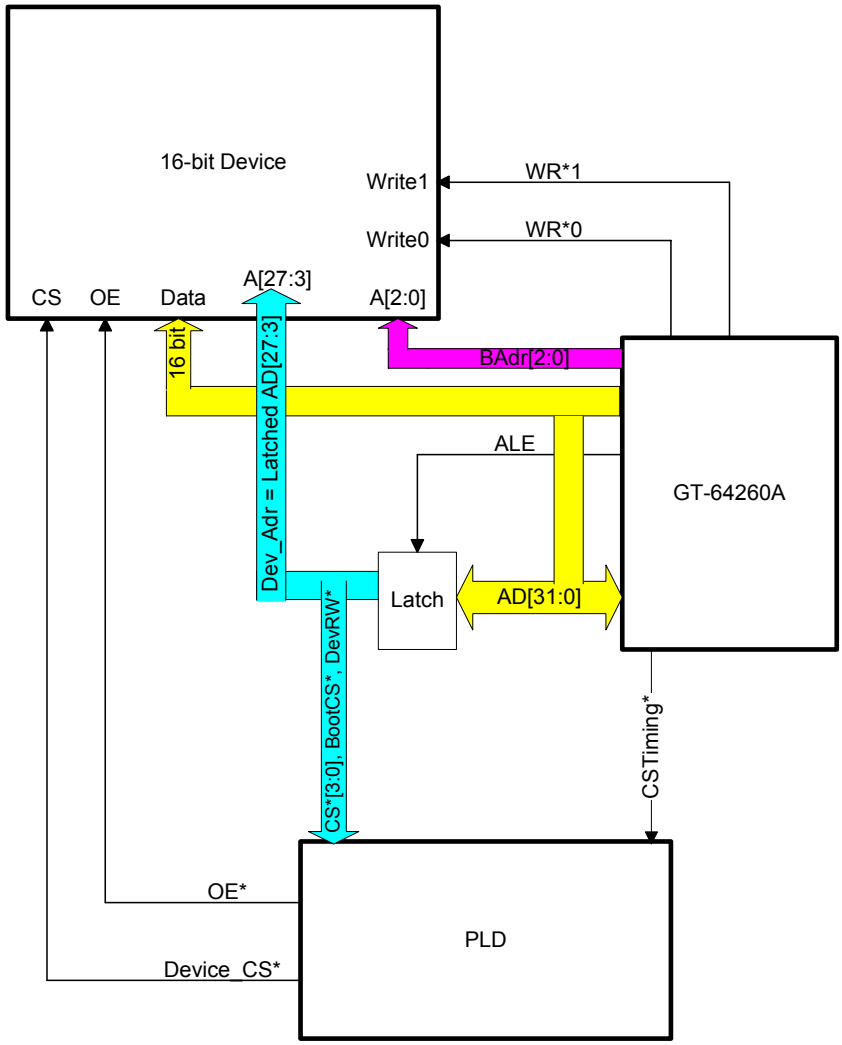
6.3 16-bit Device

The GT-64260A device controller supports 16-bit wide devices connected on the AD[15:0] bus. The device controller support up to 8 byte bursts to/from 16-bit devices. The burst address is supported by a dedicated three bit BAdr[2:0] bus that must be connected directly to the device address bus.

When connecting 2x8-bit devices in parallel to achieve a 16-bit data width, the address pins to both 8-bit devices must be connected the same as for a 16-bit device. Both devices must be connected to the same CS* and OE* pins. Connect the Wr0* pin to one device and connect the other device to the AD[7:0] bus and to the Wr1* pin.

Figure 22 shows the connection of a 16-bit wide device to the GT-64260A. for more information, on the 16-bit device connection, see the GT-64260A datasheet's "Interfacing With 8/16/32-bit Devices" section.

Figure 22: 16-bit Device Connection Example



Note

The device controller does not support bursts longer than 8 bytes for a 16-bit device. Any attempt to support longer bursts causes the GT-64260A to assert an interrupt. For more information on the interrupt assertion, see Section 11, "Interrupt Controller Functional Overview", on page 82. The following restrictions are also implied.

- The CPU must configure the memory space of 16-bit device as uncacheable space.
- Since the IDMA engines do not support a data transfer limit smaller than 8B, the only DTL allowed from the IDMA engine to a 16-bit device is 8B.
- When using MPC745x CPU, the boot device must be 32-bit wide. for more information, see Section 3.5.1, "MPC745x Burst to Boot Address", on page 21.
- It is only possible to read 16-bit devices from a PCI non-prefetchable region.

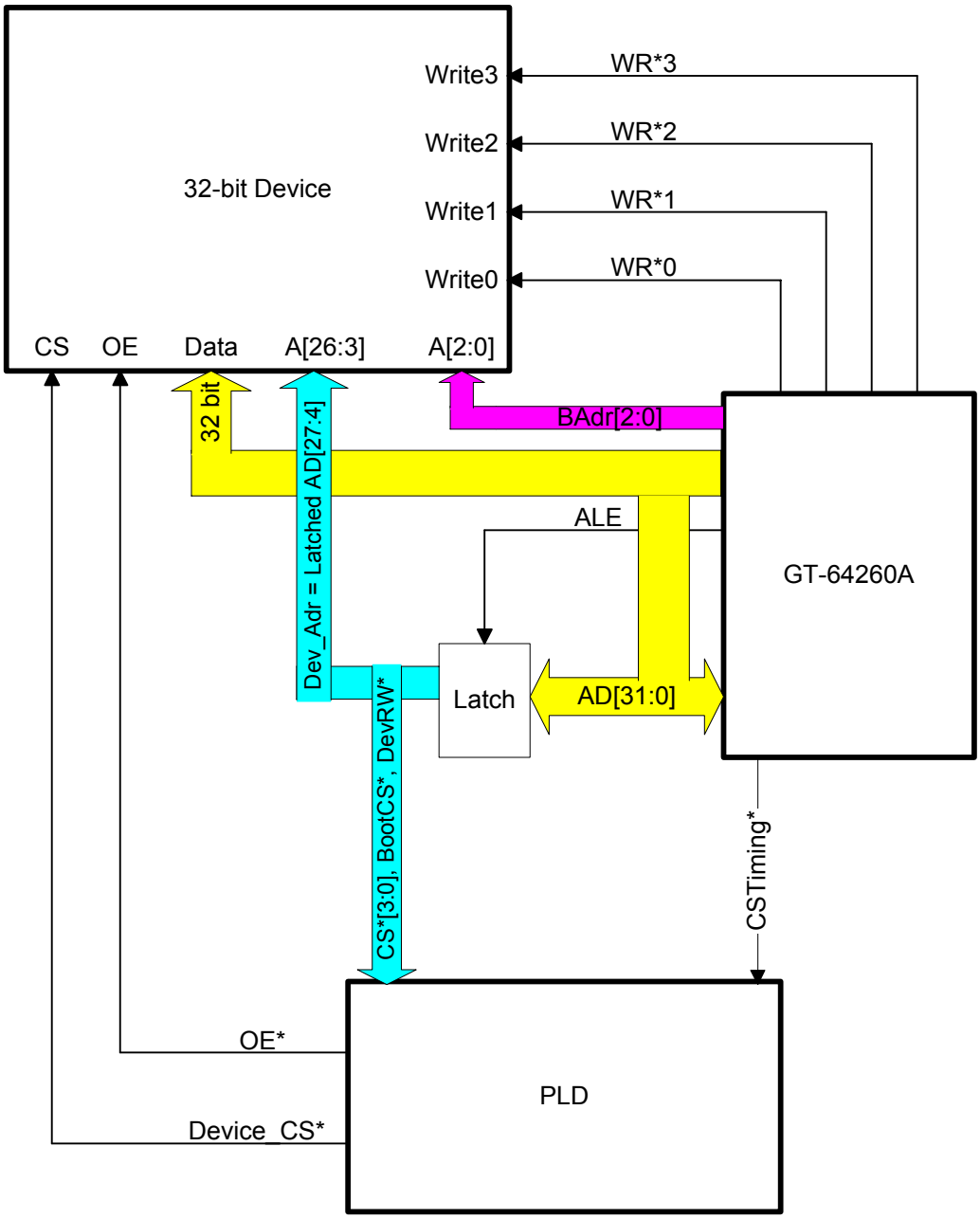
6.4 32-bit Device

The GT-64260A device controller supports 32-bit wide devices connected on the AD[15:0] bus. The device controller support up to 32 byte bursts to/from 32-bit devices. The burst address is supported by a dedicated three bit BAdr[2:0] bus, that must be connected directly to the device address bus.

When connecting 2x16-bit devices in parallel to achieve a 32-bit data width, the address pins to both 16-bit devices must be connected the same as for a 32-bit device. Both devices must be connected to the same CS* and OE* pins. Connect Wr[1:0] and AD[15:0] to one 16-bit device and connect Wr[3:2] and AD[31:16] to the other 16-bit device. Use a similar connection for 4x8-bit configuration.

Figure 23 shows the connection of 32-bit wide device to the GT-64260A. For more information on the 32-bit device connection, see the GT-64260A datasheet's "Interfacing With 16/16/32-bit Devices" section.

Figure 23: 32-bit Device Connection Example



6.5 Signals Timing

Note

For more information on device bus signals, see the GT-64260A datasheet's "Pin Information" section.

The GT-64260A provides programmable access timing for various device implementations. The programmable parameters for device access have a 1 clock cycle granularity and are set per CS* (bank). All programmable access timings are set in the corresponding Device Bank Parameters register (Offsets: 0x45c, 0x460, 0x464, 0x468). There are separate parameters for write and read accesses, the read access parameters include TurnOff, Acc2First, Acc2Next, and BAdrSkew. The write parameters include ALE2Wr, WrLow and WrHigh.

Figure 24 shows a device burst read example with the following device timing parameters.

- Acc2First = 5 cycles.
- Acc2Next = 3 cycles.
- BadrSkew = 0 cycles.

Figure 24: Device Burst Read Example

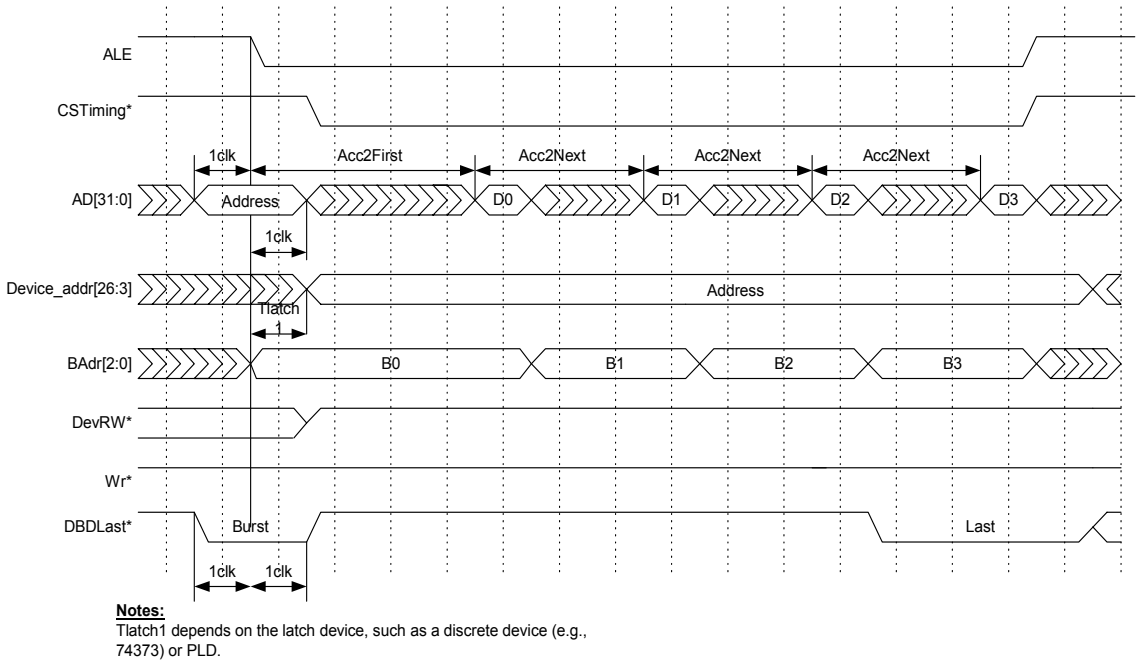
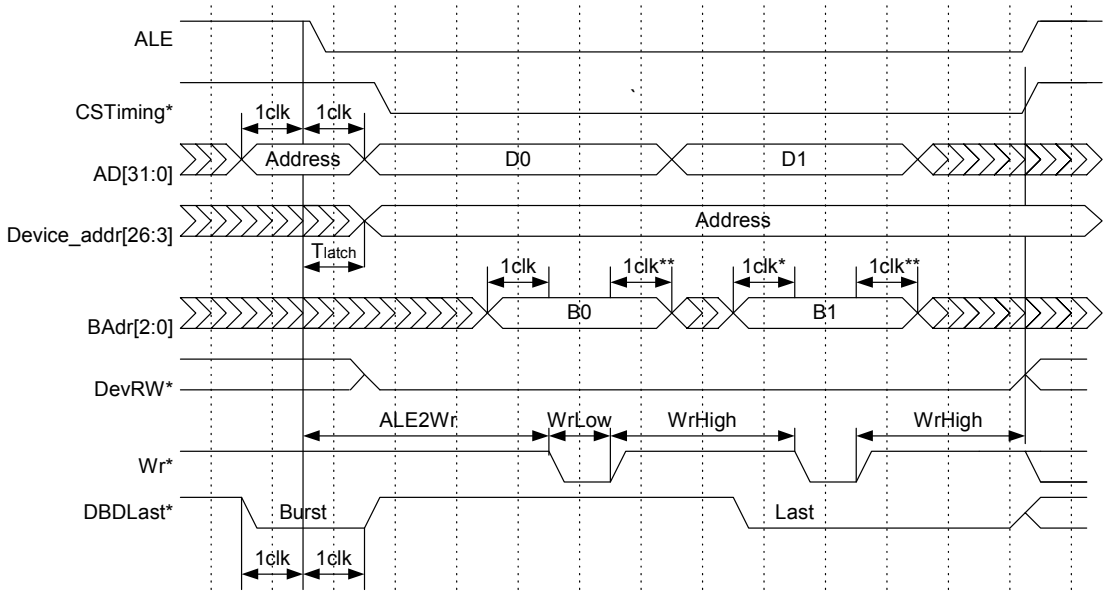


Figure 25 shows a device burst write example with the following device timing parameters:

- ALE2Wr = 5 cycles.
- WrLow = 1 cycles.
- WrHigh = 3 cycles.

Figure 25: Device Burst Write Example



Notes:
1. * One clock setup time only if the WrHigh is larger than 1
2. ** One clock hold time only if the WrHigh is larger than 0
3. Latch depends on the latch device. Such as a discrete device (e.g., 74373) or PLD.

6.6 Ready Support

To support devices with long or undetermined access times, the GT-64260A implements the Ready* pin on the device interface. The Ready* pin extends the access timing parameters for Acc2First, Acc2Next, and WrLow.

In addition, the Ready* pin supports non-sampled and sampled modes. The sampled mode adds an additional clock cycle to the access and is used to reduce the setup time of the Ready* pin.

Note

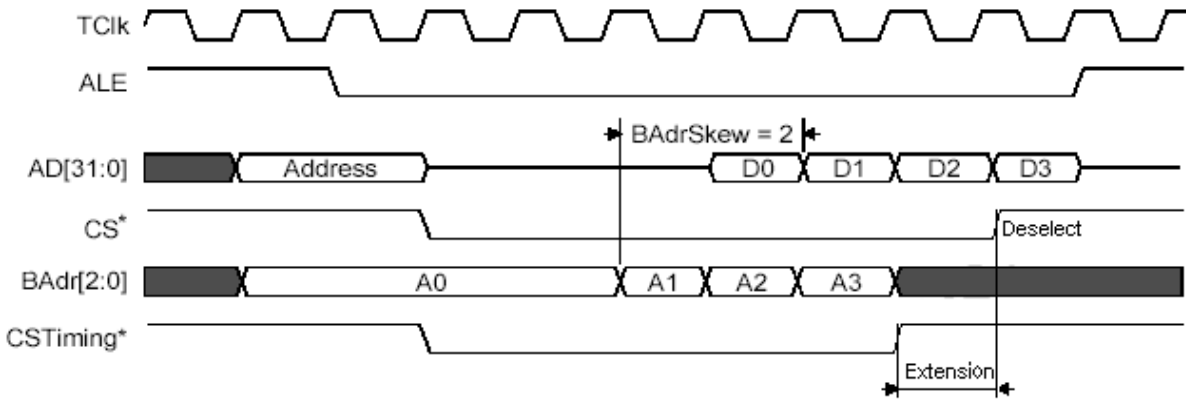
In the GT-64260A datasheet, the Ready* pin AC timing refers to the sampled mode. There is no AC timing for the non-sampled mode. For more information, see the GT-64260A datasheet's "AC Timing" section.

6.7 Syncburst SRAM

Syncburst SRAM devices can be connected to the GT-64260A device interface. It is recommended to use pipelined syncburst SRAM devices since they are designed for 133 MHz or higher frequencies.

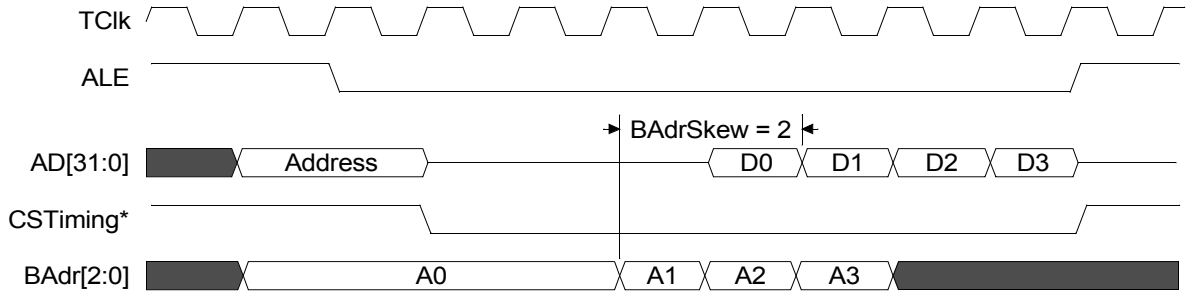
When using a Single Cycle Deselect (SCD) pipelined syncburst SRAM, an external logic must be used to extend the CS and Read (DevRW*) pins for one cycle on read transactions because the SRAM outputs are disabled within one clock cycle after deselect. (See Figure 26.)

Figure 26: SCD Pipeline Sync Burst SRAM Read Example



When using Dual Cycle Deselect (DCD) pipelined syncburst SRAM, no external logic is needed because the SRAM outputs are disabled within two clock cycles after deselect. (See Figure 27.)

Figure 27: DCD Pipeline Sync Burst SRAM Read Example



Section 7. Communication Interface Functional Overview

The GT-64260A integrates the following into its communication unit:

- Three Ethernet controllers
- Two MPSC controllers
- An I^2C interface
- SDMAs
- Baud Rate Generators (BRG)

7.1 Ethernet Controllers

There are three 10/100 Mbps full duplex Ethernet ports in the GT-64260A. Each port is fully compliant with the IEEE 802.3 and 802.3u standards and integrates MAC function and a dual speed MII interface. The Ethernet ports can be configured to MII or RMII (three Ethernet ports configuration is available only with RMII). The port's speed (10 or 100Mb/s) as well as the duplex mode (half or full duplex) is Auto-Negotiated through the PHY and does not require user intervention.

The Ethernet interface implements the following pins: E0[14:0], E1[14:0], MDC, and MDIO. The E0 and E1 pins are multiplexed to support MII and RMII configurations.

Notes

- For Ethernet pin multiplexing information, MII, and RMII mode pin descriptions, see the GT-64260A datasheet.

Since the serial ports have different configurations and part of them "wake-up" after reset as tri-state pins, the following pull downs are required. However, it is recommended to pull down all unused pins for electrical protection from software errors.

- If not using port E0 and to minimize the number of pull downs, configure the port to MII and pull down pins E0[1] and E0[14:6].
- When E0 is configured as two RMII ports, pins E0[7] and E0[14] are not connected and must be pulled down. If only one of the two RMIIs are used (Ethernet controller 0), pins E0[13] and E0[11:10] must not be connected and must be pulled down.
- If not using port E1 and to minimize the number of pull downs, configure the port to MII and pull down pins E1[1] and E1[14:6].
  - When E1 is configured as an RMII port, pins E1[1], E1[7:6], E1[12:10], and E1[14] are not connected and must be pulled down.

7.2 MPSC Controllers

The two, integrated MPSCs on the GT-64260A support UART, HDLC, and transparent protocols. The MPSCs are implemented in hardware. This implementation allows for superior performance versus micro coded implementations.

The MPSC interface implements the following pins: S0[6:0] and S1[6:0]. For information on the S0 and S1 pins configuration, see the GT-64260A datasheet's "Pin Information" section.

Since the serial ports have different configurations and part of them wake-up after rest as tri-state pins, the following pull downs are required. However, it is recommended to pull down all unused pins for electrical protection from software errors.

- If not using port S0, configure the port to MPSC and pull down pins S0[1] and S0[6:3].
- If not using port S1, configure the port to MPSC and pull down pins S1[1] and S1[6:3].

7.3 Cache Coherency

The GT-64260A does not support cache coherency from the communication unit by the hardware. To achieve cache coherency between the SDRAM and the CPU from the communication interfaces, it is necessary to implement cache coherency with the software.

The GT-64260A defines snoop regions that identify the regions in memory. Accessing these regions initiates a snoop cycle on the CPU bus. The GT-64260A supports IDMA and PCI cache coherency. For more information, see 3.4 "Cache Coherency" on page 18.

There are two options for software cache coherency support.

7.3.1 Software Cache Coherency Support Option 1

The CPU defines the part of the memory as cached and the rest as uncached. In addition, cache coherent regions will be defined in the GT-64260A for IDMA transactions. In this solution, the communication driver maintains a separate set of buffers (and descriptors) that it will use for IDMA I/O of the communication interface. These buffers are located in the cached memory and configured in the GT-64260A as cache coherent.

The Communication interface uses uncached and non-cache coherent regions in GT-64260A. Before handing off the packet to other software entities, the communication driver copies the received frame into another buffer in a cacheable and cache coherent memory region by using the IDMA engine. A similar copy takes place on a transmitted frame.

The solution's advantage is its easy implementation involving minimal risk. Using the IDMA engines in GT-64260A to implement the copy can mitigate the performance penalty of the copy.

Note

For better resource management, use a dedicated IDMA engine for each port.

7.3.2 Software Cache Coherency Support Option 2

The CPU defines the memory as cached but it is the software's responsibility to flush the data back to memory in a way that the communication interfaces always receive the updated data. This means the buffers/descriptors for the communication interfaces are cacheable as far as the CPU is concerned. However, the buffers/descriptors are

outside the snoop regions defined in the GT-64260A. As a result, the communication unit does not drive any snoop cycles when it is accessing to memory.

This solution increases software complexity and since the buffer pool is shared among many devices in the software, this creates additional complications.

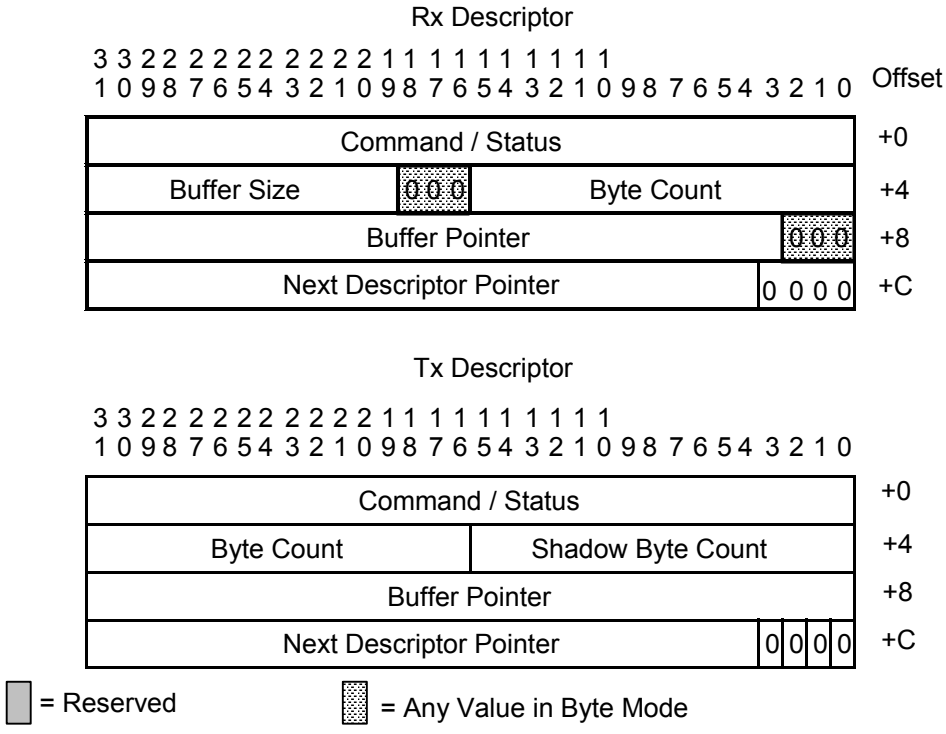
7.4 MPSC and Ethernet SW Implications

The GT-64260A communication interfaces include three Ethernet (two for MII) ports and two MPSCs ports. Each port implements two SDMA engines, one for transmit and one for receive. After appropriate initialization, the SDMA is designed to minimize CPU intervention. For code examples, see Appendix E. "Communication Example Code" on page 158.

7.4.1 Descriptors Management

The descriptors are the engines of the SDMA that enable packet transfers through the GT-64260A communication ports. The figure below describes the Rx and Tx descriptors.

Figure 28: SDMA Descriptor Format



Although similar, the MPSC SDMA and the Ethernet SDMA differ in the SDMA descriptors' command status field (different for each protocol) and the Tx descriptors' shadow Byte Count field. The descriptors are 16 byte in size and do not require enormous processing resources.

It's recommended to allocate the descriptors in a non-cached region. If the descriptors are allocated in a cacheable region, memory coherency must be guaranteed.

For efficient use, descriptors are managed in a chain list.

Tx Descriptors

It is recommended to use the Tx descriptors in a cyclical chained list to reduce software overhead.

For better performance, use two pointers for each Tx descriptors queue. One of the pointers always points to the next available descriptor. In this way, packets are prepared for transmission without waiting for the previous packets to its transmission.

The second pointer is used to release the Tx descriptors after transmit ends.

For improved descriptor manipulation, extra fields can be added to the Tx descriptors. The additional field must be an integer factor of the descriptor size. For example,

```
typedef struct SdmaTxDesc
{
    unsigned int    bytecnt:16;
    unsigned int    shadow:16;
    TX_COMMAND              cmd_sts;
    unsigned int    next_desc_ptr;
    unsigned int    buf_ptr;
    /* extra field - where the source buffer taken from */
    unsigned int              pointerToRxQueue;
    /* extra field - help to release used buffers */
    unsigned int              shadowOwner:1; /* if 1 -> belong to the GT */
} TX_DESC;
```

ShadowOwner is an extra 1-bit field which determines if this descriptor is released. This is because a Tx descriptor is owned by the CPU has no indication whether it's free or should be released, by the CPU after sending a packet.

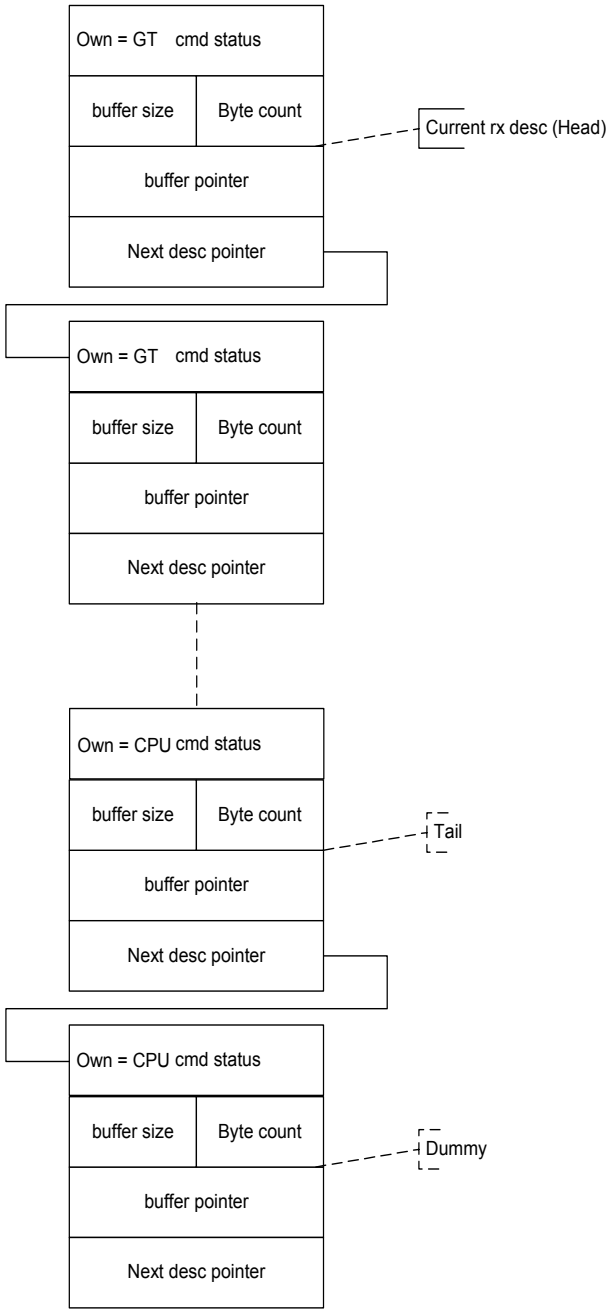
Before sending a packet, the shadowOwner field is marked as SHADOW_OWNER_BY_CPU (defined as '0'). This is the indictor to free this descriptor. After the descriptor is freed, this field is marked as SHADOW_OWNER_BY_GT (defined as '1').

Rx Descriptors

It is recommended to use the Rx descriptors in a long chain terminated with two descriptors owned by the CPU. The last descriptor is a dummy used to avoid a NULL pointer. The descriptor before the last one is the tail.

The software holds two pointers. The head is the first descriptor in the list and the tail is the last descriptor before the dummy descriptor. (See Figure 29.)

Figure 29: Rx Descriptor Chain

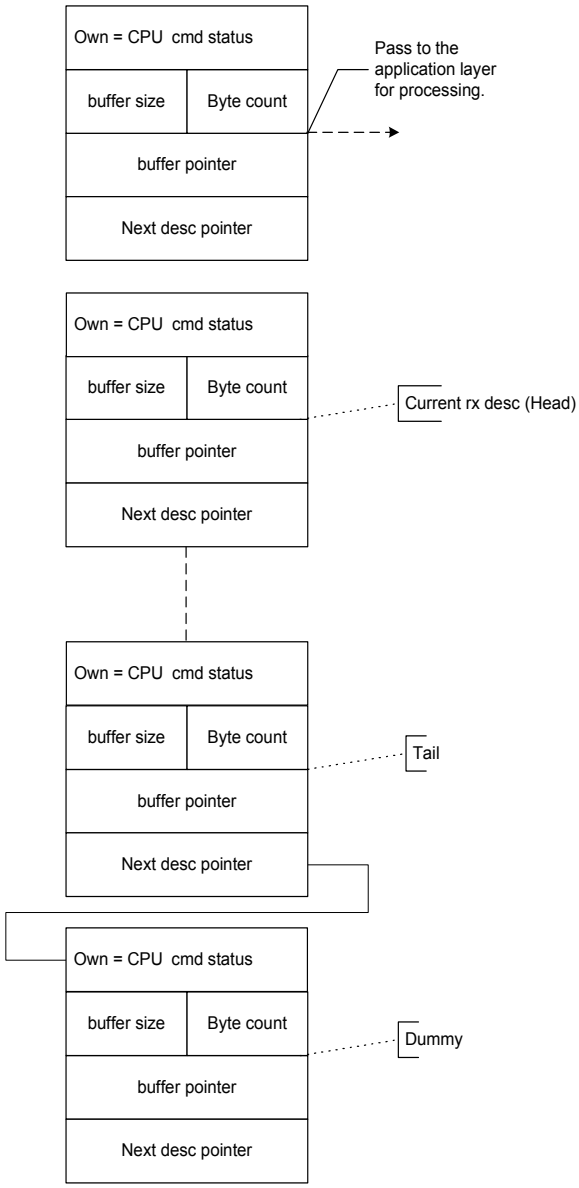


For each receive packet:

- Disconnect the descriptors that formed this packet by moving the head pointer to the next descriptor in the chain.
- Pass the descriptors to the application layer.

Figure 30 shows an example of disconnecting the descriptor.

Figure 30: Disconnecting the Descriptor Chain



When the descriptor is moved to the application layer, it can:

- Process the packet.
- Send the packet to a different queue.
- Ignore the packet and release the descriptor immediately.

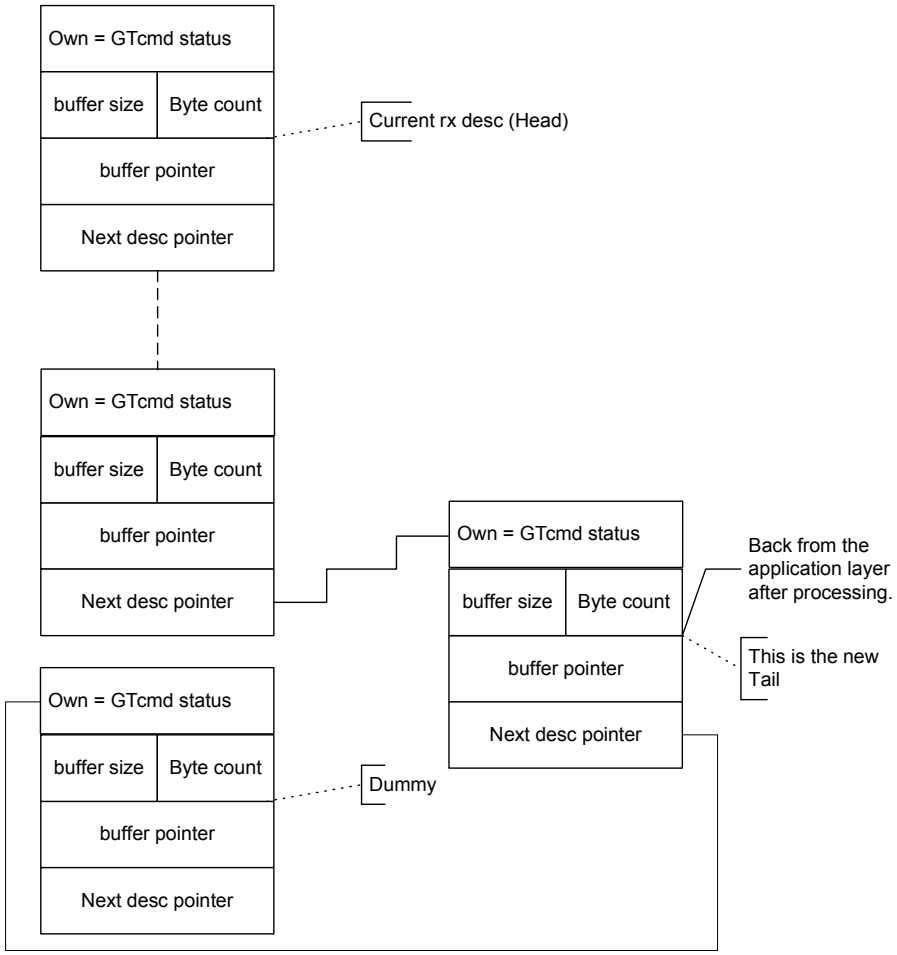
Heavy packet processing might delay the receive queue handling. This method enables parallel descriptor handling and receive new packets.

To release the descriptor:

- Connect the descriptor between the tail and the dummy.
- Set the tail to be owned by the GT device.
- Mark the released descriptor as the new tail.

Figure 31 shows an example of releasing the descriptor.

Figure 31: Releasing the Descriptor Chain



7.4.2 *DemoInidev* Description and Performance Aspects

To demonstrate the communication unit ability, the Board Support Package (BSP) implements an additional application layer, located in the demoApi.c file that functions as a repeater.

This driver uses the communication unit driver's APIs (see the BSP User Manual) to operate the GT-64260A Communication Unit.

This application layer is responsible for the initialization and configuration of the various communication units. It is also responsible for handling all events generated by the communication units (e.g., interrupts).

The driver also includes a switching table that determines where to switch the packets received in each communication port (repeater functionality).

The following switching table is placed in the DemoApi.c file in the BSP:

```
UINT32 switchingTable[MAX_NUMBER_OF_RX_QUEUES] = {
/* Rx queues                        target queue switching */
/* MPSC0 */                         ETHER0_LOW_SWITCH,
/* MPSC1 */                         ETHER1_LOW_SWITCH,
/* ETHER0_PRIO0 */                  ETHER1_LOW_SWITCH,
/* ETHER1_PRIO0 */                  ETHER2_LOW_SWITCH,
/* ETHER2_PRIO0 */                  ETHER0_LOW_SWITCH,
/* ETHER0_PRIO1 */                  MPSC0_SWITCH,
/* ETHER1_PRIO1 */                  MPSC1_SWITCH,
/* ETHER2_PRIO1 */                  MPSC0_SWITCH,
/* ETHER0_PRIO2 */                  MPSC0_SWITCH,
/* ETHER1_PRIO2 */                  MPSC1_SWITCH,
/* ETHER2_PRIO2 */                  MPSC0_SWITCH,
/* ETHER0_PRIO3 */                  MPSC0_SWITCH,
/* ETHER1_PRIO3 */                  MPSC1_SWITCH,
/* ETHER2_PRIO3 */                  MPSC0_SWITCH
};
```

Note

Each Ethernet port has four Rx priority queues and two Tx priority queues, totalling 14 Rx queues and eight Tx queues.

The application reacts with receive packets and transmit packets events. For each received packet, the application uses the switching table to determine the transmit queue where the packet must be transmitted (without any other packet processing). For example, all the packets received in Ethernet port 0, lowest priority, are sent to Ethernet port 1, low priority Tx queue. (See the switching table above, third entry).

7.4.2.1 *DemoInidev* Initialization

- Connects an ISR to the GT-64260A Interrupt Controller so the application can respond to the communication interface events.
- Initializes the internal registers as needed for proper use.
- Spawns the ISR task, which handles the communication unit interrupts.

- Initializes the MPSC ports according to the user's definition. See *AN-72: Operating the MPSC as a UART* and E.3 "MPSC API" on page 160.
- Initializes the Ethernet ports. See *AN-66: Initializing Ethernet Ports on the GT-642xx and GT-961xx Devices* and E.1 "Ethernet Initialization" on page 158.

All communication interrupt events are marked in the ISR linked list.

7.4.2.2 Driver ISR

When the GT-64260A Communication Unit generates an interrupt, the interrupt controller calls a single ISR that handles the variety interrupt causes. The ISR processes the interrupt in three stages:

1. It reads the Interrupt Cause register, acknowledges the interrupt events, and masks the interrupt events bits.
2. Adds a new node to the ISR linked-list according to the interrupt events. The counters in the ISR linked-list nodes are set to MINIMUM_INTERRUPT_COUNTER (the default value is 30). The counter represents the number of iterations of the application call back function. See Section 7.4.2.3 "Driver's Task" .
3. Using a semaphore, the driver's task is invoked.

These three stages are necessary to allow high performance packets processing during stress. The main processing takes place in the driver's task, thus an effort has been made to reduce ISR overhead. For this reason, the node counter is set to a value of MINIMUM_INTERRUPT_COUNTER (value 30).

7.4.2.3 Driver's Task

The communication unit provides a single task named isrTask, located in the isrTask.c file. This task is responsible for handling the communication unit events. This task has a linked-list data structure, in which each node represents a communication unit interrupt bit.

The task's purpose is to scan this linked-list data structure and execute the user routine specified for each node (call back functions). The call back functions treat each event accordingly. When the linked-list is not empty, the task reads the first node, decreases a counter, and calls the user routine call back function. If the counter reaches zero, the node is deleted from the linked-list and the corresponding interrupt bit is unmasked. This task is active as long as the linked-list is not empty. If the linked-list is empty, it exits and waits for a pin from the ISR.

This demo is very useful for communication unit performance measurements.

7.4.2.4 Tips for Using *DemoInidev*

- Compile the VxWorks with O2 optimization flag.
- Compile bootrom.hex to support CAS and RAS of 2. More details can be found in the BSP user manual in the BSP features chapter.
- Check the required communication flow and change the switching table to support this flow.
- Configure the SmartBit ports to Auto-Negotiation. This guarantees that the Ethernet ports are set to full duplex and 100 MB/s.
- Perform the demo press 'demoInidev' after the vxWorks boot, and send traffic using SmartBits.
- Performance is measured by dividing the Rx rate by the Tx rate in the SmartBits counters.

7.4.2.5 Relevant Documentation

- AN-66: Initializing Ethernet Ports on the GT-642xx and GT-961xx Devices
- AN-72: Operating the MPSC as a UART
- BRG settings tool

7.4.3 UART Over MPSC

The BSP is supplied with a UART over MPSC capability. To use this capability, refer to the BSP user manual. The code is implemented in: uartMpsc.c and uartMpsc.h.

The workaround for the MPSC Errata is implemented in the BSP API.

7.4.4 Improving Performance

- Set CAS and RAS to '2'.
- Set the SDRAM Configuration register's `VInterEn` bit [14] to '1', at offset: 0x448.
- Set CPU Configuration register's `Pipeline` bit [13] to '1', at offset 0x000.
- Use the internal arbitration control.

7.5 I²C Interface

The GT-64260A has full I²C support. It can act as master generating read/write requests and as a slave responding to read/write requests. It fully supports multiple I²C master's environments.

The I²C port consist of two open drain pins that must be connected to positive supply voltage via a current-source or Pull-up resistor.

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

Note

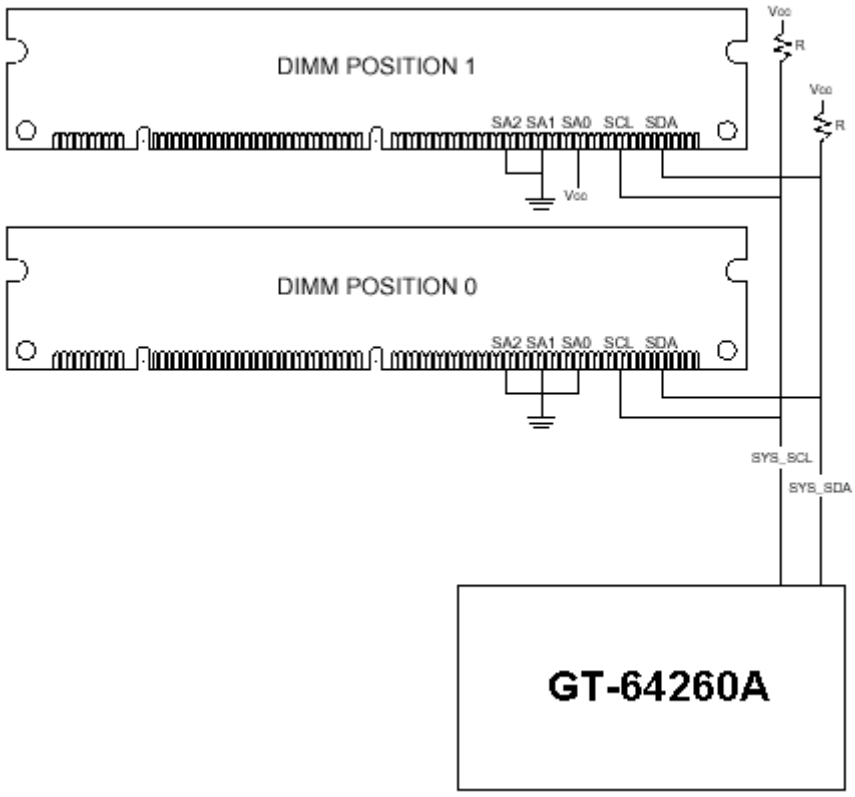
The register that is used to generate the I²C interrupt is the I²C control register, at offset 0xC008. See `IFlg` bit [3] and `IntEn` [7] in that register. The cause for an I²C interrupt is shown in the I²C Status register, at offset C00C.

7.5.1 Memory Serial Presence-Detect

The I²C can be used for SDRAM module serial presence detect. Serial presence-detect (SPD) has been standardized through JEDEC so that implementation will be consistent from one module standard to the next.

The SPD data stored in the EEPROM placed on the SDRAM module is defined as bytes of information in a specific order. For more information on the SPD data structure, see Micron's TN-04-42 "Memory Module Serial Presence-Detect" on the Micron web site (www.micron.com). Figure 32 shows the GT-64260A I²C interface connection to the SDRAM DIMMS, enabling the use of the SPD.

Figure 32: GT-64260A I²C Interface Connection to SDRAM DIMMS



7.5.2 I²C Interface Initialization

To work with the I²C interface, it must be initialized first. The interface initialization sequence is as follows:

1. Reset the I²C logic by writing to the I²C Soft Reset Register, at offset 0xC01C.
2. Set the I²C frequency in the I²C Baud Rate Register, at offset 0xC00C.
3. When working in interrupt mode, set the I²C Control register's `IntEn` bit [7], at offset 0xc008, and unmask the I²C interrupt bit in the corresponding mask register. See Section 11. "Interrupt Controller Functional Overview" on page 82. At this stage, when the CPU gets an interrupt form the I²C unit, it reads the I²C Status Register, at offset 0xC00C.

To only work with the I²C interface as a Master, disable the general call and keep the slave address 0x0 (which is the address of the general call). The I²C never returns Ack as a slave, but only as a master.

Note

When he I²C Control register's `I2CEn` bit [6] is set to '0', at offset 0xc008, the GT-64260A does not monitor SDA and SCL. This effectively disables both the master and slave. Actually, the master can still transmit (send address with R/W command) but it can't get the data returned or send an acknowledgement.

7.5.3 Serial ROM Initialization

If serial ROM initialization is enabled (AD[0] pin sampled high on SysRst* de-assertion), the GT-64260A I²C master starts reading initialization data from the serial ROM and writes it to the appropriate registers.The serial ROM initialization enables the user to initialize the GT-64260A internal registers, configuration registers, and other devices connected on the PCI.

The serial ROM initialization must implement access to address 0x1400xxxx, since the I²C interface does not sample the internal base setting at reset. This is not correct for the CPU interface, when the internal base is set to 0xf1000000, the CPU must access the GT-64260A internal registers at 0xf1000000 + offset. It is possible to configure to the internal base register through the serial ROM. In this case, all of the following accesses to the registers must correspond to this value. For example, if the Serial ROM includes the following lines:

```
0x14000068
0x00000f20
```

All the access following this line must be written to 0xF2000000. For example:

```
0xf2000000
0x4281a8ff
```

For more information on the serial ROM initialization, see the GT-64260A datasheet's "Serial ROM Initialization" section and the 3.5.1 "MPC745x Burst to Boot Address" on page 21 for an serial ROM initialization example.

Note

Make sure that all the reset configuration strapping required by the GT-64260A with serial ROM initialization are pulled to the correct values (the same value configured in the serial initialization). For more information, see the GT-64260A datasheet's "Reset Configuration" section and the latest Datasheet Changes and Update document.

7.6 Baud Rate Generator

The GT-64260A implements two Baud Rate Generators (BRG) that select one of the input clocks:

- TClk – GT-64260A internal clock, the same clock as the SDRAM interface.
- BclkIn – multiplexed on the MPP interface.
- SClk[1:0] – MPSC port 0 or port1 serial clock.
- TSClk[1:0] – MPSC port 0 or port1 transmit serial clock.

The BRGs use these source clocks to generate a derived clock, depending on the BRGx Configuration register's `CDV` bits [15:0], at offsets 0xB200 or 0xB208. For more information, see the GT-64260A datasheet's "Baud Rate Generators" Section.

The BRG output clock can be configured to drive the BclkOut pin multiplexed on the MPP interface. This clock is useful as a source clock to an external device. In addition, the BRG output clock can be a source clock for the MPSC ports.

Note

If the BRG output clock is a source clock for the MPSC ports, the BRGs can only use TClk or BclkIn clocks as source clock. For more details, see the errata and restriction document.

Section 8. Multi-Purpose Pin Interface Functional Overview

The GT-64260A has 32 Multi-Purpose Pins (MPP). Each one can be assigned to a different functionality through the MPP Control registers (Offsets: 0xf000, 0xf004, 0xf008, and 0xf00c). The MPP pins can be used as hardware control signals to the various GT-64260A interfaces (UMA control, DMA control, or PCI arbiter signals) or as General Purpose Pins. For more information, see the GT-64260A datasheet's "Pins Multiplexing" section.

Note

Since the MPP interface is configured as input at reset and for hardware protection from software errors, it is recommended to pull all the MPP pins either high or low.

8.1 General Purpose Pin (GPP)

Each MPP can be configured to function as a General Purpose Pin. When an MPP is configured as GPP, it can function as input or output (GPP I/O Control register, Offset: 0xf100) and it can be configured as an active low or high signal (GPP Level Control Register, Offset: 0xf110).

When a certain GPP is configured as output, it is driven to its inactive state by default. In this configuration, the associated bit in the GPP Value register (offset 0xf104) is read/write. Setting the corresponding bit in the GPP Value register to '1' sets the associated GPP output pins (inverted in case of active low pin).

To initialize the GPP outputs:

1. Select the IO control for the corresponding GPP pins in the GPP I/O Control register, at offset 0xf100. If a specific bit is set to '1', the corresponding GPP pin is set as an output.
2. In the GPP Level Control register, at offset 0xf110, select the GPP pins activity (polarity). Setting a specific bit in the GPP Value register, at offset 0xf104, to '1' asserts the corresponding GPP pin. If the pin is an active low, the GPP pin will be driven low. if it is active high, the GPP pin is driven high.

Note

If the GPP output is routed back to another GPP input, they must have the same polarity.

3. Select the GPP functionality to GPP in the MPP Controlx registers (Offsets: 0xf000, 0xf004, 0xf008, 0xf00c, 0xf010).

For example, to set GPP10 as an active low GPP output:

```
/* Initialization */
    SET_REG_BITS(GPP_IO_CONTROL,BIT10);                                  /* Output */
    SET_REG_BITS(GPP_LEVEL_CONTROL,BIT10);                             /* Active Low */
    RESET_REG_BITS(MPP_CONTROL1, BIT8 | BIT9 | BIT10 | BIT11);    /* GPP functionality */
/* Asserting */
    SET_REG_BITS(GPP_VALUE, BIT10); /* Set the GPP bit */
/* Deasserting */
```

```
RESET_REG_BITS(GPP_VALUE, BIT10); /* Clear the GPP bit */
```

Note

If GPP10 is configured to active high, the GPP output and the Asserting and De-asserting code are the same. The only difference is the GPP level setting at the Initialization:

```
RESET_REG_BITS(GPP_LEVEL_CONTROL,BIT10);                    /* Active High */
```

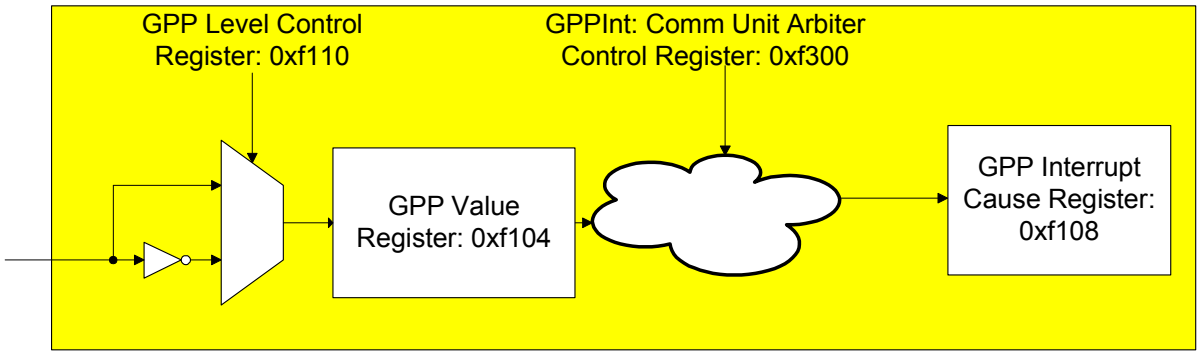
When a certain GPP is configured as input, the associated bit in the GPP Value register, at offset 0xf104, is only read and it represents the value of that GPP pin. In this configuration, the GPP can be used to register external interrupts to each one of the seven interrupt outputs. For more information on the interrupt outputs, see Section 11, "Interrupt Controller Functional Overview", on page 82.

An assertion of the GPP input sets the associated bit in the GPP Interrupt Cause register, at offset 0xF108. If the associated GPP input is not masked in the GPP Interrupt Mask register, at offset 0xF10C, an assertion of the GPP input will set a bit in the main cause register.

The GPP inputs support both edge sensitive and level sensitive interrupts, depending on the setting of Comm Unit Arbiter Control register's `GPPInt` bit [10], at offset 0xF300. If set to '0', it is configured as edge trigger. If set to '1', it is configured to level sensitive.

Figure 33 describes the GPP interface when configured as input.

Figure 33: GPP Configured as Input



Note

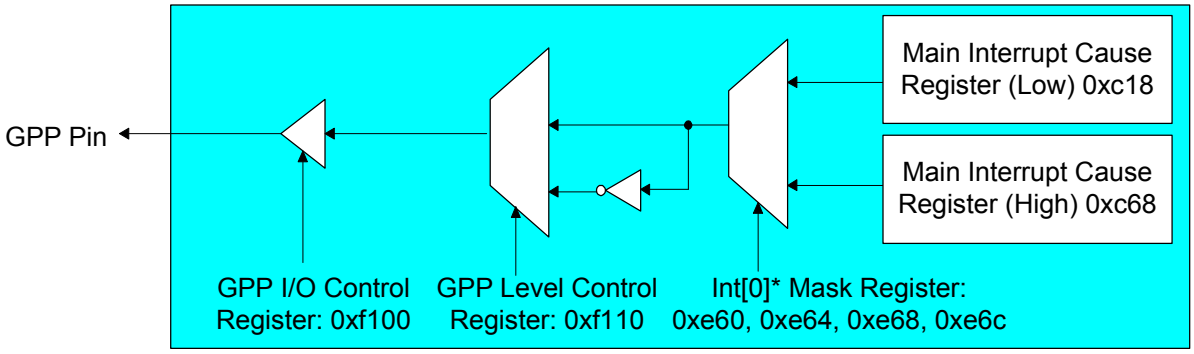
When the GPP is configured to edge trigger, it must be kept active for at least one TClk cycle to guarantee that the interrupt is registered.

8.2 Interrupt Outputs

The MPP interface implements four interrupt pins multiplexed on the MPP interface Int[3:0]*. Each pin is kept active as long as there is at least one non-masked cause bit set in the Interrupt Cause register. Each Int[3:0]* interrupt implements a single mask that can be selected to mask one of the main cause registers. Figure 34 describes

the MPP interrupt outputs. For more information on the interrupt controller, see Section 11, "Interrupt Controller Functional Overview", on page 82.

Figure 34: MPP Interrupt Outputs



8.3 PCI Arbiter

The MPP interface can be used for the PCI internal arbiter signals. For more information on the pin multiplexing, see Section 5.2, "PCI Arbitration", on page 48.

8.4 DMA Request

When working in demand mode (the ChannelX Control register's `DemandMode` bit [11] is set to '0'), the MPP interface can be used as an input to the IDMA unit for an external trigger to activate the channel. Each channel is coupled to a DMAReq* pin. For more information on the IDMA engine, see Section 10, "IDMA Unit Functional Overview", on page 79.

Note

Setting a channel to demand mode without configuring an MPP pin to act as the channels DMAReq* causes the channel to hang.

8.5 DMA acknowledge

When working in demand mode, the MPP interface can be used as an output from the IDMA unit notifying the external device that its request is being served. Each channel is coupled to a DMAAck* pin. For more information on the IDMA engine, see Section 10, "IDMA Unit Functional Overview", on page 79.

8.6 Unified Memory Architecture Control

The GT-64260A can be configured to support Unified Memory Architecture (UMA) at the reset configuration (see Section 20, "Reset", on page 144).

Also, the GT-64260A can be configured to act as a UMA master or slave. UMA systems require two additional signals for arbitration of the SDRAM interface. For more information, see the GT-64260A datasheet's "Unified Memory Architecture Support" section.

8.7 DMA End of Transfer

When the EOT support is enabled (the Channel Control register's `EOTEn` bit [18] is set to '1'), the IDMA engine transfer can be terminated by external hardware via EOT pins. For more information, see the GT-64260A datasheet's "End of Transfer" section.

8.8 Timer Counter Enable

When an external TCen support is enabled (the Timer/Counter Control register's `TCxTrig` bit [2] is set to '1'), the Timer/Counter can be stopped by external hardware via TCEn pins.

The MPP interface can be configured as a TCEn[7:0] input. Each timer/counter has its own Tcen input pin. For more information on the Timer/Counter unit, see the GT-64260A datasheet's "Timer/Counters" section.

Note

If using an external count enable input, it is necessary to configure the appropriate MPP pin prior to counter activation.

8.9 Initialization Active

The MPP interface can be configured as InitAct output. It will be driven High during the serial ROM initialization. For more information, see the GT-64260A datasheet's "MPP Multiplexing" section.

Notes

- When serial initialization is enabled, one of the MPP pins must be configured as InitAct.
- Since the InitAct pin is configured as input at reset, it must be pulled up.

8.10 BRG Clock

The MPP interface can be configured as BRG input/output. One of the MPP pins can be configured to BclkIn, as an input clock, to one of the BRGs and another MPP pin as BClkOut0, as an output clock of BRG0.

Section 9. JTAG Interface Functional Overview

The GT-64260A JTAG interface is compliant with the Boundary-Scan standard known as IEEE 1149.1.

The following JTAG commands are supported by the GT-64260A JTAG TAP controller:

- EXTEST (0000)
- SAMPLE (0001)
- BYPASS (1111)
- HIGHZ (0011)
- IDCODE (0010)
- TEST (0100)

The GT-64260A IDCODE value is 0x12300157, see the BSDL file at Marvell secured web site, http://www.galileot.com/secure/products/discovery/#BSDL MODELS.

If the current instruction is IDCODE in the GT-64260A TAP controller implementation, each exit from the SHIFT-DR state will cause the IDCODE to be fetched to DR (all 32 bits). While the controller is in the SHIFT-DR state, it shifts data from TDI to the register and, after the IDCODE was shifted out from the TDO, it starts shifting out that data.

It is recommended to use the BYPASS instruction to shift the data from TDI to TDO. This requires fewer shifting cycles.

Note

See the IEEE 1149.1 specification for details on JTAG operation.

Section 10. IDMA Unit Functional Overview

The GT-64260A IDMA controller transfers blocks of data independent of the local processor or PCI hosts. Data movement can occur from any interface to any interface.

The GT-64260A implements eight IDMA channels. Each IDMA engine can move data between any source and destination, such as the SDRAM, Device, PCI_0, PCI_1, or CPU bus.

The IDMA controller can be programmed to move up to 16 MB of data per transaction. The burst length of each transfer of IDMA can be set from 8 bytes to 128 bytes. Accesses can be non-aligned both in the source and the destination.

The IDMA controller clock source is the TClk input, the same clock used for the SDRAM interface. For more information about the GT-64260A clocking, see Section 19. "Clocks" on page 143.

The DMA engine implements two independent machines. The read machine performs reads from the source address and pushes the data into the buffer. The write machine pulls data out of the buffer and writes it to the destination address. With this implementation, the buffer can be filled with multiple reads before there is a write. When the read and write accesses are to/from different interfaces (i.e., read from SDRAM and write to PCI), the accesses can be executed in parallel.

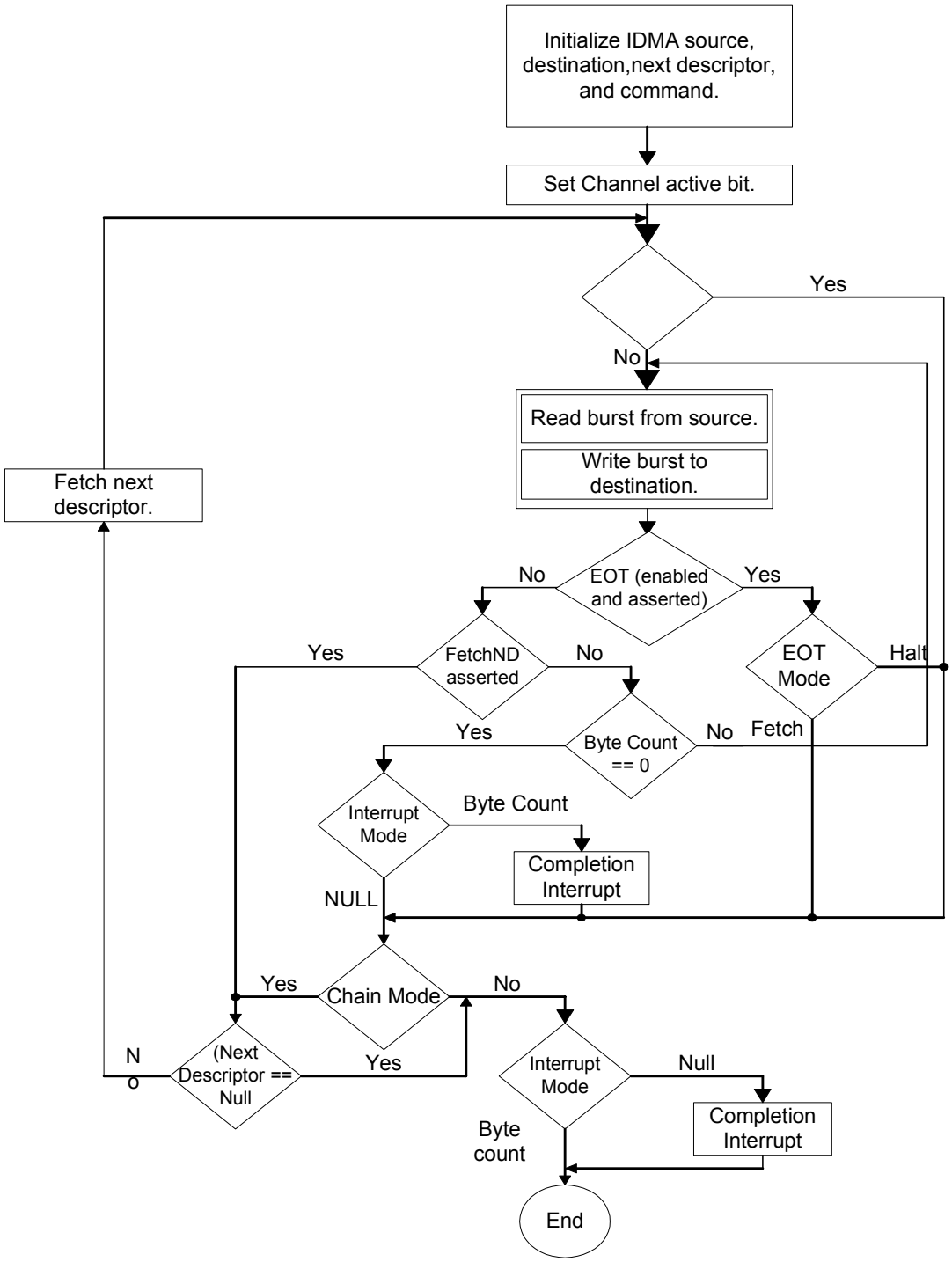
Some additional features of the GT-64260A IDMA unit are:

- Eight DMA channels (7:0).
- All channels accessible by processor core and remote PCI masters.
- Misalign transfer capability.
- Chain mode.
- Direct mode (non-chain mode).
- The CPU, external request or timer/counter, can initiate DMAs transfers.
- Interrupt on completed segment, chain, and error conditions.
- DMA transfer to all interfaces.
- IDMA unit contains two 2 KB buffers.
- Increment or hold of source and destination addresses.
- Source/Destination/Descriptor address override capability.
- Support byte/word-swapping to/from PCI.
- Support cache coherency between SDRAM and CPU.

Figure 35 shows a general flow diagram for the operation of the DMA controller.

For more information, see the GT-64260A datasheet's "IDMA Controller" section.

Figure 35: DMA Controller General Flow



Notes

- The DMA controller functions as a PCI agent relative to the other GT-64260A internal resources.
- The IDMA engines cannot transfer data to or from the GT-64260A internal registers.

10.1 Chain Mode

The GT-64260A IDMA engines can be activated in a chain mode. In this mode, the DMA controller loads descriptors from memory (SDRAM, PCI, etc.) prior to a DMA transfer. The DMA controller begins the transfer according to the descriptor information loaded for the segment.

When the chain is completed, the Current Descriptor Register continues pointing to the chain's last descriptor. The register will not be updated to null, even though the next descriptor pointer of the last chain is null.

When clearing the channel enable bit of an active channel, the channel only stops after completing the burst in progress. If that burst happens to be the last burst in the chain/descriptor (depending on Interrupt mode - control_I[10]), a completion interrupt is asserted anyway, even if the enable bit is off.

For more information, see the GT-64260A datasheet's "Chain Mode" section.

10.2 Cache Coherency

The GT-64260A supports cache coherency between CPU and SDRAM.

Each IDMA transaction to a cache coherent region generates a snoop transaction on the CPU bus. There are up to four configurable address ranges in which cache coherency is maintained. IDMA Snoop Base and Top Address registers define the address windows. See 3.4 "Cache Coherency" on page 18.

Section 11. Interrupt Controller Functional Overview

The GT-64260A can drive up to seven interrupt pins:

- Interrupt* - dedicated interrupt output to CPU.
- PCI_Int*0 - dedicated interrupt open-drain output to PCI0.
- PCI_Int*1 - dedicated interrupt open-drain output to PCI1.
- Int*[3:0] – four additional CPU interrupts multiplexed on MPP pins.

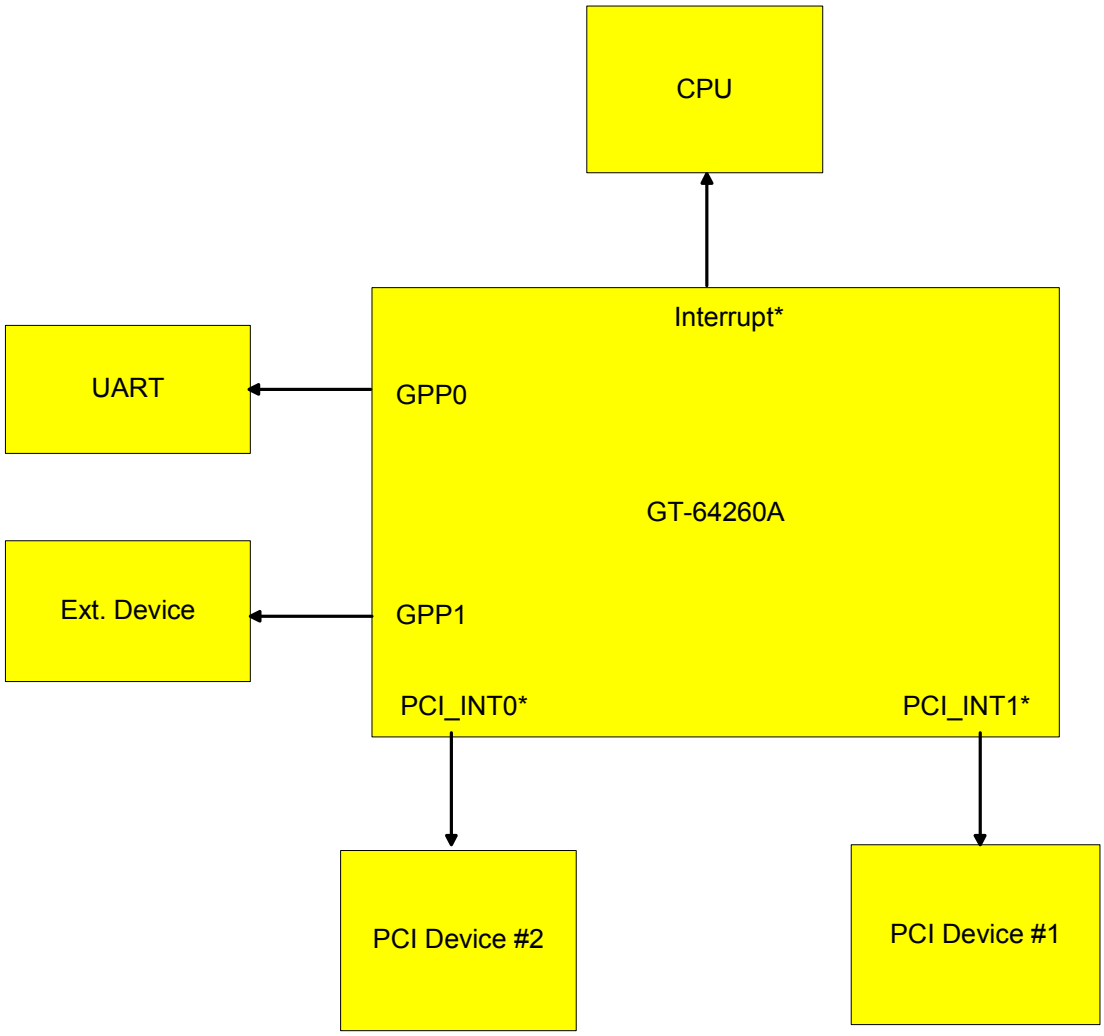
All seven interrupts driven by the GT-64260A are "Level Sensitive Type". The interrupt is kept active as long as there is at least one non-masked cause bit set in the Interrupt Cause register.

If the interrupt source is an external device driving a GPP input (see 8.1 "General Purpose Pin (GPP)" on page 74), the GT-64260A can be configured to receive a level or edge trigger. If the Comm Unit Arbiter Control register's `GPP_Int` bit [10] is set to '0', at offset 0xF300, the external interrupts are treated as edge trigger interrupts. This means an assertion of a GPP input pin results in setting the corresponding bit in the GPP Interrupt Cause register. Only an edge will trigger the interrupt.

If the `GPP_Int` bit is set to '1', the external interrupts are treated as level interrupts. In this mode, an interrupt is always generated as long as at least one of the GPP Value register bits is asserted and the GPP Interrupt Mask register does not mask it.

Figure 36 shows an interrupt routing example.

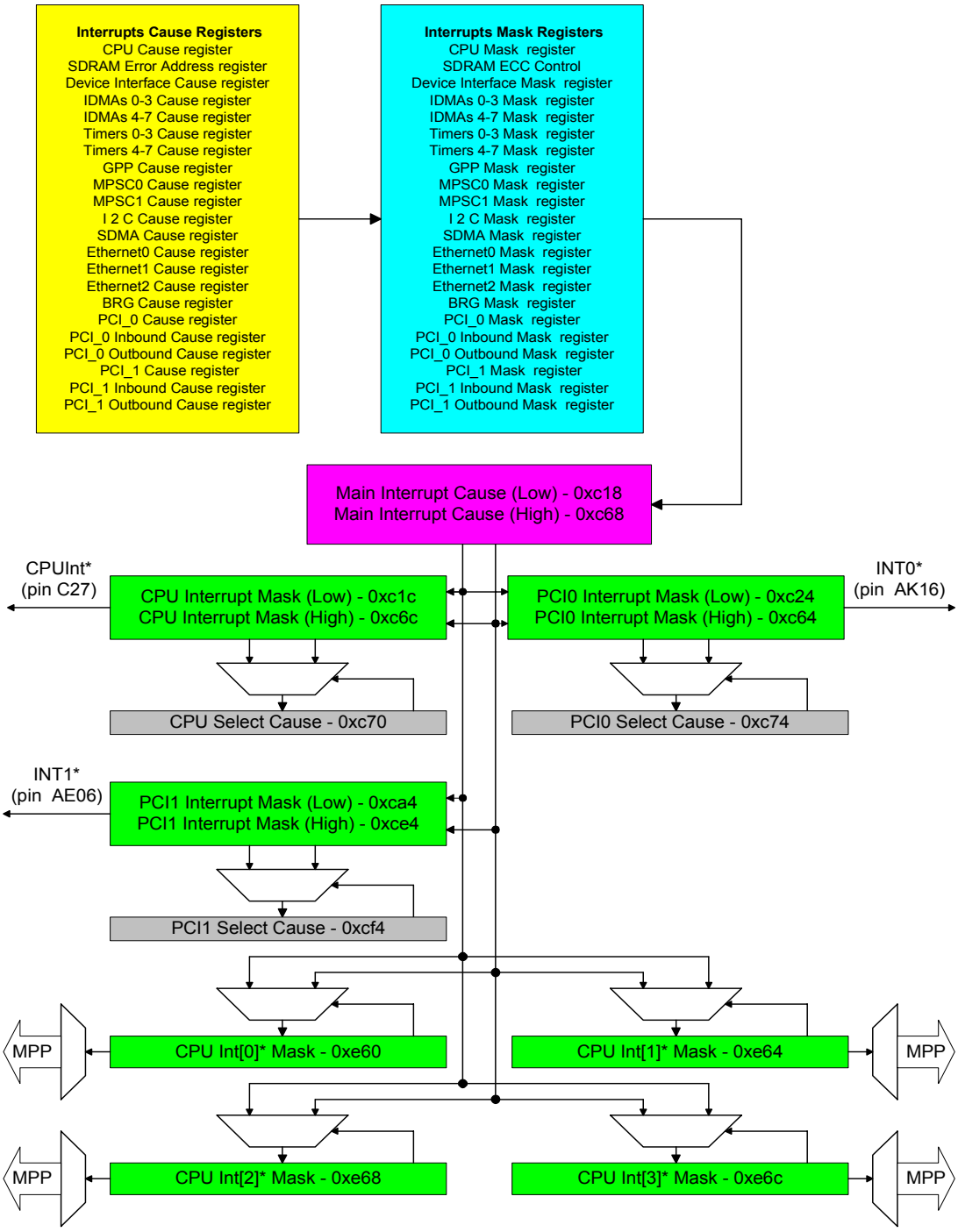
Figure 36: Interrupt Routing Example



The GT-64260A handles interrupts in two stages. It includes a main cause register, summarizing the interrupts generated by each unit, and specific unit cause registers, distinguishing between each specific interrupt event.

Figure 37 shows the GT-64260A interrupt routing architecture.

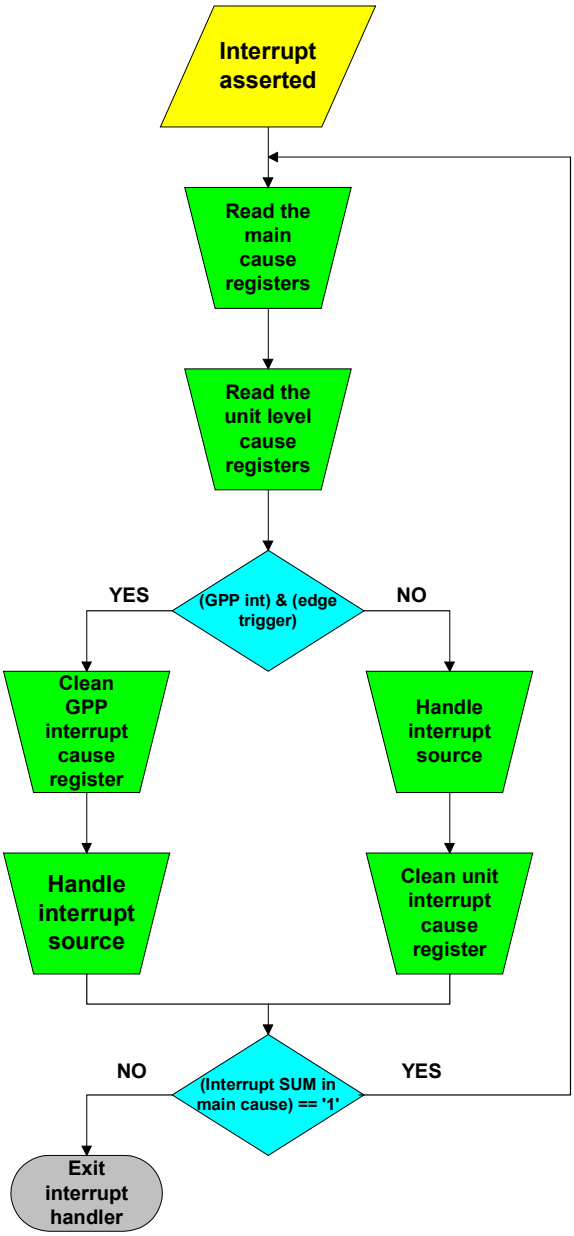
Figure 37: GT-64260A Interrupt Routing Architecture



When an interrupt is asserted, the interrupt handler first reads the main cause registers low and high, at offsets 0xC18 and 0xC68, to distinguish which unit/interface generates the interrupt. To reduce the interrupt overhead,

the interrupt handler reads only the select cause register. The Select Cause register's `Sel` bit [30] indicates which of Low or High Cause registers are currently represented by the Select Cause register. At this stage, the interrupt handler reads the unit level interrupt cause register and handles the interrupt source. Is also cleans the interrupt in the unit level interrupt cause register. Figure 38 explains the interrupt handling procedure.

Figure 38: Interrupt Handling Procedure



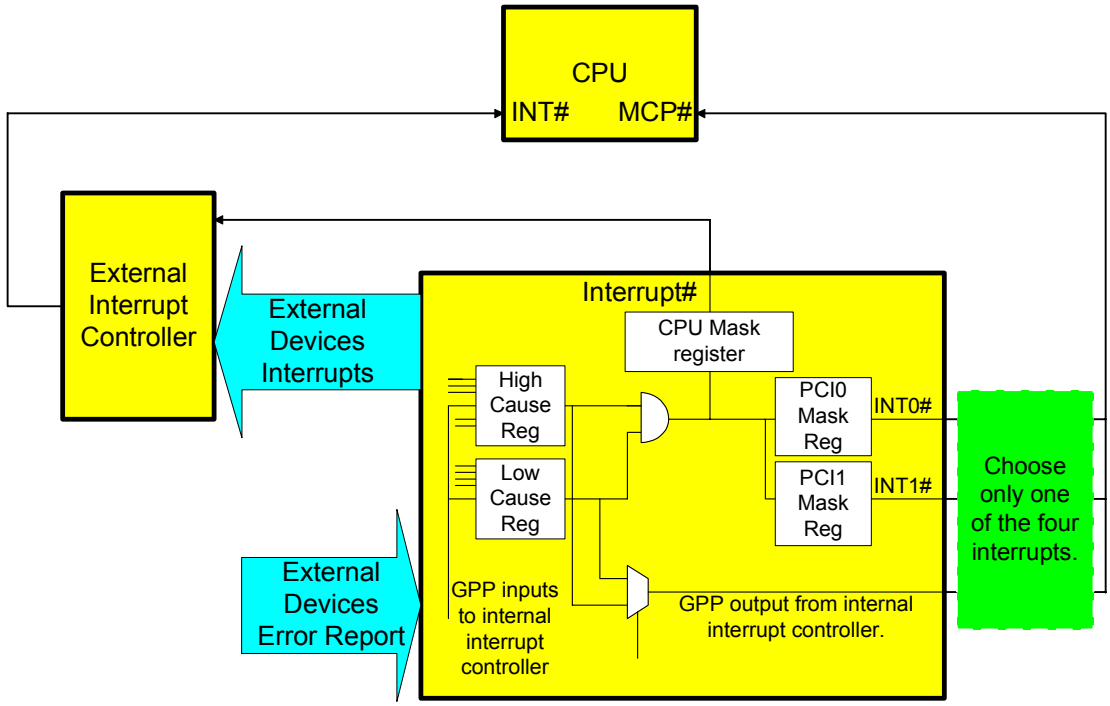
Note

More than one unit can generate an interrupt concurrently. The software's must set the priorities of the pending interrupts and handle them in the correct order.

11.1 Using External Interrupt Controller

The GT-64260A interrupt controller can be connected to an external interrupt controller. This configuration is useful when the software must be backward compatible to previous designs. Figure 39 is an example of a system that uses an external interrupt controller.

Figure 39: External Interrupt Controller



Section 12. Messaging Units Functional Overview

The GT-64260A messaging unit includes hardware hooks for message transfers between PCI devices and the CPU. The messaging unit can be divided into three different messaging types: messaging, doorbell, and I_2O.

Note

The polarity of the messaging unit doorbells, interrupt cause, and interrupt mask registers bits are determined via the Queue Control register's Polarity bit 8, at offset 0x1C50. If set to '0', interrupts are masked by a mask bit set to '1' and cause bits are cleared by writing '1'. If set to '1', interrupts are masked by a mask bit set to '0' and cause bits are cleared by writing '0'.

12.1 Messaging

These types of messages send and receive short messages over the PCI bus, without transferring data into local memory. Messages from PCI-to-CPU use the Inbound message mechanism, and messages from CPU-to-PCI use the Outbound message mechanism. The GT-64260A implements four channels of messaging types, two inbound and two outbound. Each channel implements one message register, one interrupt cause bit, and one interrupt mask bit.

Each Inbound Message register can only implement a single message. This means two messages per PCI interface (total of four messages). It is the system's responsibility to restrict the message sender to a single message by using the outbound message or any other acknowledge. If the system cannot restrict the message generator to a single message, it must use the Circular Queues mechanism.

For more information, see the GT-64260A datasheet's "Message Registers" section.

12.2 Doorbell

This mechanism enables the system to generate interrupts to CPU and PCI interfaces by using the Doorbell registers. The doorbell to the CPU is generated by the PCI which sets a bit in the Inbound Doorbell register, at offsets 0x1c20 and 0x1ca0. An interrupt to the PCI is generated by the CPU which sets a bit in the Outbound doorbell register, at offsets 0x1C2C and 0x1CAC.

The PCI cannot clear the Inbound Doorbell register. It can only set the bits to '1' and generate an interrupt.

Note

In the above sequence, the CPU must clear a bit in the Inbound Doorbell register by writing '1' and not '0'.

For more information, see the GT-64260A datasheet's "Doorbell Registers" section.

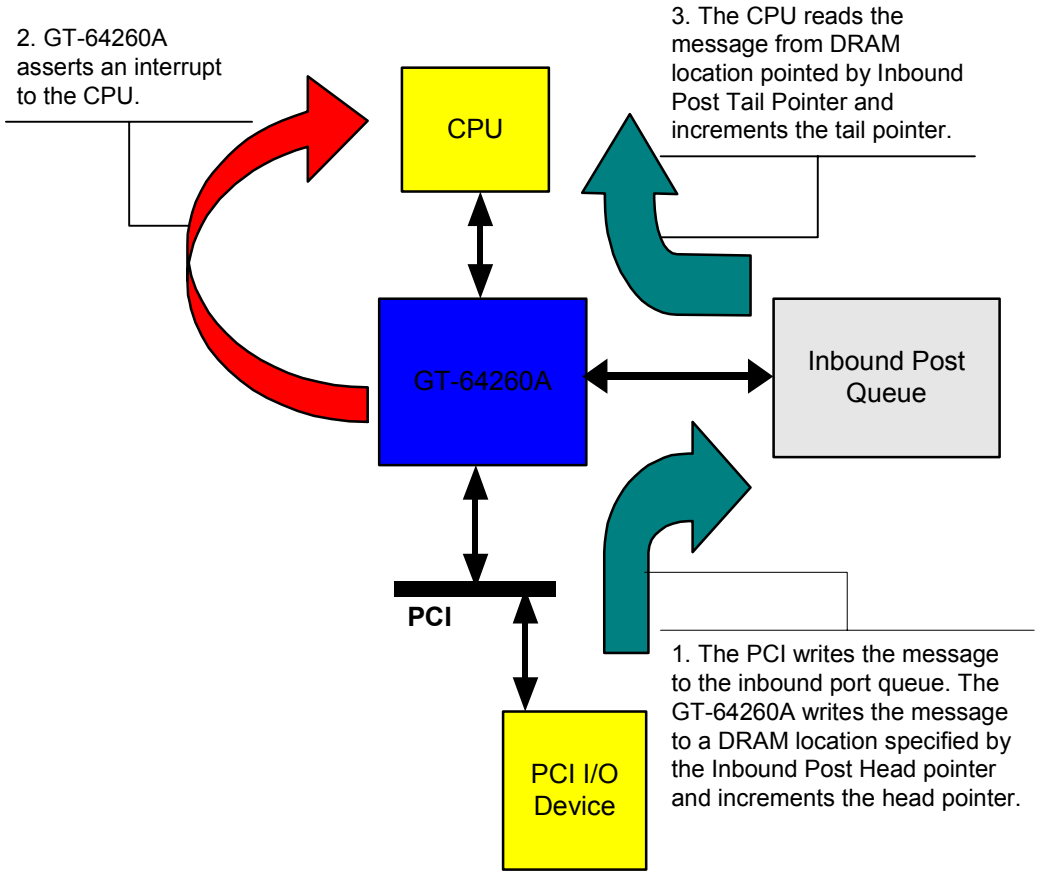
12.3 Circular Queue

The Circular Queue mechanism is used for passing queued messages between intelligent agents on a PCI bus to the CPU. The I_2O mechanism supplies the user with a useful way of passing messages on the SDRAM from PCI devices to the CPU without accessing the SDRAM.

12.3.1 Inbound Circular Queue

The PCI device writes to the Inbound Queue Port Virtual register (Offset: 0x1c40 and 0x1cc0) and the GT-64260A asserts an interrupt to the CPU and increments the head pointer. The CPU reads the message, increments the tail pointer, and writes to the Inbound Free Head Pointer register (Offset: 0x1c60 and 0x1ce0). This asserts an interrupt to the PCI device as a message acknowledgement. See Figure 40.

Figure 40: Inbound Circular Queue



The inbound message handling is as follows:

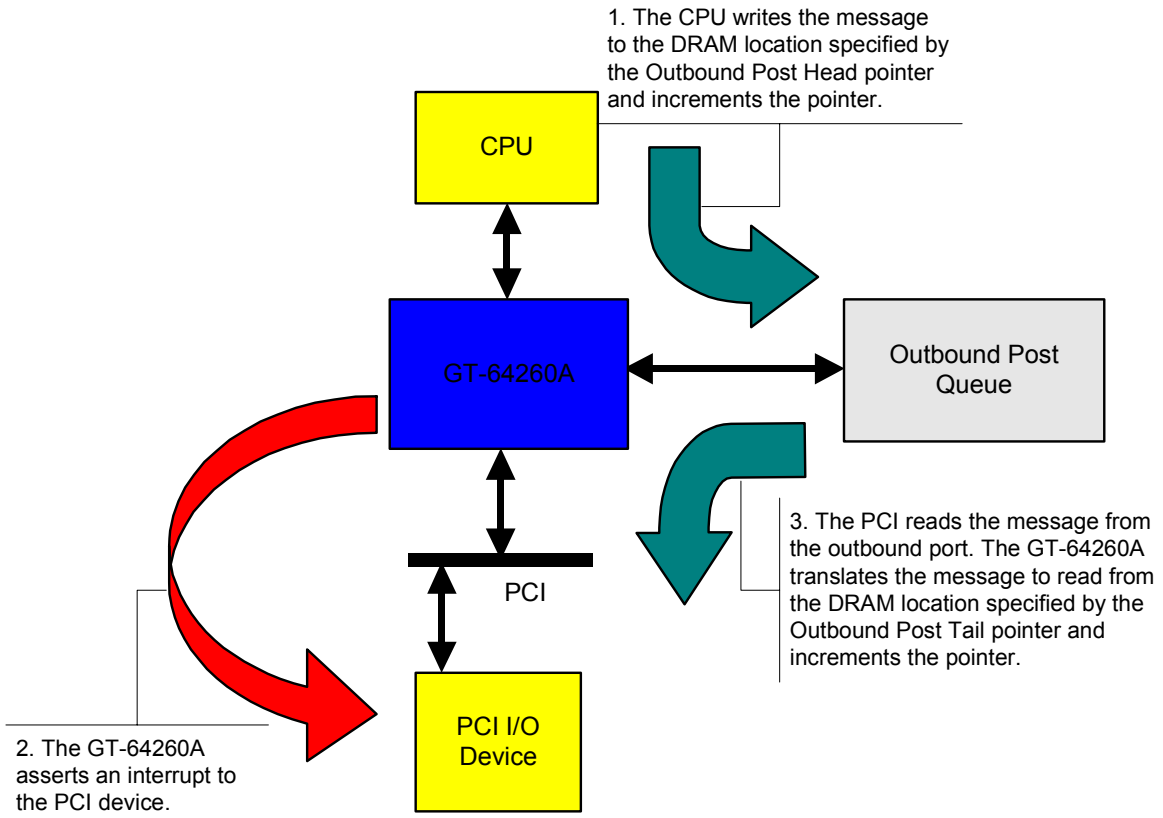
1. When not using interrupts while in idle state (no message pending), the CPU must poll the Inbound Interrupt Cause register's `InPQ` bit [4], at offsets 0x1c24 and 0x1ca4. When interrupt is used, the interrupt handler must identify the interrupt cause.

2. When InPQ is set to '1', the CPU must clear it.
3. At this point, the CPU reads the Inbound Post Head Pointer register, at offset 0x1c68, and the Inbound Post Tail Pointer register, at offset 0x1c6c, so it can calculate the number of pending messages.
4. The CPU starts reading the pending messages. It increments the tail pointer for each message.
5. The CPU returns to Step 1 until all the pending messages are read.

12.3.2 Outbound Circular Queue

The CPU writes to the Outbound Queue Port Virtual Register (Offset: 0x1c44 and 0x1cc4) and increments the head pointer. The GT-64260A asserts an interrupt to the PCI device. The PCI device reads the message and the tail pointer is incremented by the GT-64260A. The PCI device must write to the Outbound Free Head Pointer register (Offset: 0x1c70 and 0x1cf0). This write asserts an interrupt to the CPU as a message acknowledgement, Figure 41.

Figure 41: Outbound Circular Queue



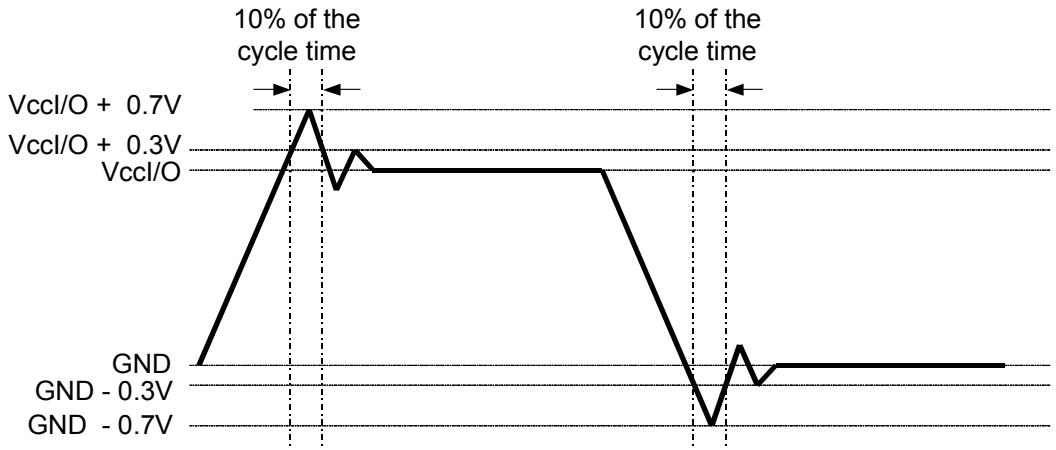
For more information, see the GT-64260A datasheet's "Circular Queues Data Storage".

Section 13. Design Consideration Overview

This following sections describe the various connectivity, Electrical Specification, termination topology, placement, timing constraints, and routing rules for the GT-64260A interfaces. The purpose of these sections is to help the designer achieve improved signal integrity on the board and avoid timing problems.

Figure 42 describes the allowable undershoot and overshoot voltage for the GT-64260A. This figure is applicable to all GT-64260A interfaces, except for the PCI interface. The PCI interface pads are PCI complaint and their maximum and minimum rating are compliant to the PCI specification 2.2. document section "4.2.2.3 Maximum AC Ratings and Device Protection".

Figure 42: GT-64260A Overshoot/Undershoot Voltage



Section 14. CPU Interface Design Considerations

The CPU interface is used to connect the GT-64260A and the CPU.

The GT-64260A CPU interface is a 60x bus compatible master and slave. In addition, it is a MPX bus compatible slave.

Note

As an MPX bus compatible slave, this interface only supports "address only bus mastering" for snooping capability.

The GT-64260A CPU interface internal 60x arbiter supports up to two external 60x masters and the GT-64260A 60x master. The MPX internal arbiter supports one external MPX master and the GT-64260A MPX master, for address only transactions. In Multi-GT mode, the GT-64260A supports up to four slaves connected on the same 60x bus.

The GT-64260A CPU bus configuration and internal bus arbiter and multi-GT support is sampled at reset according to the following configuration.

Table 13: CPU Interface Configuration at Reset

Pin(s)	Configuration Function
AD[7:6]	CPU Bus Configuration
00- 01- 10- 11-	60x bus MPX bus Reserved Reserved
AD[8]	Internal 60x bus Arbiter
0- 1-	Not supported Supported If using the MPX bus mode, AD[8] must be set to 1.
AD[9]	Multiple GT-64260A Support
0- 1-	Not supported Supported

14.1 CPU Interface Connectivity

The CPU bus connectivity is according to the 60x/MPX specification

14.2 Electrical Specification

The CPU interface can be configured to support 3.3V or 2.5V IO voltage. This is configured at reset by sampling the AD[31].

- 0 = 2.5V IO
- 1 = 3.3V IO

14.3 Termination Topology

The 60x and MPX standards do not require any termination. Simulate the board topology to determine if termination is needed.

14.4 Timing Requirements

The CPU interface can run up to 133 MHz. The timing is very tight and the designer must simulate the system to work in the maximum frequency.

14.4.1 Calculating the Reference Point

The output delay values in the GT-64260A datasheet's AC timing table is defined for a specific load. This value includes the rise/fall time of the output. To calculate the signal fly time, the rise/fall time must be measured and a reference measuring point must be set. Figure 43 describes the test circuit for the GT-64260A (Cload = 15pf):

Figure 43: GT-64260A Test Circuit (Cload = 15pf)

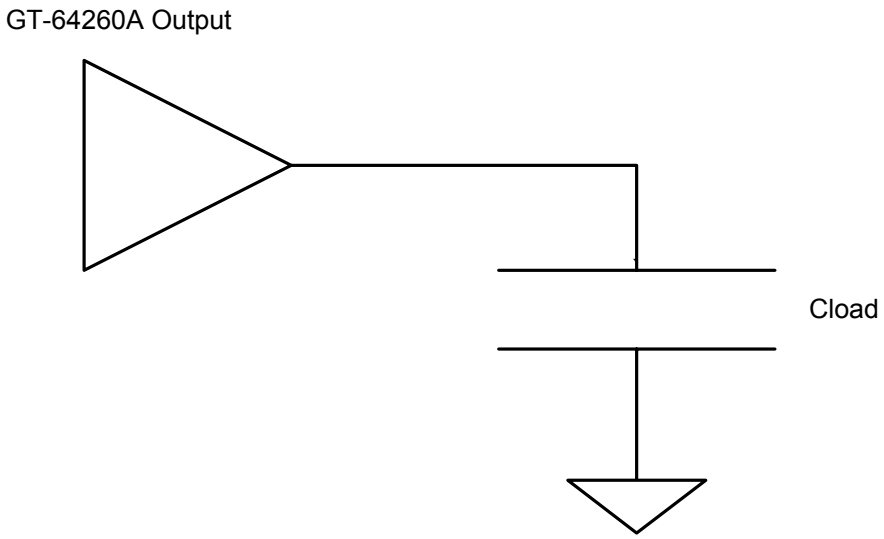
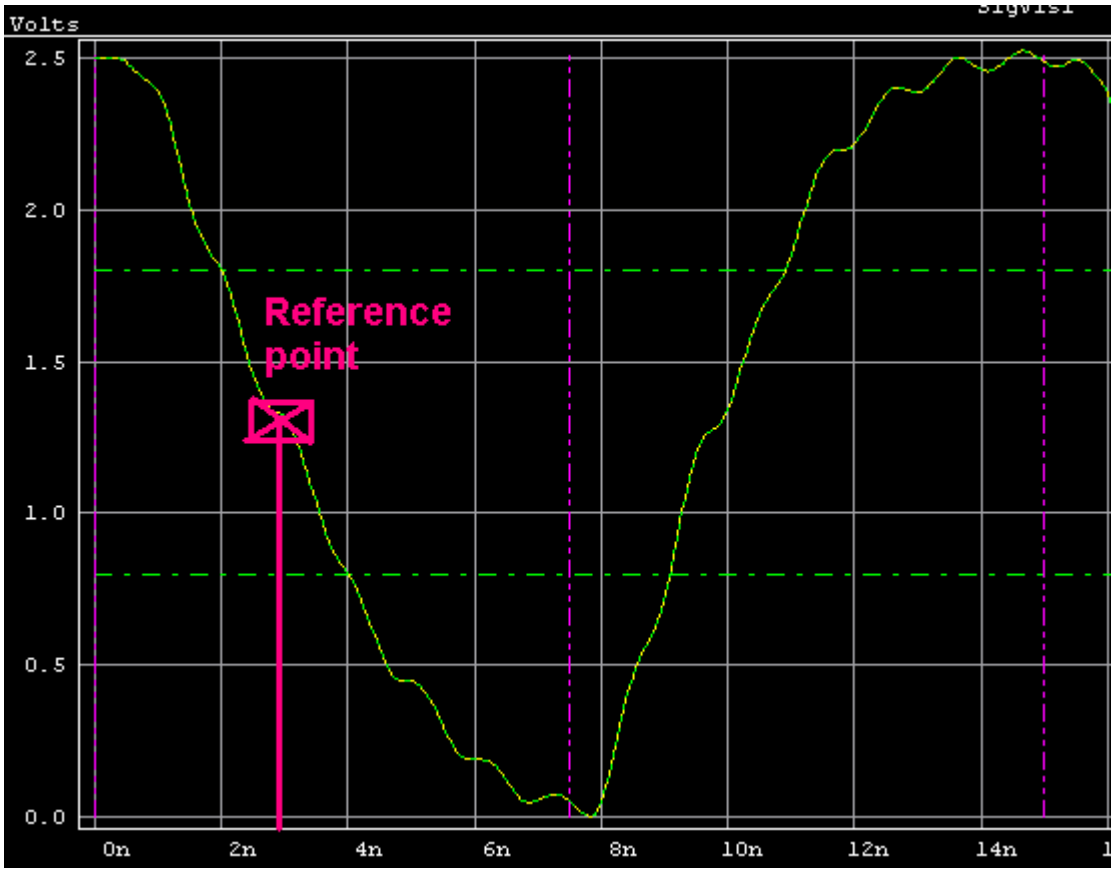


Figure 44 shows the results of the test circuit simulation.

Figure 44: Test Circuit Results (Cload = 15pf)



OVdd/2 = 1.25V at 3 ns in Figure 46. This is marked as the reference point to measure the signal fly time in the system simulation. The reference point for the rising edge is smaller than the falling edge (2.1 ns), so the reference point is determined by the falling edge. The fly time is measured from the reference point to the Vil measured on the load.

The output delay values in the AC timing table of the CPU datasheet are defined for a specific test circuit. This value includes the rise/fall time of the output. To calculate the signal fly time, the rise/fall time must be measured and a reference measuring point must be set. The CPU reference point is measured in a similar way as the GT-64260A, but with a different test circuit. Figure 45 shows the CPU test circuit (Rload = 50 Ohm).

Figure 45: GT-64260A Test Circuit (Rload = 50 Ohm)

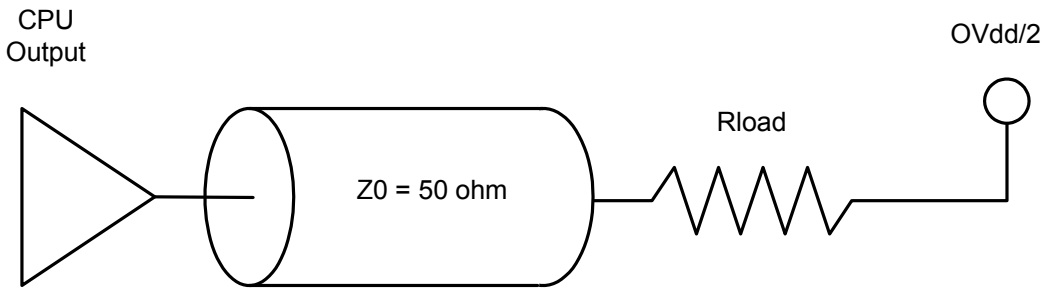
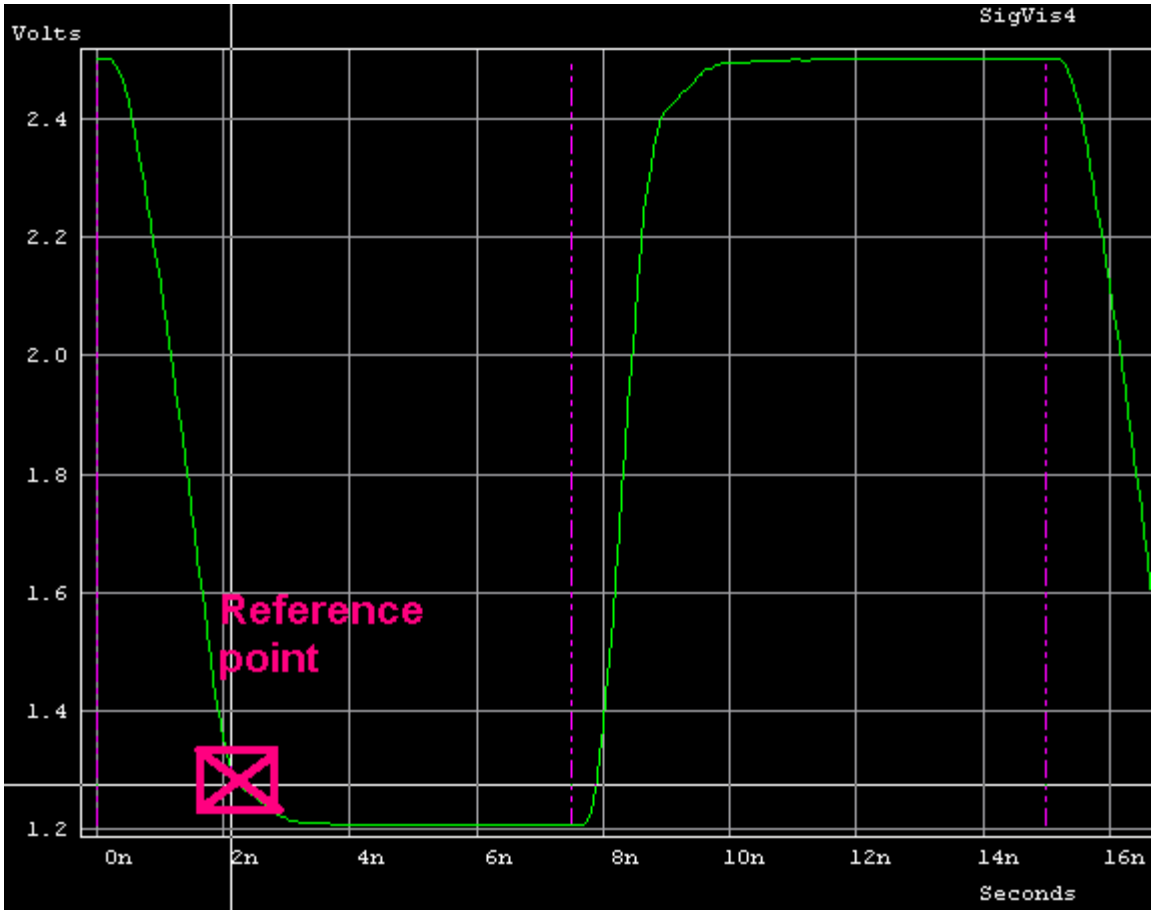


Figure 46 shows the results of the test circuit simulation.

Figure 46: Test Circuit Results (Rload = 50 Ohm)



14.4.2 Timing Simulation

The timing calculations depend on the system configuration. The configuration changes include the CPU type and number of devices connected to the CPU bus. Most CPU AC timings will match the following tables.

Table 14: Typical CPU AC Timings

Parameter	Value		Unit
	Min	Max	
Input Setup	2		ns
Input hold	1		ns
output delay	0.5	3	ns

The following sections provide the CPU interface timing requirements for some typical system configurations.

Single GT-64260A and a Single CPU System (Including Cache Coherency) in Single-GT Mode

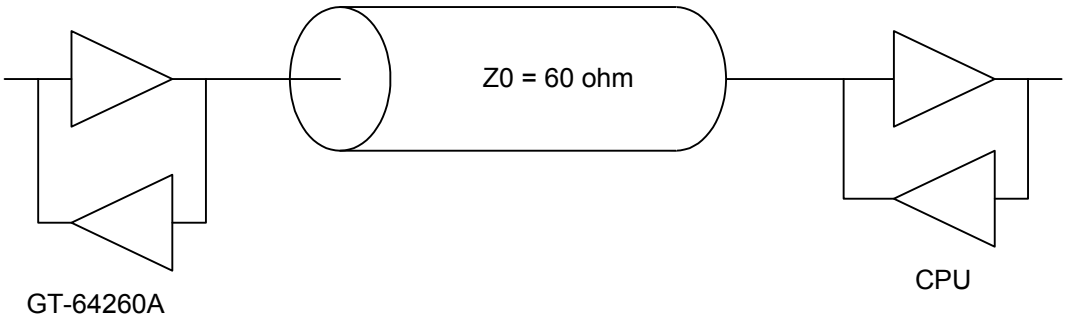
Table 15: Single-GT and Single CPU AC Timing

Parameter	Value		Unit
	Min	Max	
Input Setup[1]	3		ns
Input hold	0		ns
output delay	1	4.2	ns

1. ARTRY* for address only.

In this configuration, the signals are connected from the GT-64260A to the CPU in a point-to-point configuration. (See Figure 47.)

Figure 47: GT-64260A to CPU Point-to-Point Configuration



The timing requirements for the CPU to the GT-64260A (based on Table 14 on page 95 and Table 15 on page 95) are:

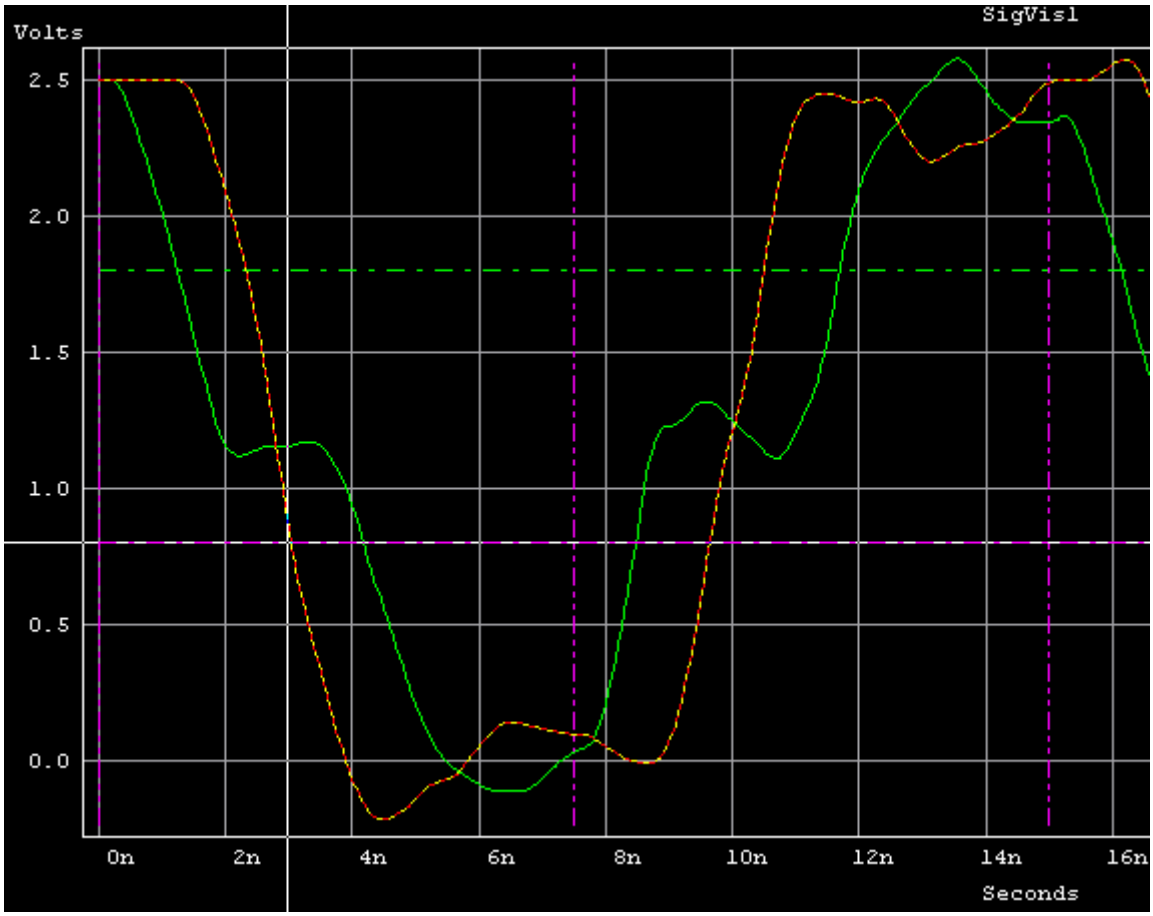
Tcycle > Toutput_delay(CPU) + Tsetup(GT-64260A) + Tdelay(fly_time) + Tclock_skew

7.5 > 3 + 3 + Tdelay(fly_time) + 0.5

Tdelay(fly_time) < 1 ns

For 200 ps delay for 1 inch, the maximum distance is 5 inches. Figure 48 shows a simulation of 1 ns delay trace. The fly time is measured from the CPU reference point that was measured in "Calculating the Reference Point" on page 92 (2.1 ns) to the Vil measured on the GT-64260A pin (3.1 ns) in Figure 48 (board simulation).

Figure 48: 1 ns Delay Trace Simulation



The timing requirements for the CPU to the GT-64260A are:

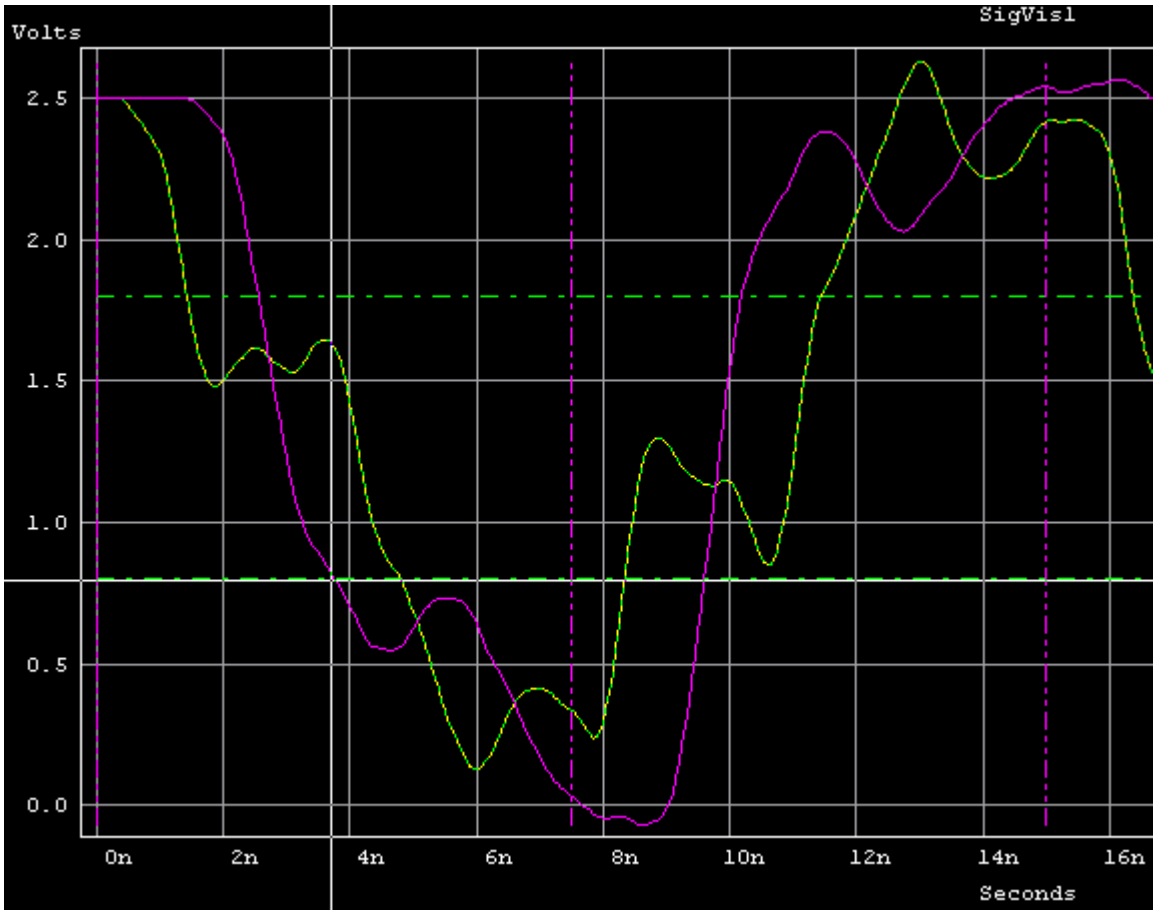
Tcycle > Toutput_delay(GT-64260A) + Tsetup(CPU) + Tdelay(fly_time) + Tclock_skew

7.5 > 4.2 + 2 + Tdelay(fly_time) + 0.5

Tdelay(fly_time) < 0.8 ns

For 200 ps delay for 1 inch, the maximum distance is 4 inches. Figure 49 shows a simulation of 0.8 ns delay trace. The fly time is measured from the GT-64260A reference point that was measured in “Calculating the Reference Point” on page 92 (3 ns) to the Vil measured on the CPU pin (3.8 ns) in Figure 49 (board simulation).

Figure 49: 0.8 ns Delay Trace Simulation



Single GT-64260A and Multiple CPU System (or Multi Data Masters Systems) in Multi-CPU Mode

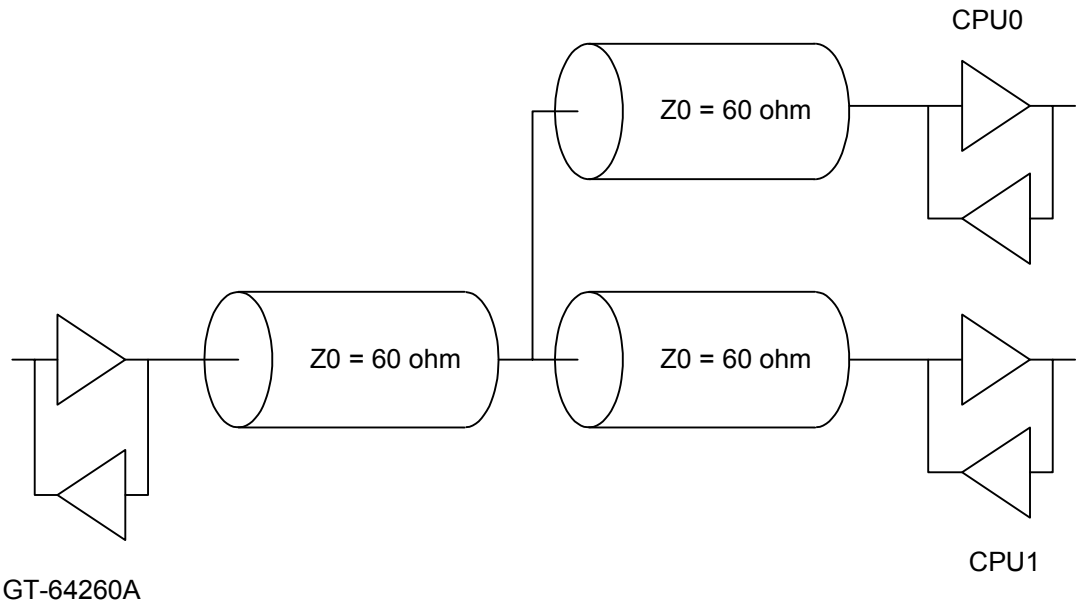
Table 16: Single-GT and Multiple CPU AC Timing

Parameter	Value		Unit
	Min	Max	
Input Setup[1]	3.5		ns
Input hold	0		ns
output delay	1	4.2	ns

1. ARTRY* for multiple data masters.

In this configuration, the signals are connected from the GT-64260A to the CPUs in a 'T' topology. (See Figure 50.)

Figure 50: GT-64260A to Multiple CPU Configuration



From Table 14 on page 95 and Table 16 on page 98, the timing requirements for the CPU to GT-64260A are as follows:

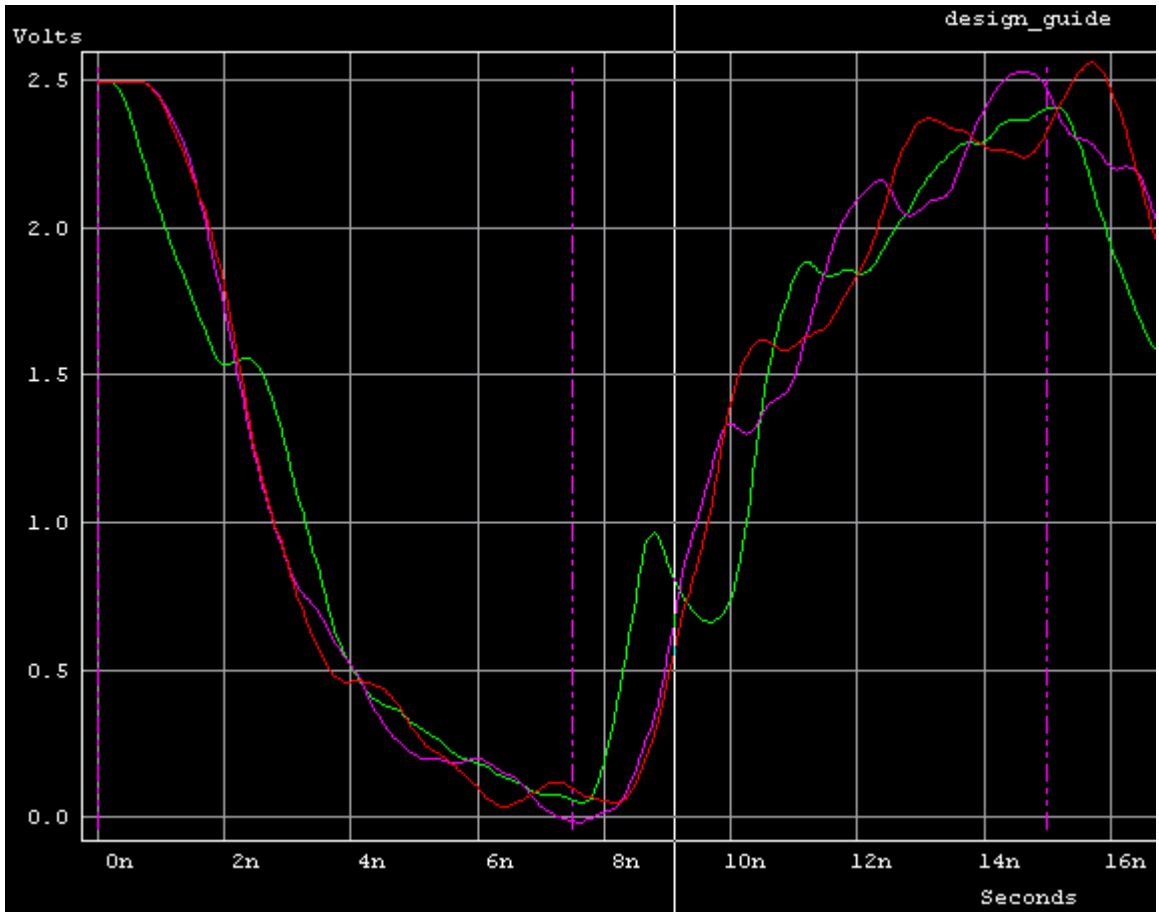
Tcycle > Toutput_delay(CPU) + Tsetup(GT-64260A) + Tdelay(fly_time) + Tclock_skew

7.5 > 3 + 3.5 + Tdelay(fly_time) + 0.5

Tdelay(fly_time) < 0.5 ns

For 200 ps delay for 1 inch, the maximum distance is 2.5 inches. Figure 51 shows a simulation of 0.5 ns delay trace. The fly time is measured from the CPU reference point that was measured in 14.4.1 "Calculating the Reference Point" on page 92 (2.1 ns) to the Vil measured on the GT-64260A pin (2.6 ns) in Figure 51(board simulation).

Figure 51: 0.5 ns Delay Trace Simulation (Maximum Distance 2.5 Inches)



The GT-64260A to CPU calculation is the same as in the single GT-64260A and single CPU example.

Note

For multiple CPU configurations, a separate IBIS model must be used. For more information, contact your local Field Application Engineer (FAE).

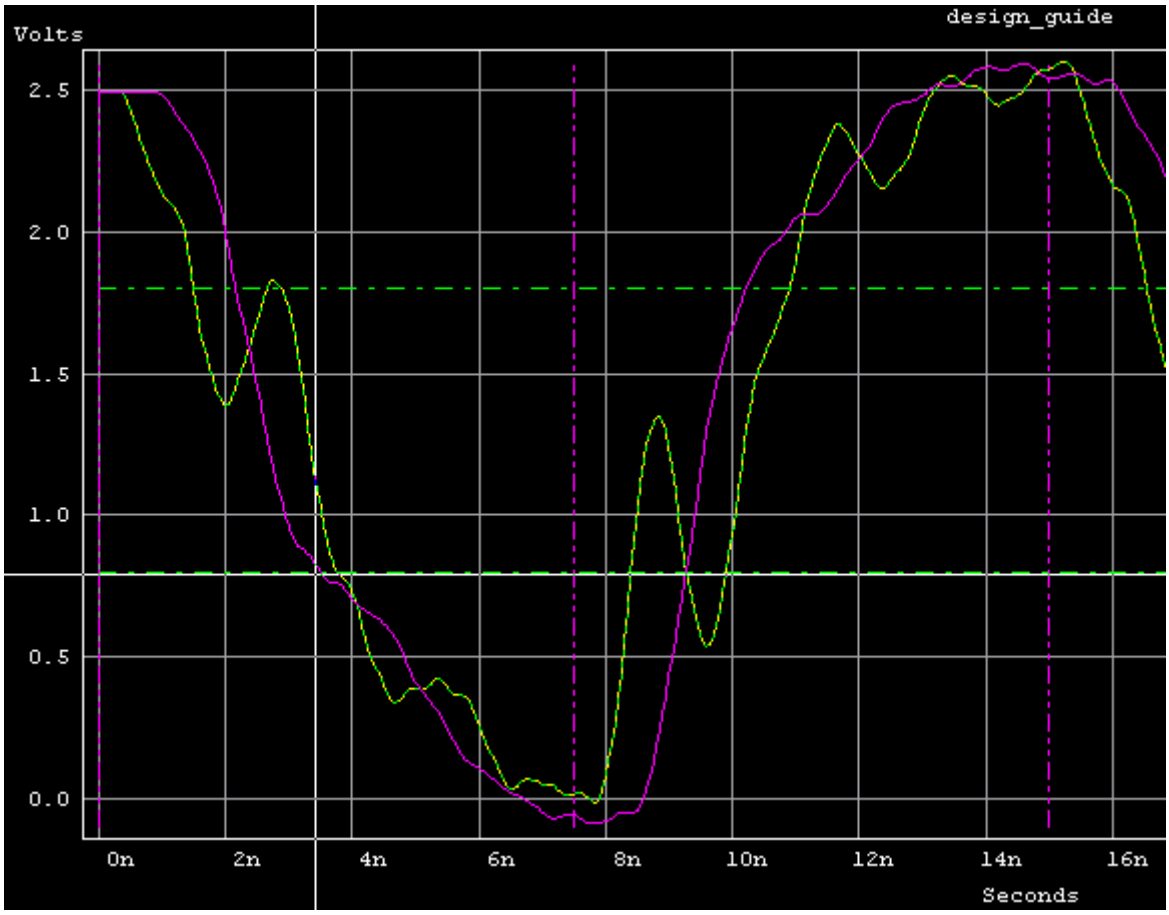
Tcycle > Toutput_delay(GT-64260A) + Tsetup(CPU) + Tdelay(fly_time) + Tclock_skew

7.5 > 4.2 + 2 + Tdelay(fly_time) + 0.5

Tdelay(fly_time) < 0.8 ns

For 200 ps delay for 1 inch, the maximum distance is 4 inches. Figure 52 shows a simulation of 0.5 ns delay trace. The fly time is measured from the GT-64260A reference point that was measured in 14.4.1 "Calculating the Reference Point" on page 92 (3 ns) to the Vil measured on the CPU pin (3.4 ns) in Figure 52 (board simulation).

Figure 52: 0.5 ns Delay Trace Simulation (Maximum Distance 4 Inches)



Multiple GT-64260As and a Single CPU System in Multi-CPU Mode

Table 17: Multiple GT-64260As and a Single CPU AC Timing

Parameter	Value		Unit
	Min	Max	
Input Setup[1]	4.5		ns
Input hold	0		ns
output delay	1	4.2	ns

1. AACK* for multi-GT mode.

Note

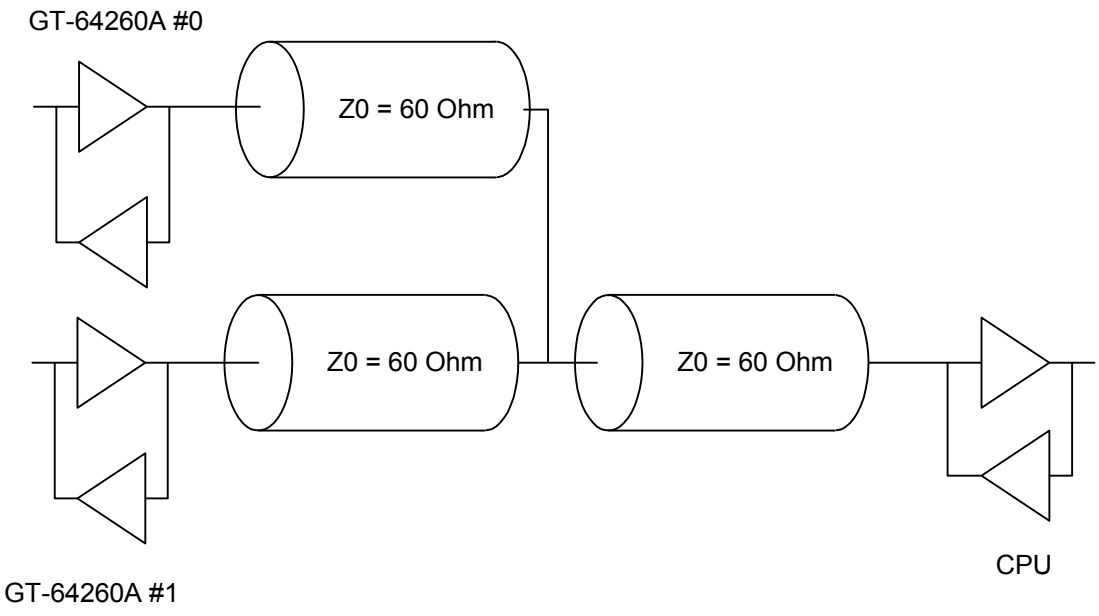
In multi-GT mode, the ARTRY* signals setup time is 4.9 ns. This is ignored in the timing calculation example, since most applications do not use the ARTRY*.

In this configuration, the signals are connected from the GT-64260A to the CPU in a 'T' topology. See Figure 53.

Note

The GT-64260A in multi-GT mode is targeted to operate at 100 MHz.

Figure 53: Multiple GT-64260As to a Single CPU Configuration



The timing requirements for multiple GT-64260As to a single CPU (based on Table 14 on page 95 and Table 17) are:

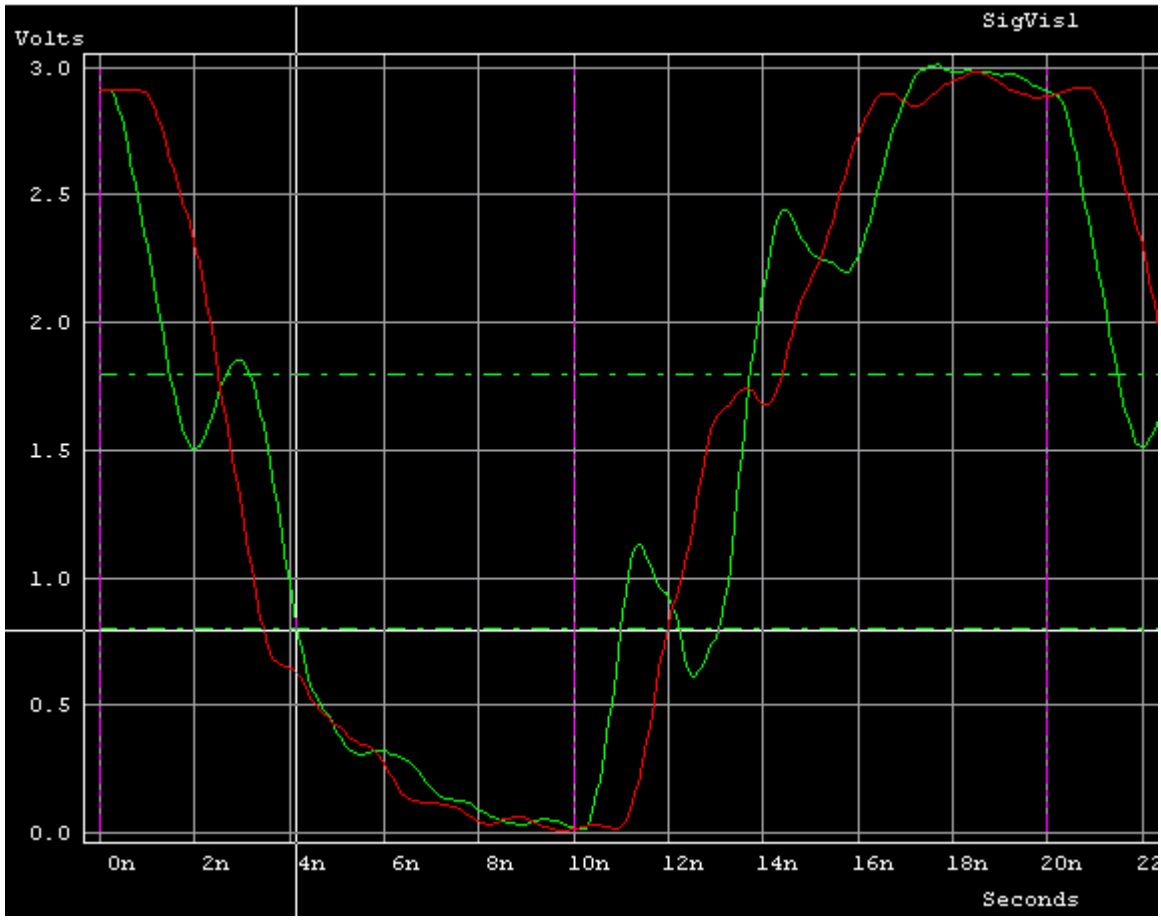
Tcycle > Toutput_delay(CPU) + Tsetup(GT-64260A) + Tdelay(fly_time) + Tclock_skew

10 > 3 + 4.5 + Tdelay(fly_time) + 0.5

Tdelay(fly_time) < 2 ns

Figure 54 shows a simulation of 1.1 ns delay trace. The fly time is measured from the CPU reference point that was measured in 14.4.1 "Calculating the Reference Point" on page 92 (2.1 ns) to the Vil measured on the GT-64260A pin (3.2 ns) in Figure 54 (board simulation).

Figure 54: 1.1 ns Delay Trace Simulation



The GT-64260A to CPU calculation is the same as the single GT-64260A and a single CPU but the longest path is from one GT-64260A to the other GT-64260A.

Note

For the multiple GT-64260As configuration, separate AC timings must be used. See the GT-64260A datasheet's "AC Timing" section or contact a Marvell FAE for more information.

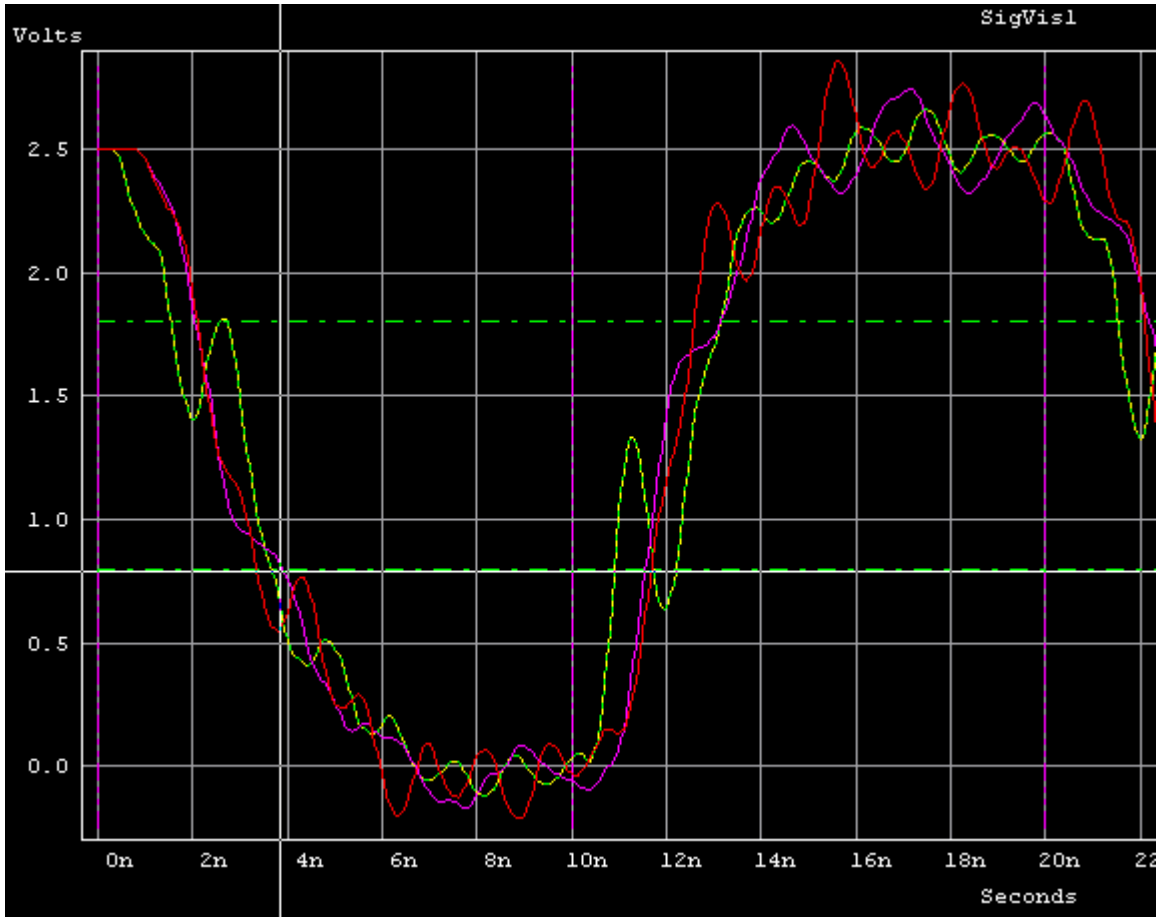
Tcycle > Toutput_delay(GT-64260A) + Tsetup(GT-64260A) + Tdelay(fly_time) + Tclock_skew

10 > 4.2 + 4.5 + Tdelay(fly_time) + 0.5

Tdelay(fly_time) < 0.8 ns

Figure 55 shows simulation of 0.8 ns delay trace. The fly time is measured from the GT-64260A reference point that was measured in 14.4.1 "Calculating the Reference Point" on page 92 (3 ns) to the Vil measured on the CPU pin (3.8 ns) in Figure 55 (board simulation).

Figure 55: 0.8 ns Delay Trace Simulation



14.5 Layout Instructions

14.5.1 Placement

All devices must be placed as close as possible to each other. Figure 56 shows the placement for single a GT-64260A connected to a single CPU. Figure 57 shows the placement for single a GT-64260A connected to multiple CPUs.

Figure 56: Layout for a Single GT-64260A to a Single CPU

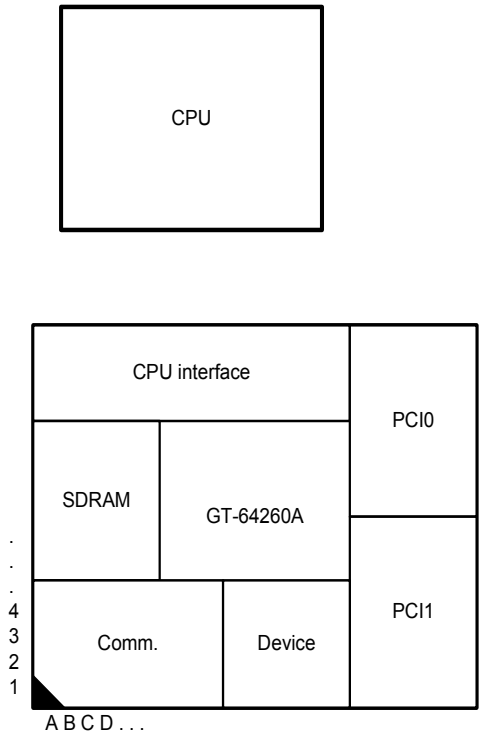
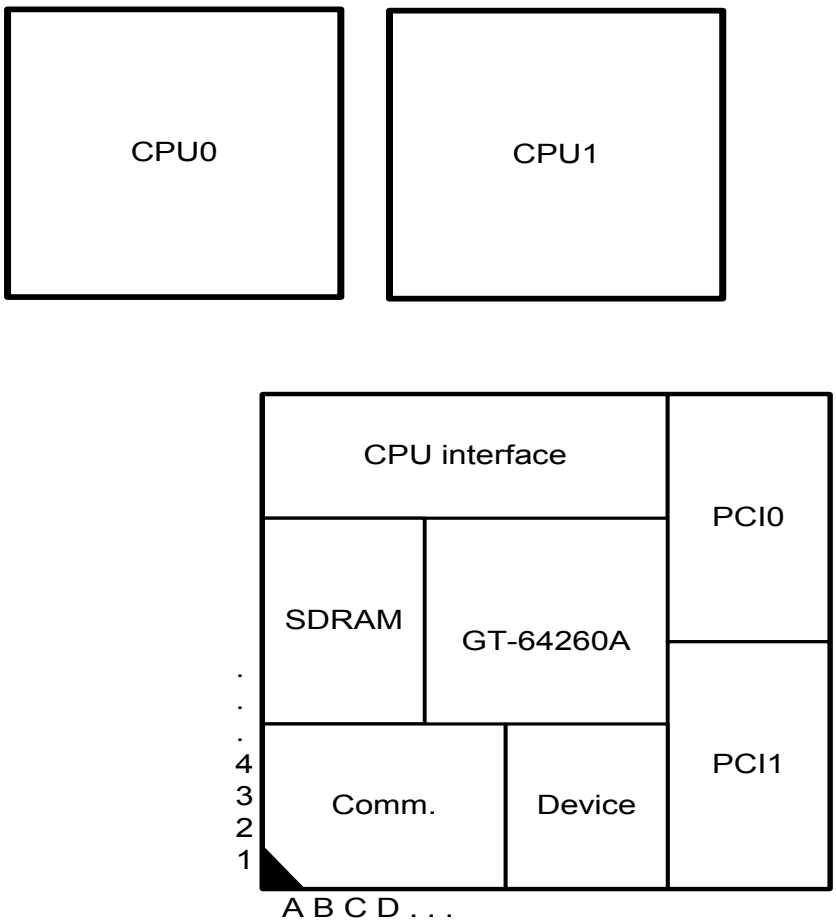


Figure 57: Layout for a Single GT-64260A to Multiple CPUs



The placement for multi-GT mode depends on the number of GT-64260A devices and interfaces used on each one of them.

Depending on the system configuration and timing simulation, parallel termination on the bi-directional signals can be placed near the CPUs or the GT-64260As.

14.5.2 Routing

The CPU interface traces must be 55 to 65 Ohm impedance.

The CPU and GT-64260A clocks (see section Section 19. "Clocks" on page 143) must be routed on separate layers from the other signals.

The point-to-multipoint topology signals must be routed in a "V" or "T" shape.

Section 15. SDRAM Interface Design Considerations

The CPU interface is used to connect the GT-64260A and the SDRAM. The GT-64260A supports three different SDRAM configurations.

- Registered SDRAM
- Heavy load SDRAM
- Regular SDARM

Note

For more information, see Section 4. "SDRAM Interface Functional Overview" on page 33.

The GT-64260A supports UMA mode as master or slave. The UMA mode is sampled at reset by AD[12] signal for enable/disable setting and AD[13] for master/slave setting. For more information, see Section 20. "Reset" on page 144.

15.1 Interface Connectivity

The SDRAM interface connectivity is according to the SDRAM specification. For more information on the SDRAM interface connectivity, see Section 4.2 "Memory Connection" on page 35.

15.2 Electrical Specification

The SDRAM interface is LVTTL-compatible (3.3V), up to 133 MHz.

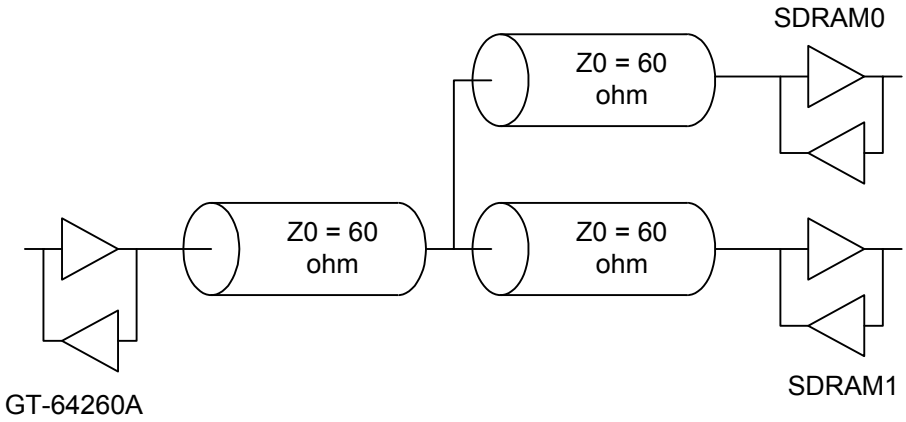
15.3 Termination Topology

The SDRAM standard does not require any termination.

To determine if termination is necessary, the user must simulate the board topology. Since the SDRAM interface was designed for large loads on the SDRAM address and control lines (IO pads of 24 mA), a serial termination might be required for a system with only a few SDRAM devices.

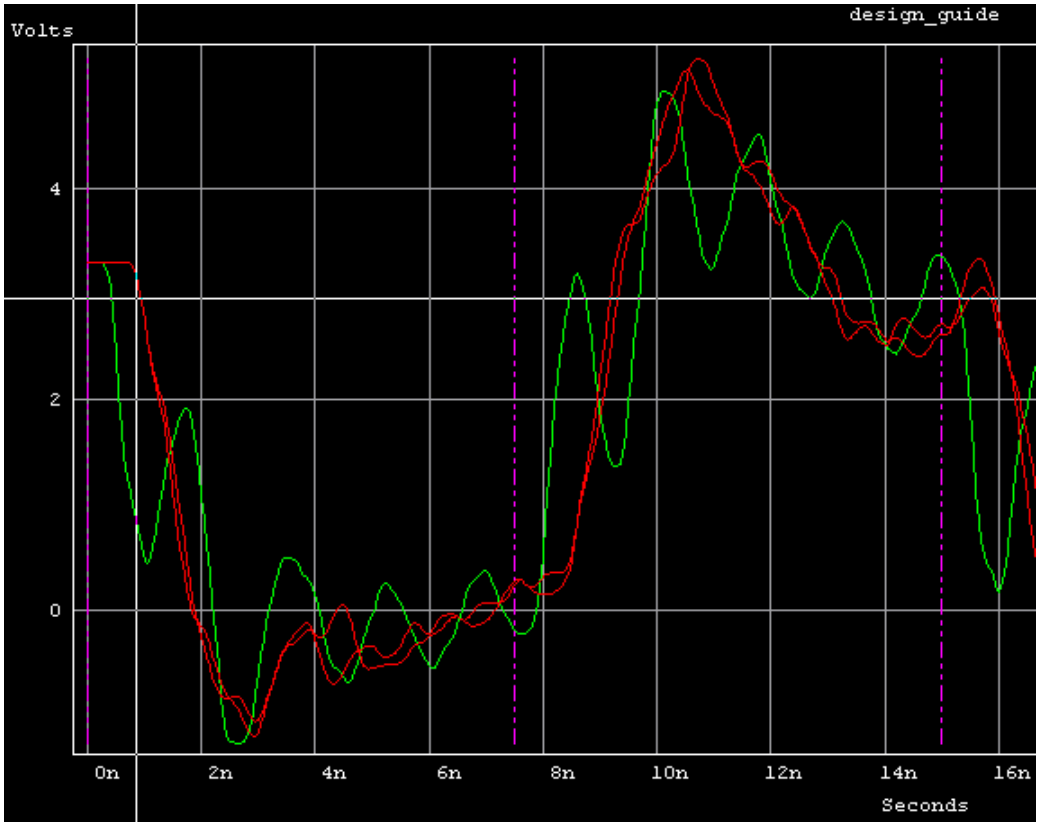
For example, see the configuration in Figure 58.

Figure 58: SDRAM Configuration Example



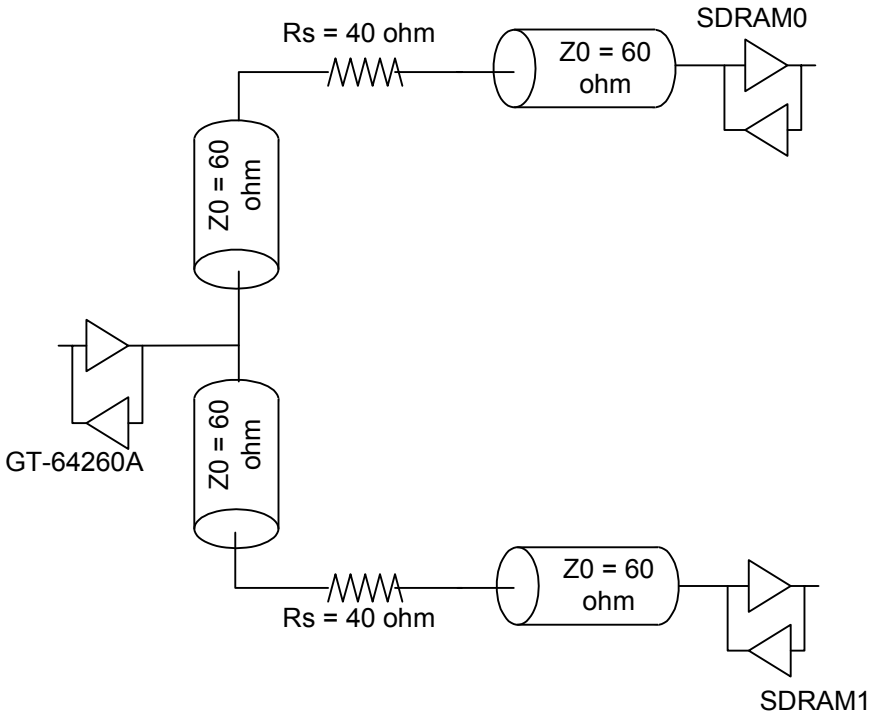
Simulating this configuration with the maximum case (fast corner) of the device gives the results shown in Figure 59.

Figure 59: SDRAM Simulation Example



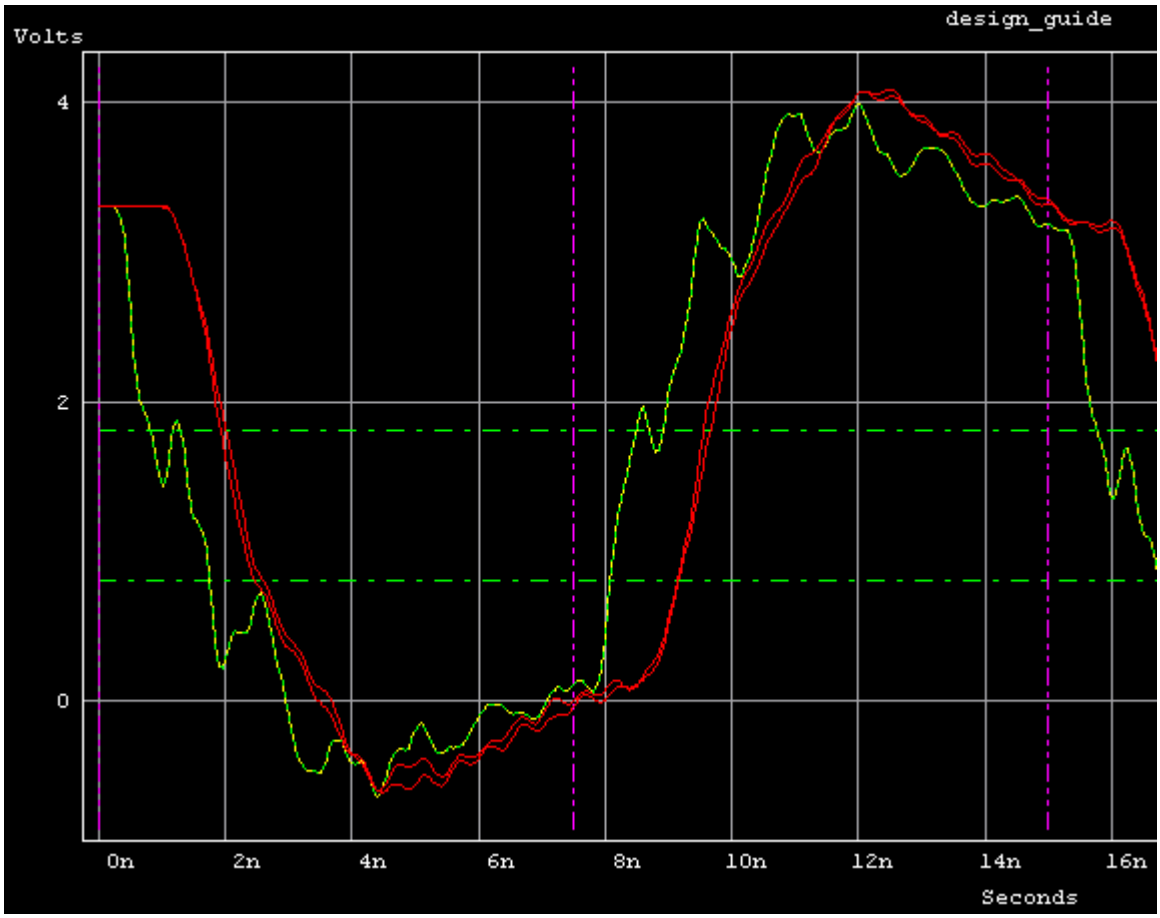
Changing the configuration in Figure 58 by adding serial resistors is shown in Figure 60.

Figure 60: SDRAM Configuration Example (With Resistors)



This topology simulation avoids the overshoot and undershoot of the previous configuration as shown in Figure 61.

Figure 61: SDRAM Simulation Example (With Resistors)



15.4 Timing Requirements

In this specification, the SDRAM timing-critical signals are categorized into five groups. Each group of signals includes the signals that have identical loading and routing topologies. Table 18 summarizes the signal groups.

Table 18: Signal Topology Categories

Group	Signals in group
Clock	SDRAM_CK[3:0]
Data	SDATA[63:0]
	ECC[7:0]
Data Mask*	SDQM[7:0]

Table 18: Signal Topology Categories (Continued)

Group	Signals in group
Chip select	SCS[3:0]
Double cycle signals	DAdr[12:0]
	BA[1:0]
	SRAS
	SCAS
	DWr

The following sections describe the routing requirements associated with each signal group

The timing simulation of the SDRAM interface depends on the memory configuration (i.e., number of devices for each physical bank, number of physical banks, device type, etc.). The simulation represent a configuration of one physical bank with eight devices (single cycle DIMM).

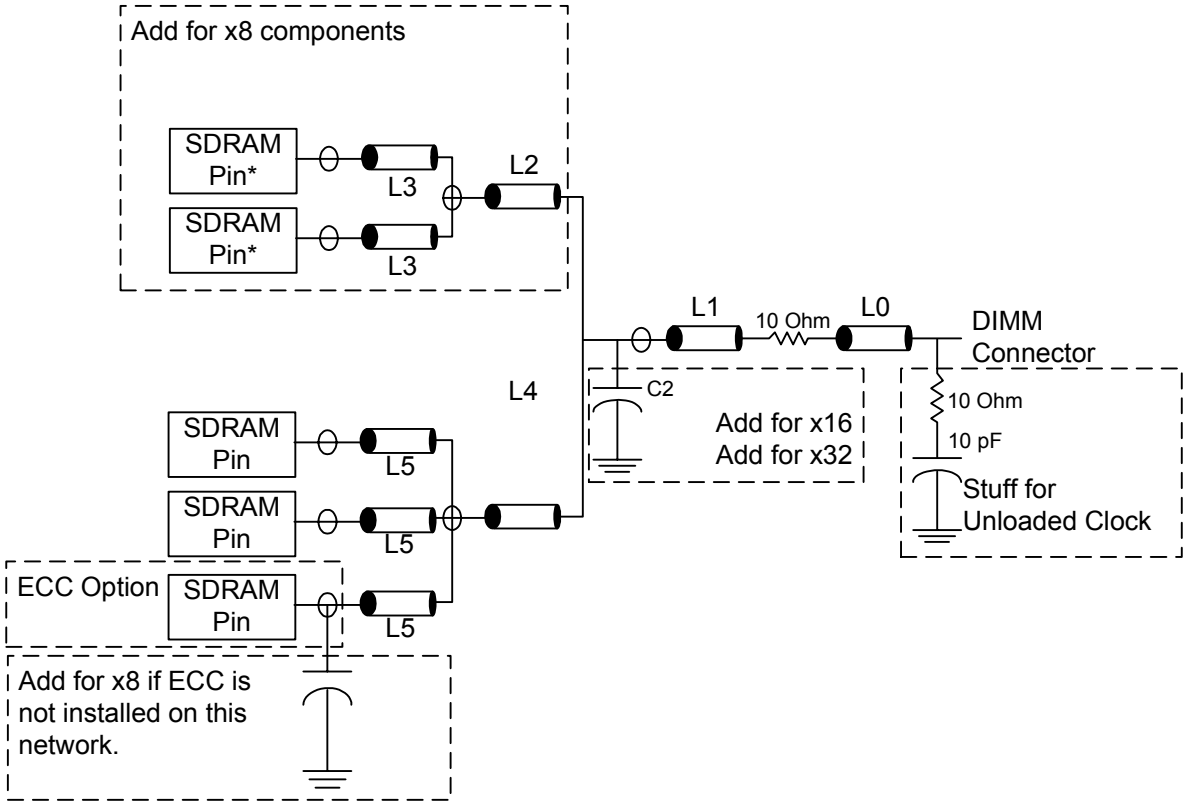
15.4.1 Clock Timing

The GT-64260A supports a few SDRAM clock configurations. For more information on the supported configurations, see Section 19. "Clocks" on page 143 or *AN-82: SDRAM Clocking Schemes in the GT-642xxA*.

Note

When SDRAM DIMM is used, the designer must take into consideration the clock signal's delay and load on the DIMM. Figure 62 describes the clock topology on the DIMM.

Figure 62: DIMM Clock Topology



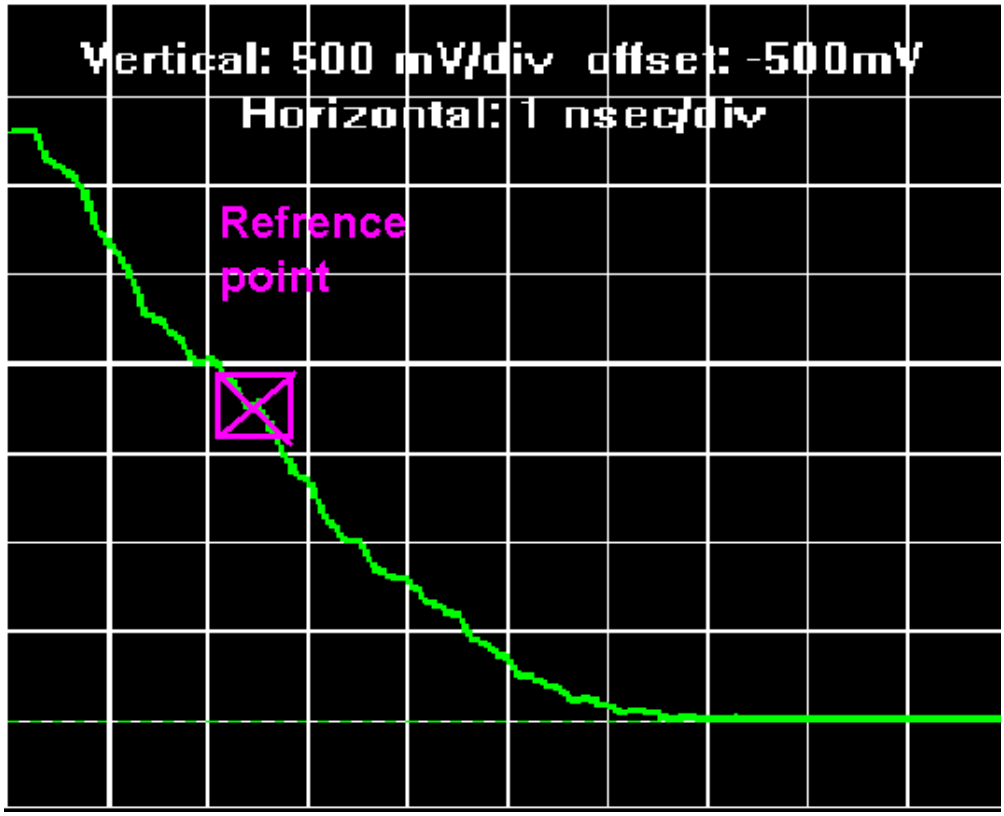
15.4.2 Data Timing

Since the AC specification of the GT-64260A and of the SDRAM devices is for a given test circuit, the first stage is to calculate the reference point for the timing calculations. The reference point is used as a starting point to measure the signal fly time.

The output delay value of the GT-64260A data signals in the AC Timings table are given for 30 pf load (Cload = 30pf). The test circuit is shown in Figure 43.

Simulating the GT-64260A SDRAM interface data signals test circuit will give a reference point of 2.3 ns. (See Figure 63.)

Figure 63: GT-64260A Data Reference Point



The SDRAM output delay AC timing uses a similar test circuit with load of 50 pf (Cl = 50pf). Simulating the SDRAM data signals test circuit will give reference point of 1.54 ns. (See Figure 64.)

Figure 64: SDRAM Data Reference Point

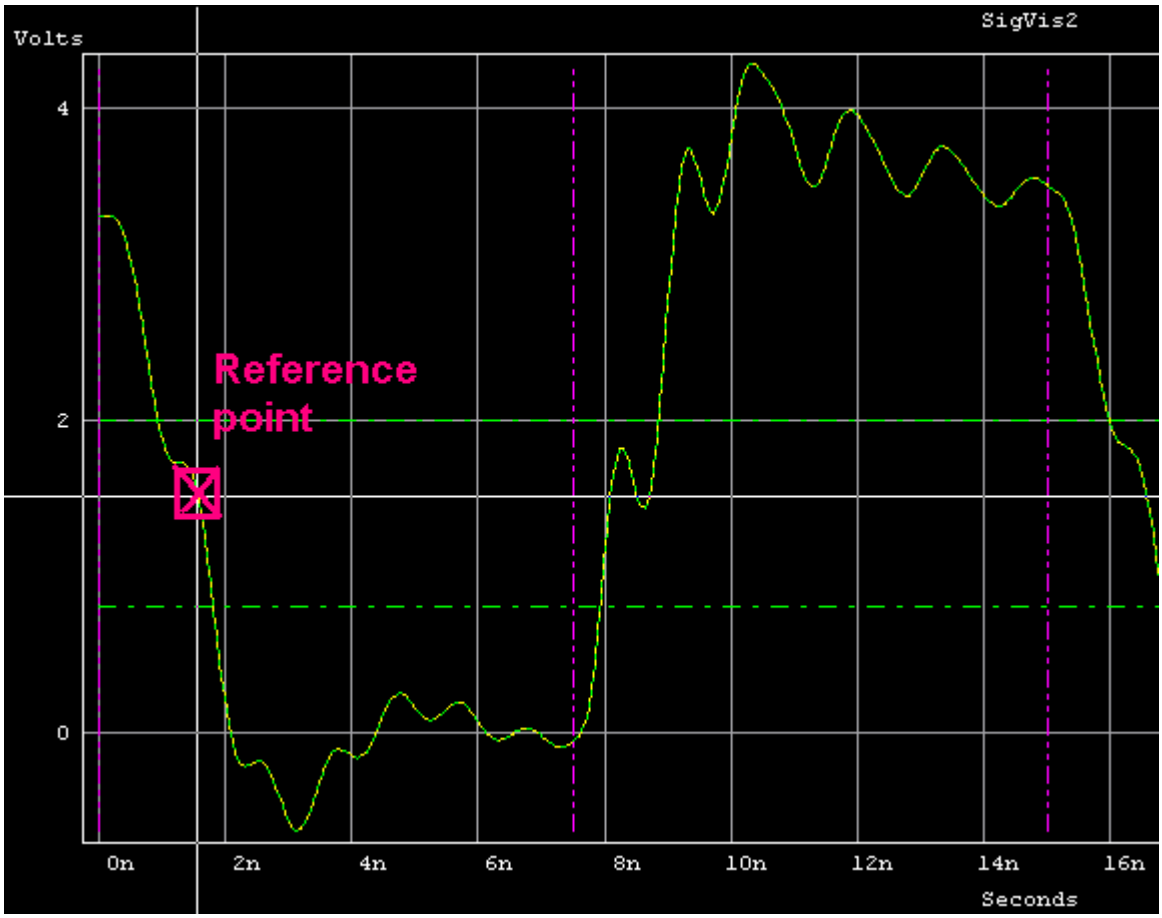


Figure 65 describes the data topology for the selected memory configuration (one physical bank with eight devices).

Figure 65: Selected Memory Configuration Data Topology

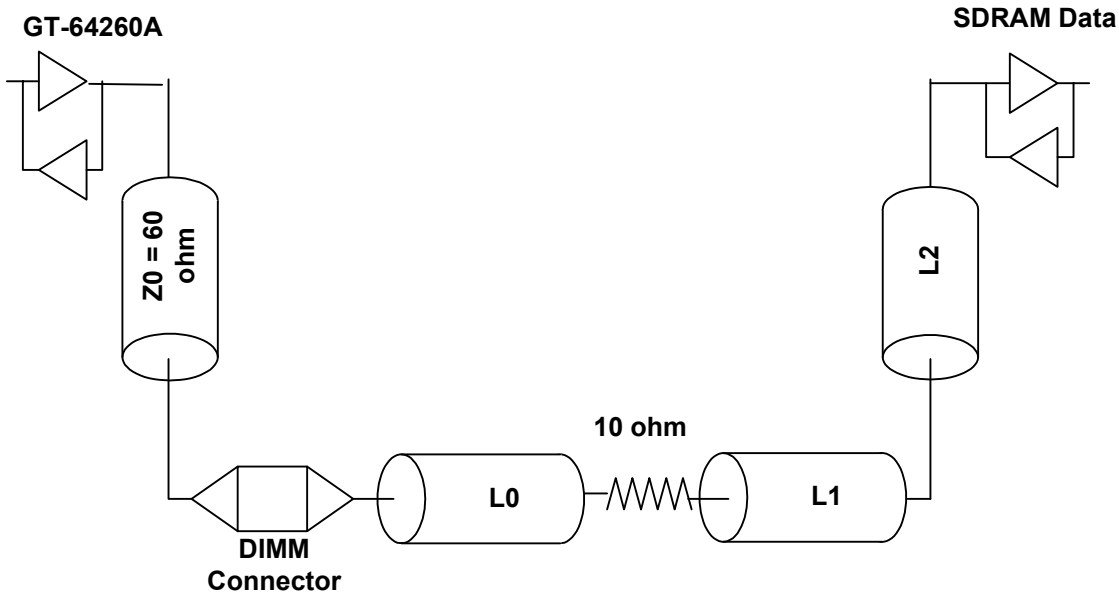


Table 19: Trace Length for Data Topologies

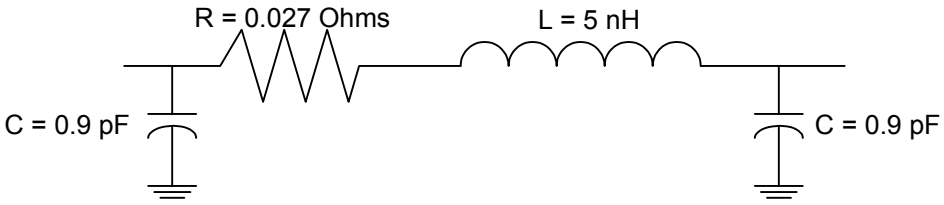
Comp Width	#of Loads	Zone	L0		L1		L2		Total Min	Total Max
			Min	Max	Min	Max	Min	Max		
All	1/2	I	0.10	0.80	0.10	0.80	0.05	0.15	0.9	1
All	1/2	II	0.10	1.00	0.10	1.00	0.05	0.15	1.0	1.4

Notes

- All distances are given in inches.
- Total Min and Total Max refer to the minimum and maximum respectively of L0 + L1 + L2. Also, the total minimum and maximum limits are tighter than the sum of the individual minimum and maximum lengths. This implies that not all individual segment lengths may be adjusted to the minimum and maximum value, respectively, at the same time.

Figure 66 shows the DIMM connector model.

Figure 66: DIMM Connector Package Model



The tables below show the required data bus AC timing for the GT-64260A SDRAM interface and SDRAM devices.

Table 20: GT-64260A SDRAM Interface AC Timing

Parameter	Value		Unit
	Min	Max	
Input Setup	1.3		ns
Input hold	0.4		ns
output delay	1	3.8	ns

Table 21: Typical SDRAM Interface AC Timing

Parameter	Value		Unit
	Min	Max	
Input Setup	1.5		ns
Input hold	0.8		ns
output delay	1	5.4	ns

GT-64260A to SDRAM data bus timing calculations:

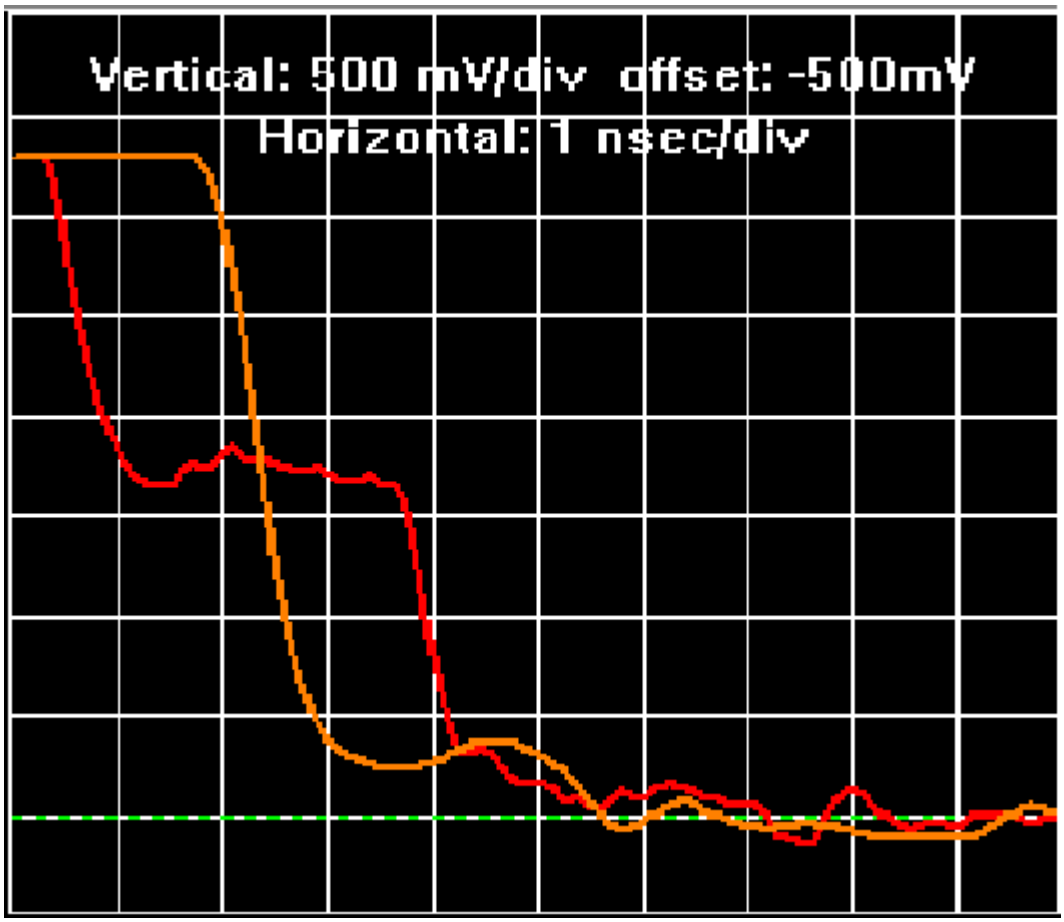
Tcycle > Toutput_delay(GT-64260A) + Tsetup(SDRAM) + Tdelay(fly_time) + Tclock_skew

7.5 > 3.8 + 1.5 + Tdelay(fly_time) + 0.5

Tdelay(fly_time) < 1.7 ns

Figure 67 shows a simulation for a 0.8 ns delay trace. The fly time is measured from the GT-64260A reference point determined in Figure 63 on page 112 (2.1 ns) to the Vil measured on the SDRAM pin (2.5 ns) in the figure below (board simulation).

Figure 67: 0.8 ns Delay Trace Simulation (2.1 ns Fly Time Reference Point)

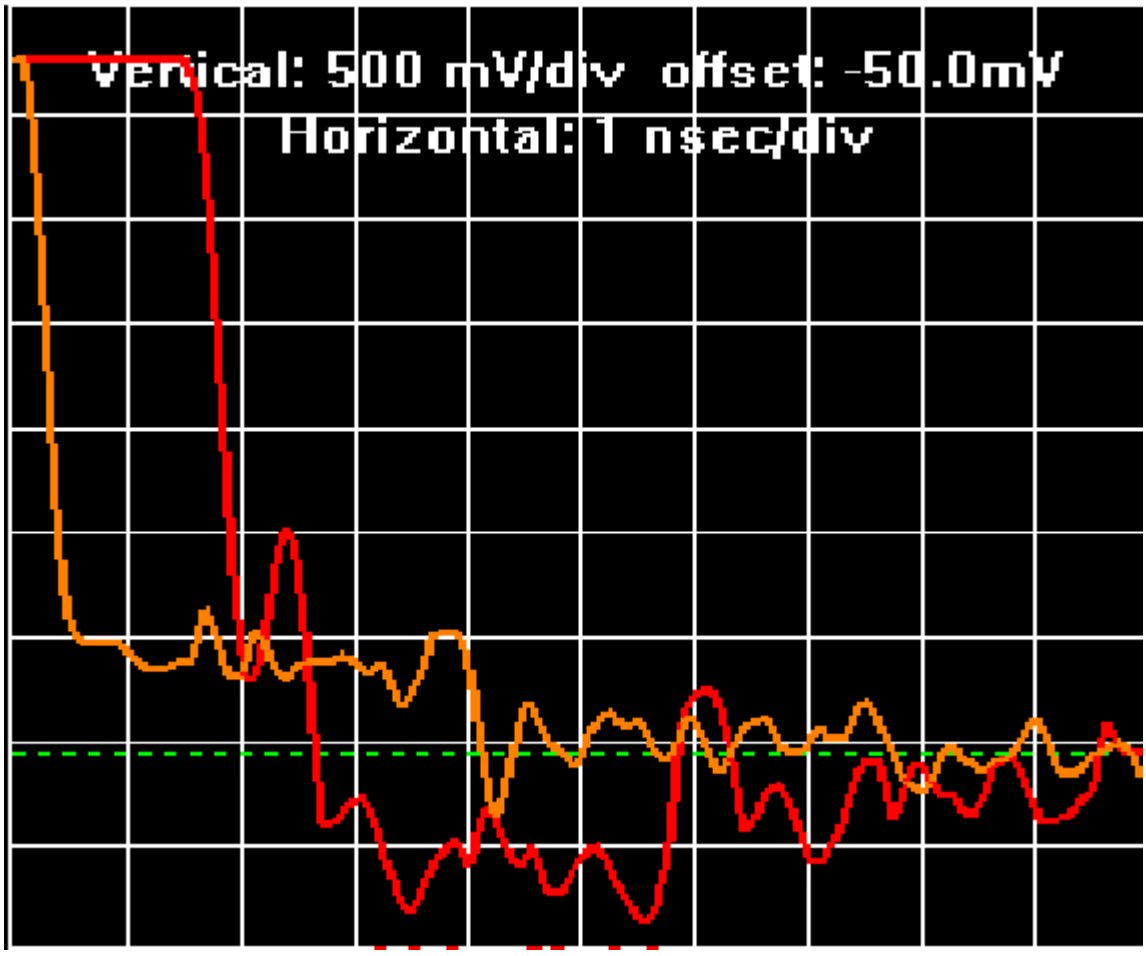


Note

The data signal fall time is smaller than the test circuit since the load is smaller (8 pf instead of 30 pf).

Figure 68 shows a simulation of 0.8 ns delay trace. The fly time is measured from the SDRAM reference point that was measured in Figure 64 on page 113 (1.54 ns) to the Vil measured on the SDRAM pin (2.5 ns) in Figure 68 (board simulation).

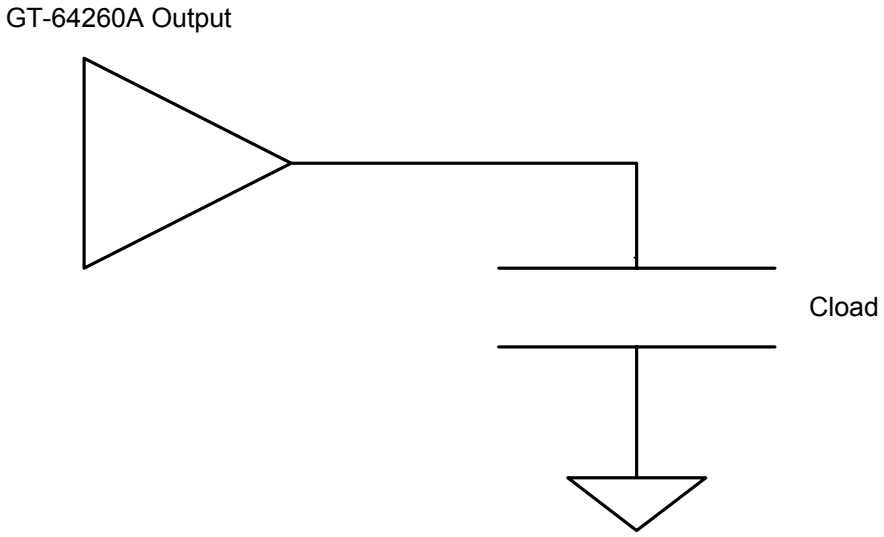
Figure 68: 0.8 ns Delay Trace Simulation (1.54 ns Fly Time Reference Point)



15.4.3 Chip Select Signals

The output delay value of the GT-64260A SCS*[3:0] signals in the AC timings table are given for 50 pf load (Cl = 50pf). (See Figure 69.)

Figure 69: GT-64260A Test Circuit (Cload = 50pf)



Simulating the GT-64260A SDRAM interface chip select signals test circuit gives the reference point of 1.3 ns. (See Figure 70.)

Figure 70: GT-64260A Chip Select Reference Point

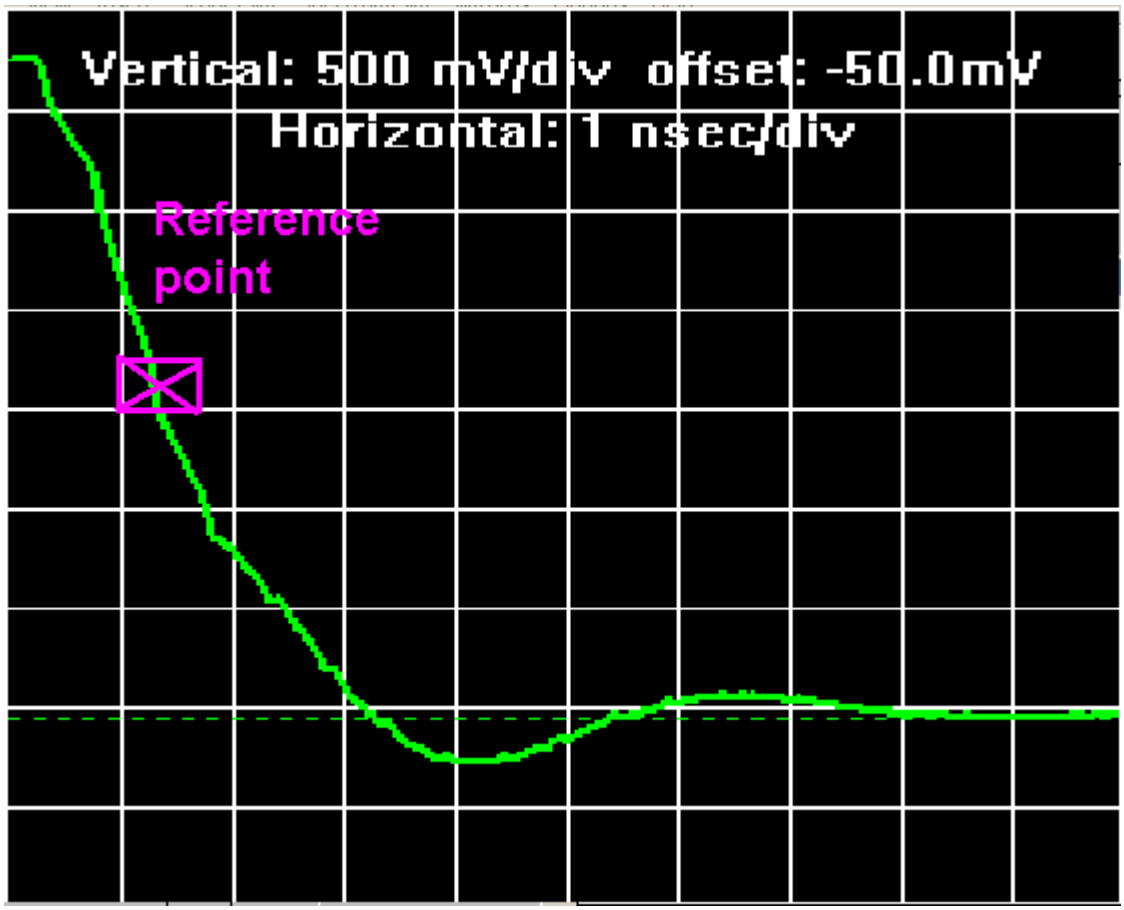


Figure 71 describes the Chip Select signal routing topologies on the DIMM module. Each GT-64260A SCS* signal is connected to two DIMM CS* pins (32-bits data width per SCS*). This means that in the sample memory configuration (one physical bank with eight devices), the SCS* signal is connected to eight SDRAM devices.

Figure 71: Chip Select Signal Routing on the DIMM Module

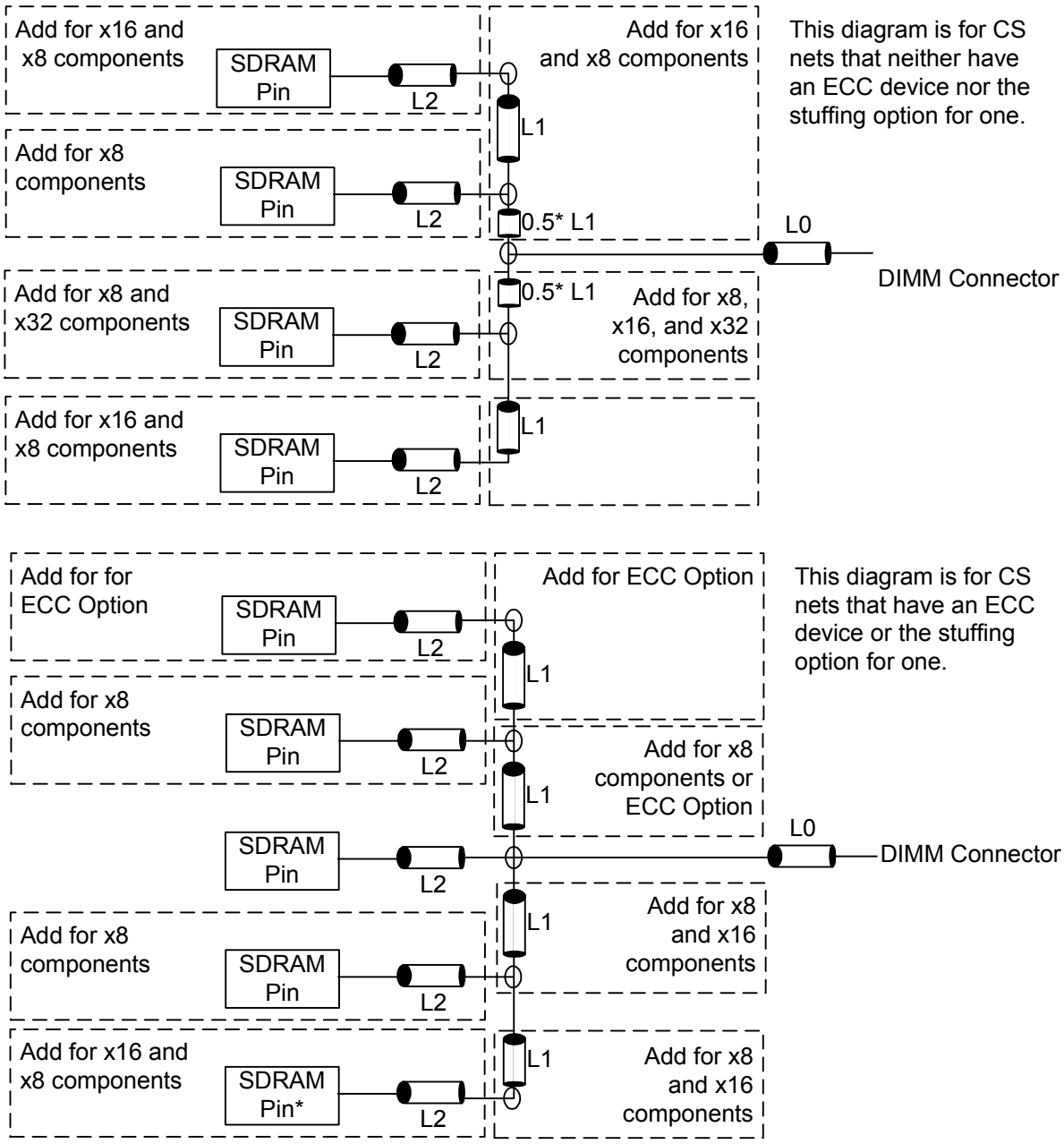


Table 22: Trace Length for Data Topologies

Comp Width	#of Loads	L0		L1		L2	
		Min	Max	Min	Max	Min	Max
x32	1/2	TBD	TBD	TBD	TBD	TBD	TBD
x18	2/3	1.30	1.65	0.50	0.60	0.06	0.08
x8	4/5	1.30	1.65	0.50	0.60	0.06	0.08

Table 23 shows the required AC timing for the GT-64260A SDRAM interface and SDRAM devices data bus.

Table 23: GT-64260A CS AC Timing

Parameter	Value		Unit
	Min	Max	
Input Setup	Output only.		ns
Input hold			ns
output delay	1	3.8	ns

Table 24: Typical SDRAM CS AC Timing

Parameter	Value		Unit
	Min	Max	
Input Setup	1.5		ns
Input hold	0.8		ns
output delay	Input Only		ns

GT-64260A to SDRAM chip select timing calculations:

Tcycle > Toutput_delay(GT-64260A) + Tsetup(SDRAM) + Tdelay(fly_time) + Tclock_skew

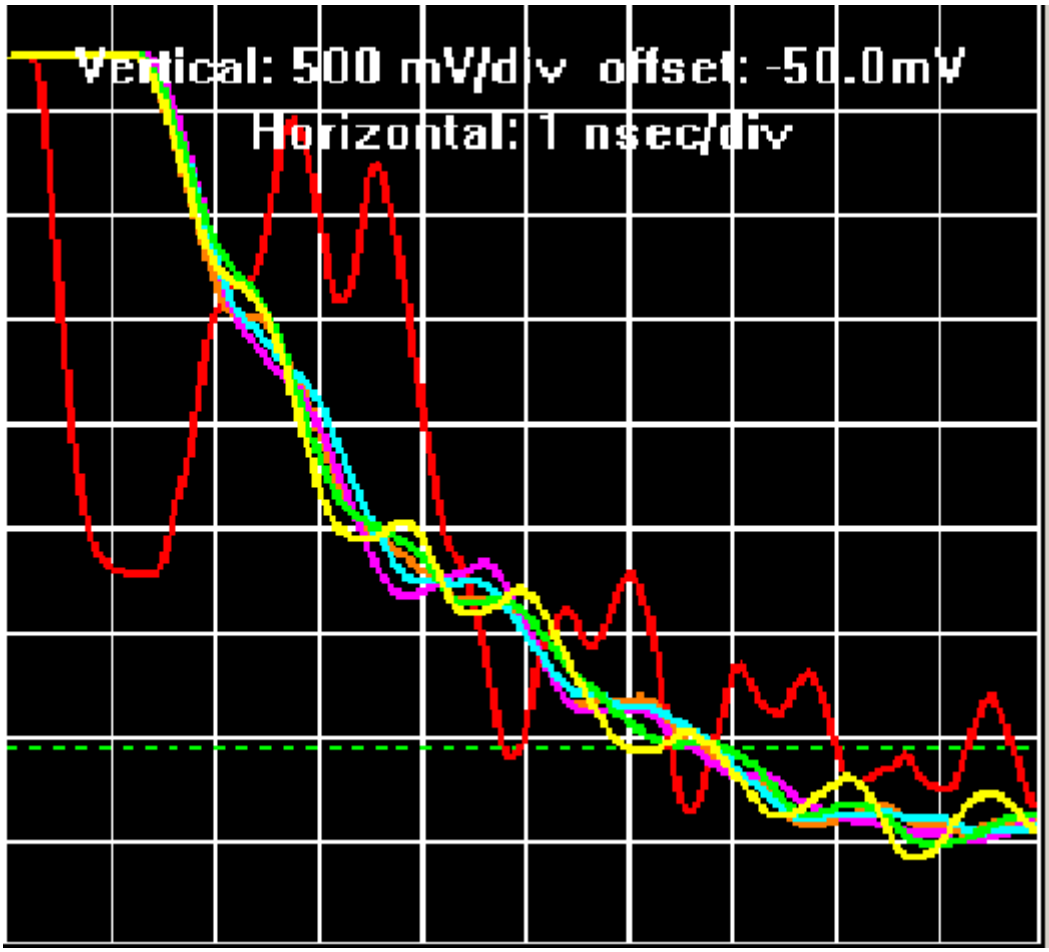
7.5 > 3.8 + 1.5 + Tdelay(fly_time) + 0.5

Tdelay(fly_time) < 1.7 ns

Figure 72 shows the simulation of 0.8 ns delay trace. The fly time is measured from the GT-64260A reference point that was measured in Figure 70 on page 119 (1.3 ns) to the Vil measured on the SDRAM pin (4.1 ns) in the figure below (board simulation).

Simulating this topology will give the following results. The calculated fly time is 4.1 - 1.3 = 2.8 ns.

Figure 72: 0.8 ns Delay Trace Simulation (2.8 ns Fly Time Reference Point)



15.4.4 Double Cycle Signals

The output delay value of the GT-64260A DAdr[12:0], SRAS*, SCAS*, and BankSel[1:0] signals in AC timings table are given for 50 pf load (Cl = 50pf). (See Figure 69.)

Simulating the GT-64260A SDRAM interface double cycle signal test circuit gives a reference point of 2 ns. (See Figure 73.)

Figure 73: GT-64260A Double Cycle Signals AC Timing

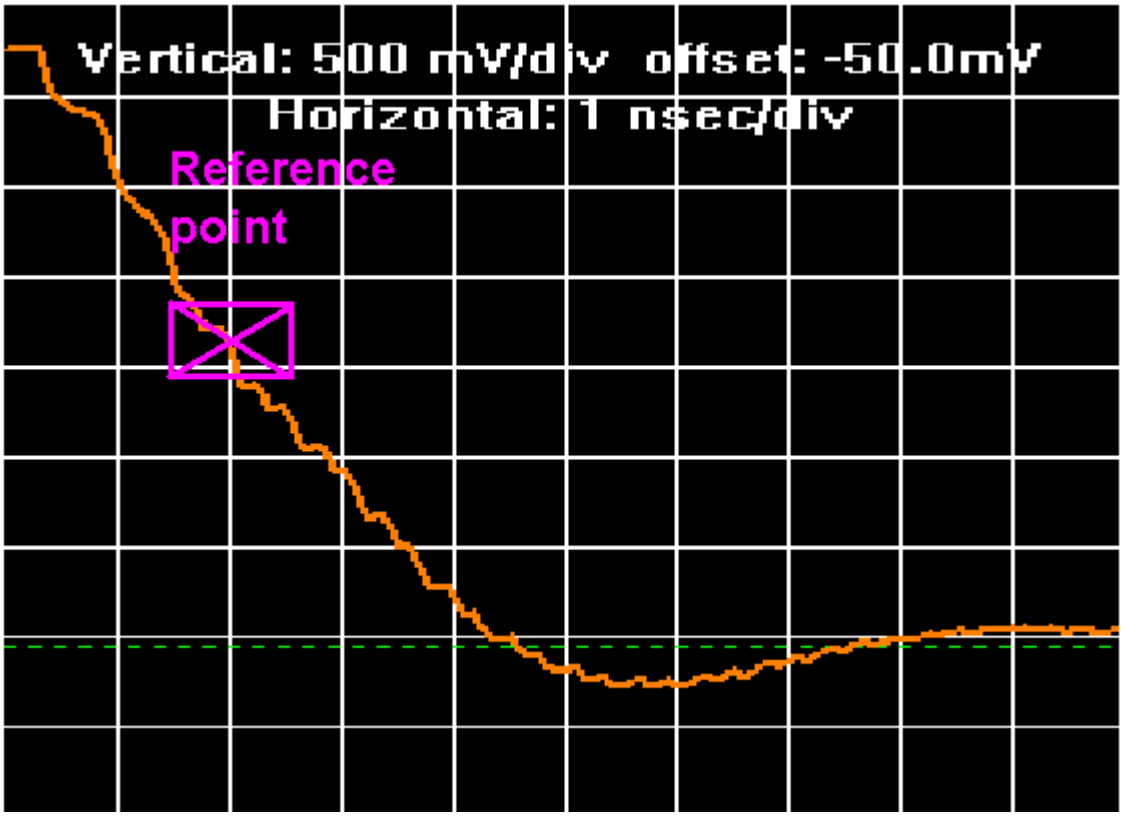


Figure 74 describes the double cycle signal routing topologies on the DIMM module. In the selected memory configuration, there are eight SDRAM devices placed on the DIMM (1 row DIMM).

Figure 74: Double Cycle Signal Routing on the DIMM Module

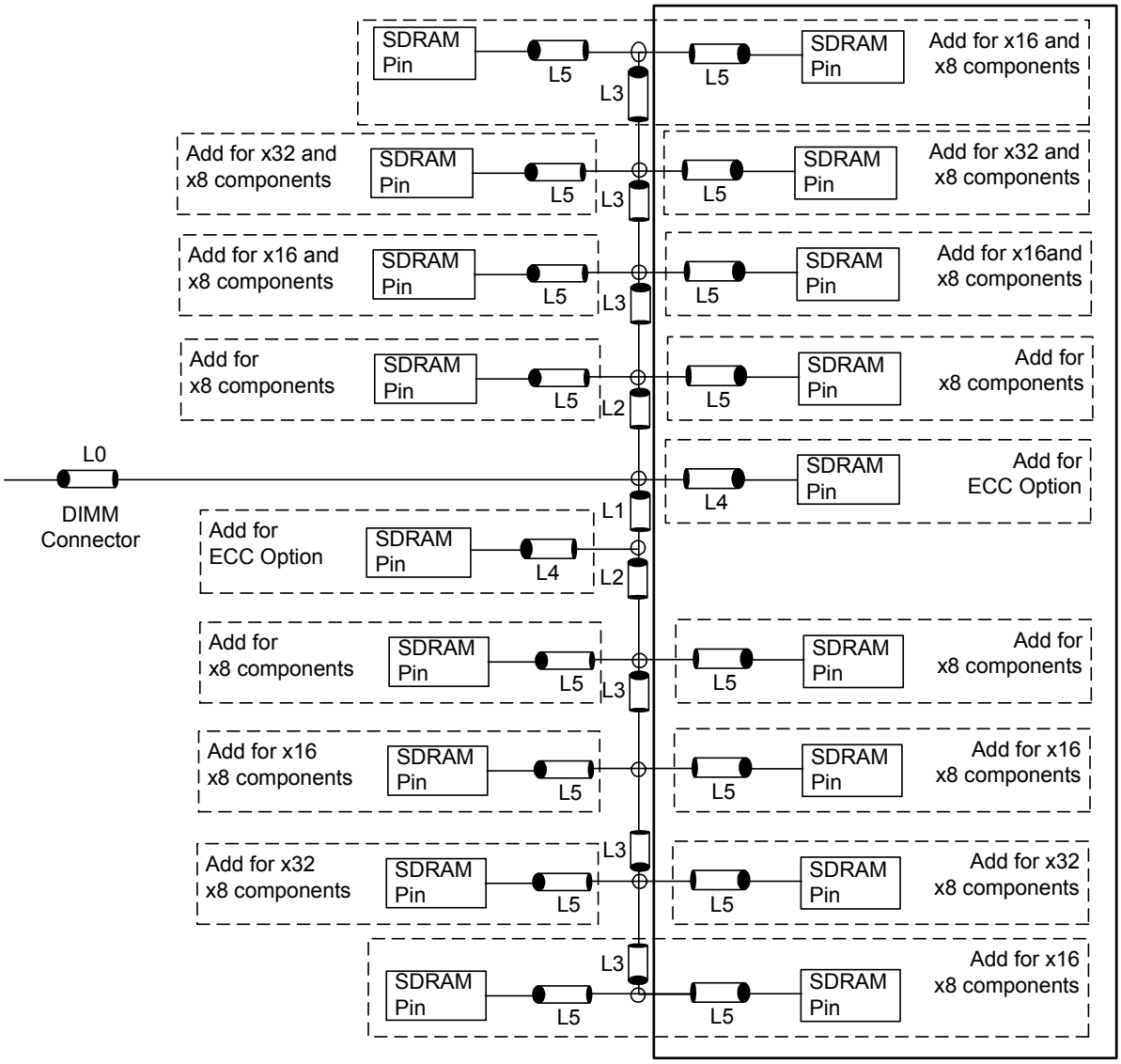


Table 25: Trace Length for Double Cycle Signal Topologies

Comp Width	#of Loads	L0		L1		L2		L3		L4		L5	
		Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
x32	2/3/4/6	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD	TBD

Table 25: Trace Length for Double Cycle Signal Topologies (Continued)

Comp Width	#of Loads	L0		L1		L2		L3		L4		L5	
		Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
x18	4/5/8/10	1.00	1.60	0.20	0.30	0.20	0.55	0.40	0.70	0.05	0.18	0.07	0.35
x8	8/9/16/18	1.00	1.60	0.20	0.30	0.20	0.55	0.40	0.70	0.05	0.18	0.07	0.35

The tables below show the required double cycle signal AC timing for the GT-64260A SDRAM interface and SDRAM devices.

Table 26: GT-64260A Double Cycle Signals AC Timing

Parameter	Value		Unit
	Min	Max	
Input Setup	Output only.		ns
Input hold			ns
output delay	1	3.7	ns

Table 27: Typical SDRAM CS AC Timing

Parameter	Value		Unit
	Min	Max	
Input Setup	1.5		ns
Input hold	0.8		ns
output delay	Input Only		ns

The GT-64260A to SDRAM chip select timing calculations:

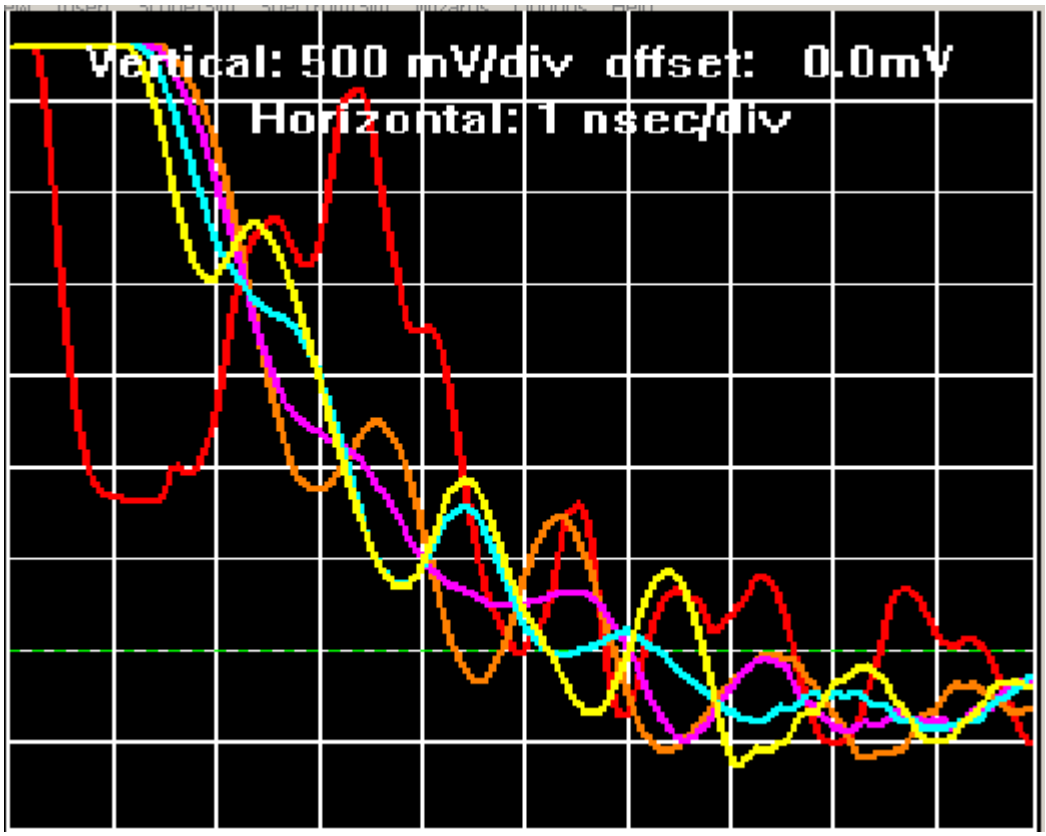
Tcycle > Toutput_delay(GT-64260A) + Tsetup(SDRAM) + Tdelay(fly_time) + Tclock_skew

7.5 > 3.7 + 1.5 + Tdelay(fly_time) + 0.5

Tdelay(fly_time) < 1.8 ns

Figure 75 shows a simulation of 0.8 ns delay trace. The fly time is measured from the GT-64260A reference point that was measured in Figure 73 on page 123 (2 ns) to the Vil measured on the SDRAM pin (3.8 ns) in the figure below (board simulation).

Figure 75: 0.8 ns Delay Trace Simulation (2.0 ns Fly Time Reference Point)



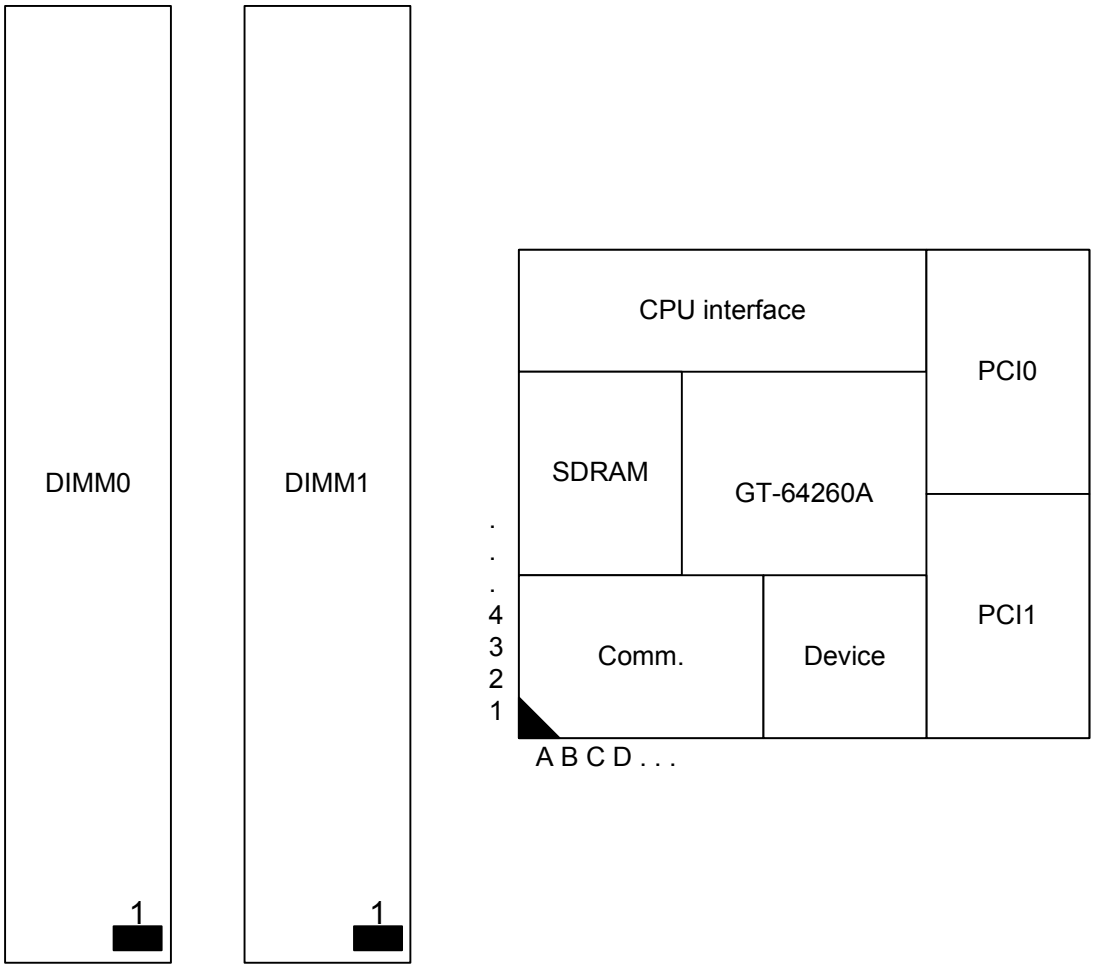
Simulating this topology means the calculated fly time is (3.8 - 2 = 1.8 ns).

15.5 Layout Instructions

15.5.1 Device Placement

All devices must be placed as closed as possible to each other. Figure 76 shows the device placement for systems using SDRAM DIMMs

Figure 76: Device Placement Example



15.5.2 Routing

The SDRAM interface traces must be 55 to 65 Ohm impedance.

The SDRAM and GT-64260A clocks (see Section 19. "Clocks" on page 143) must be routed on separate layers from the other signals.

Note

The point-to-multipoint topology signals should be routed in a V or T shape.

Section 16. PCI Interface Design Considerations

The PCI interface is used to connect the GT-64260A and other PCI devices. The GT-64260A supports two 64-bit PCI interfaces (named PCI0 and PCI1), compliant to PCI specification, Rev. 2.2.

16.1 Interface Connectivity

The PCI interfaces connectivity complies to the PCI specification, Rev. 2.2.

16.2 Electrical Definition

The PCI electrical definition provides for both 5V and 3.3V signaling environments.

Do not confuse the "signaling environments" with 5V and 3.3V component technologies. A "5V component" can be designed to work in a 3.3V signaling environment and vice versa. Component technologies can be mixed in either signaling environment. The signaling environments cannot be mixed; all components on a given PCI bus must use the same signaling convention of 5V or 3.3V.

For more information on the PCI interface electrical definition, see the "Electrical Specification" section in the PCI specification.

16.3 Termination Topology

In configurations where non-terminated trace ends propagate a significant distance from a PCI component (e.g., a section of unpopulated PCI connectors), it may be necessary to add active (e.g., diode) termination at the unloaded end of the bus, to insure adequate signal quality.

Note

Since the signaling protocol depends on the initial reflection, passive termination does not work.

16.4 Timing Requirements

Use Table 28 and Table 29 to determine the required AC timings for PCI specifications and GT-64260A PCI interfaces.

Note

The AC timing might be updated in the PCI specification or in the GT-64260A datasheet. Make sure you have the most update documents. For every conflict between this document and the specifications

document, the specifications documents will take precedence.This document provides additional information, but does not replace the specification documents.

Table 28: PCI AC Timing for 33 MHz and 66 MHz (From the PCI Specification Document, Rev. 2.2)

Symbol	Parameter	66 MHz		33 MHz		Units
		Min	Max	Min	Max	
T_{val}	CLK to Signal Valid Delay - bused signals	2	6	2	11	ns
$T_{val\ ptp}$	CLK to Signal Valid Delay - point to point signals	2	6	2	12	ns
T_{on}	Float to active delay	2		2		ns
T_{off}	Active to float delay		14		28	ns
T_{su}	Input setup time to CLK - bused signals	3		7		ns
$T_{su\ ptp}$	Input setup time to CLK - point to point signals	5		10,12		ns
T_{h}	Input hold time from CLK	0		0		ns
T_{rst}	Reset active time after power stable	1		1		ns
$T_{rst-clk}$	Reset active time after CLK stable	100		100		ns
$T_{rst-off}$	Reset active to output float delay		40		40	ns
T_{rrsu}	Req64# to RST# setup time	$10T_{gyc}$		$10T_{gyc}$		
T_{rrh}	RST# to Req64# hold time	0	50	0	50	
T_{rhfa}	RST# high to first configuration access	2^{25}		2^{25}		
T_{rhff}	RST# high to first FRAME# assertion	5		5		

Table 29: GT-64260A PCI Interface AC Timing
NOTE: All PCI interface Output Delays, Setup, and Hold times are referred to the PClk rising edge.

Signals	Description	133MHz		Units	Loading
PClk0,PClk1	Frequency		66	MHz	
PClk0,PClk1	Clock Period	15	∞	ns	
Rst0*, Rst1*	Active	1		ms	
FRAME0/1*, IRDY0/1*, TRDY0/1*, STOP0/1*, IDSEL0/1, DEVSEL0/1* REQ640/1*,ACK640/1*, PAR640/1, PERR0/1*, PAD0/1[63:0], CBE0/ 1[7:0]*, PAR0/1, GNT0/1*	Setup	3		ns	

Table 29: GT-64260A PCI Interface AC Timing (Continued)
NOTE: All PCI interface Output Delays, Setup, and Hold times are referred to the PCIk rising edge.

Signals	Description	133MHz		Units	Loading
FRAME0/1*, IRDY0/1*, PAD0/1[63:0], TRDY0/1*, STOP0/1*, IDSEL0/1, PAR640/1, DEVSEL0/1* GNT0/1*, REQ640/1*, ACK640/1*,PAR0/1, PERR0/1*, CBE0/1[7:0]*	Hold	0		ns	
FRAME0/1*, TRDY0/1*, IRDY0/1* DEVSEL0/1*, PAD0/1[63:0], STOP0/1*, CBE0/1[7:0]*, REQ640/1*,ACK640/1*, REQ0/1*, PAR0/1 PERR0/1*, SERR0/1*, PAR640/1 **NOTE:** Output delays are measured as specified in PCI specification, Rev. 2.2, section 7.6.4.3	Output Delay	2	6	ns	

For more information on the PCI timing requirements, see the following sections in the PCI specification, Rev., 2.2:

- For 33 MHz bus frequency, Section "4.3.5 System Timing Budget".
- For 66 MHz bus frequency, Section "7.7.5 System Timing Budget".

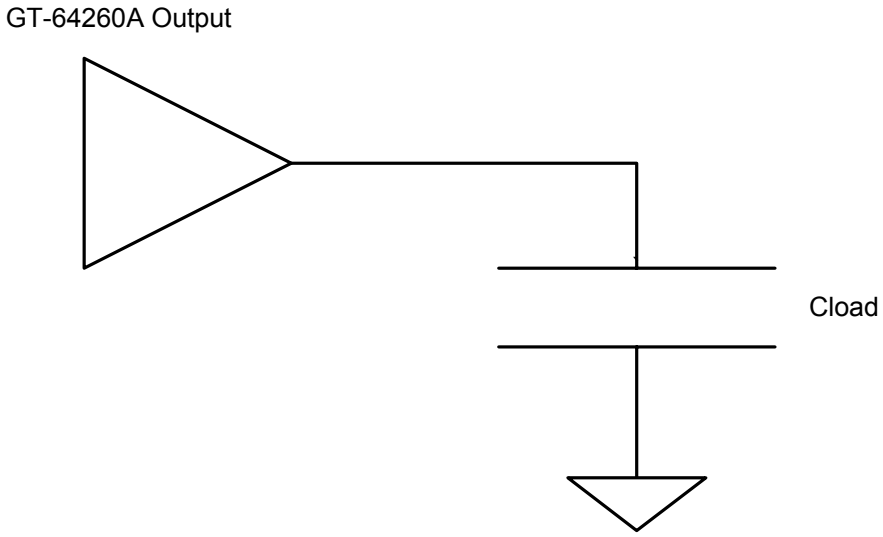
When using the GT-64260A internal PCI arbiter, the MPP interface is used for the arbitration signals. The MPP interface AC timing depends on their functionality. See the GT-64260A datasheet's "AC Timing" section. The PCI GNT* signals maximum output delay (referred to the corresponding PCIk clock) is 6.6 - 7.5 ns, depending on which pin is used.

Notes

- For more information on the MPP timing, see the GT-64260A secure web site or contact your local FAE).
- This timing violates the PCI AC specification for 66 MHz (see Table 28 on page 129). In this document, the timing calculation for the GNT* signals will assume maximum output delay of 7.5 ns.

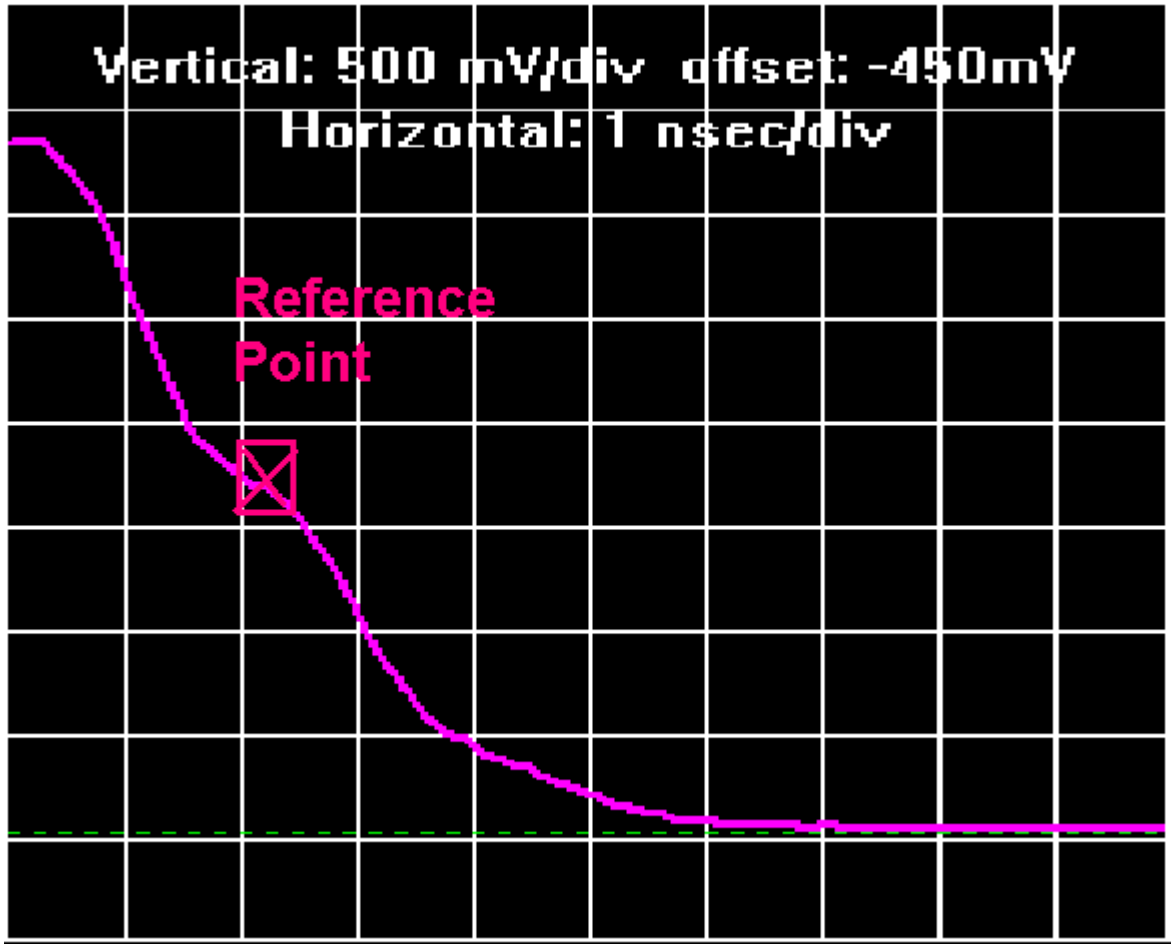
The GNT* signal output delay is measured in the following test circuit (Cload = 20pf). (See Figure 77.)

Figure 77: GT-64260A Test Circuit (Cload = 20pf)



Simulating the GT-64260A GNT* signal from the MPP interface test circuit will give a reference point of 2.1 ns. (See Figure 78.)

Figure 78: GT-64260A GNT* Signals Reference Point



GT-64260A to other PCI device GNT* timing calculations:

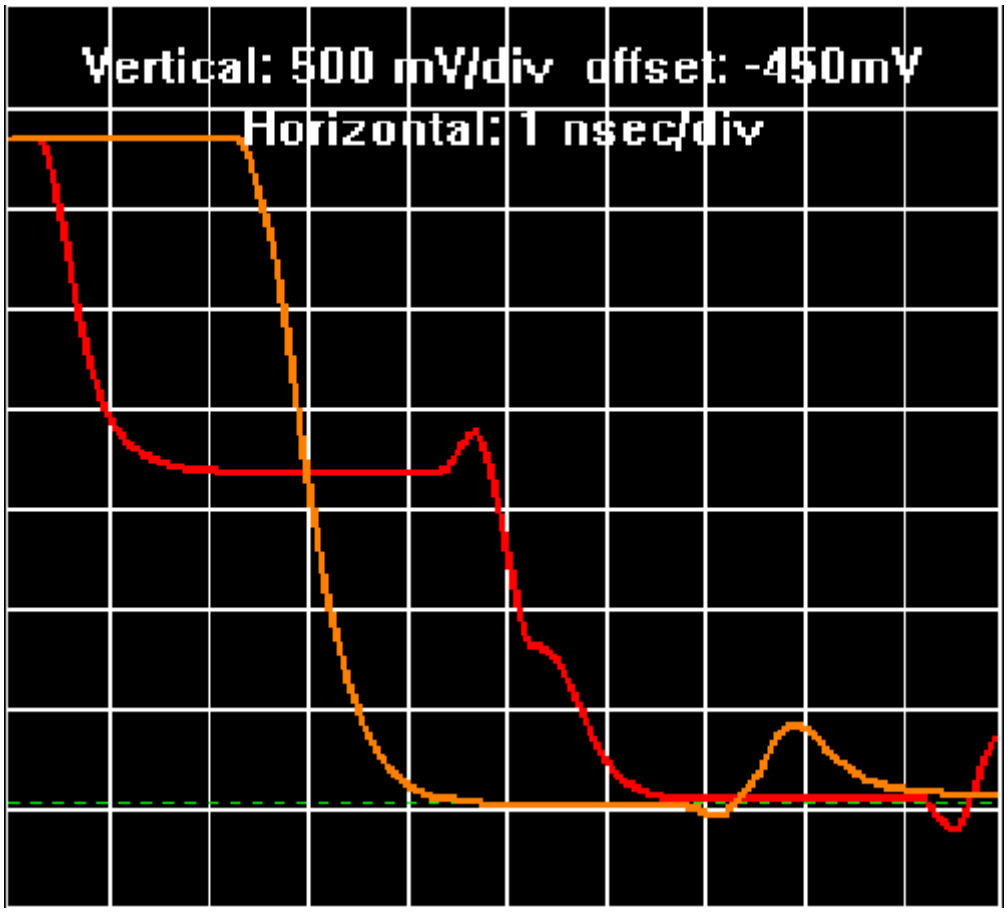
Tcycle > Toutput_delay(GT-64260A) + Tsetup(PCI_spec) + Tdelay(fly_time) + Tclock_skew

15 > 7.5 + 5 + Tdelay(fly_time) + 0.5

Tdelay(fly_time) < 2 ns

The fly time is measured from the GT-64260A reference point that is measured in Figure 78 (2.1 ns) to the Vil measured on the SDRAM pin (3.3 ns) in the figure below (board simulation).

Figure 79: 2.1 ns Fly Time Reference Point



Note

In this topology, the fall time is smaller than the test circuit since the load is smaller.

16.5 Layout Instructions

See Section "4.3.6. Physical Requirements" in the PCI Specification, Rev. 2.2.

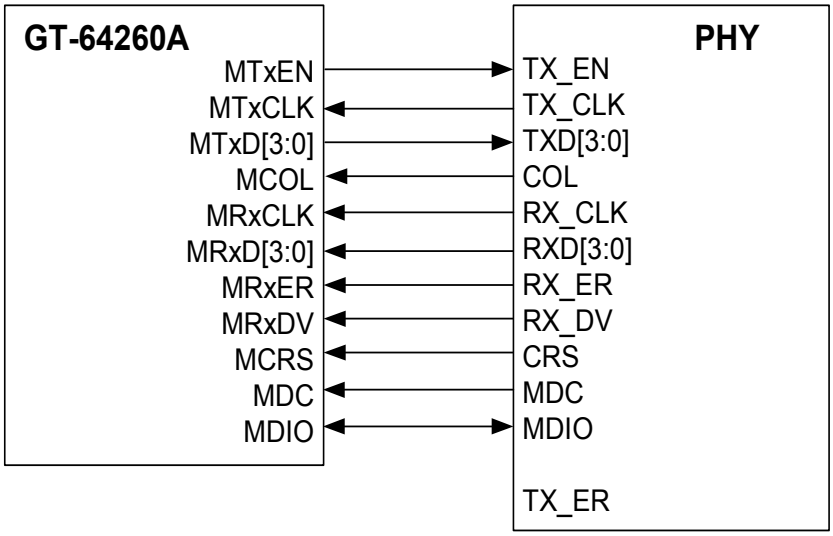
Section 17. Ethernet Interface Design Considerations

The GT-64260A contains up to three Ethernet controllers. Each controller can be configured to operate in MII or RMII mode.

Each 10/100 Mbit port is fully compliant with the IEEE 802.3 and 802.3u standards and integrates the MAC function and a dual speed MII interface.

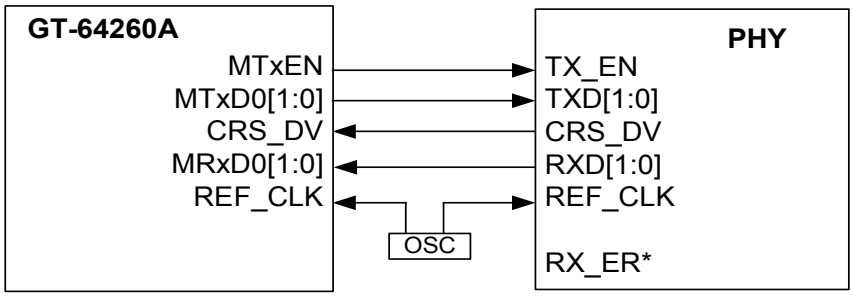
17.1 Interface Connectivity

Figure 80: MII Interface Connection



MDC/MDIO must be connected to all PHYs connected to a single pin of the GT-64260A.

Figure 81: RMII Interface Connection:



17.2 Electrical Specification

The TTL signal levels are compatible with devices operating at a nominal supply voltage of either 3.3V or 5.0V. To allow interfacing of 3.3V devices and 5.0V devices, all inputs can tolerate input potentials of 5.5V.

17.3 Termination Topology

The MII and RMII specifications do not require terminations. Nonetheless, it is recommended to simulate the board topology to make sure terminations are not needed.

Additionally, the MII/RMII interface I/O pads are very weak and, in most cases, do not need termination.

Note

If the electrical length of the circuit board traces used to implement these links exhibit transmission line effects, some form of termination may be required.

17.4 Timing Requirements

Table 30 and Table 31 shows the required AC timings for the RMII specifications and for the GT-64260A RMII interfaces.

Note

The AC timing might be updated in the RMII specification or in the GT-64260A datasheet. Make sure you have the most update documents. For every conflict between this document and the specifications

document, the specifications documents takes precedence. This documents provides additional information but does not replace the specification documents.

Table 30: RMII AC timing for 50 MHz (from RMII Specification Rev. 1.2 Document)

Symbol	Parameter	Min	Typ	Min	Units
	REF_CLK Frequency		50		MHz
	REF_CLK Duty Cycle	35		65	%
Tsu	TXD[1:0], TX_EN, RXD[1:0], CRS_DV, RX_ER Data setup to REF_CLK rising edge	4			ns
Thold	TXD[1:0], TX_EN, RXD[1:0], CRS_DV, RX_ER Data hold from REF_CLK rising edge	2			ns

Table 31: Ethernet RMII Interface

NOTE: All receive pins Setup, and Hold times refer to the RxClk rising edge. All transmit pins Output Delays, Setup, and Hold times refer to the TxClk rising edge.

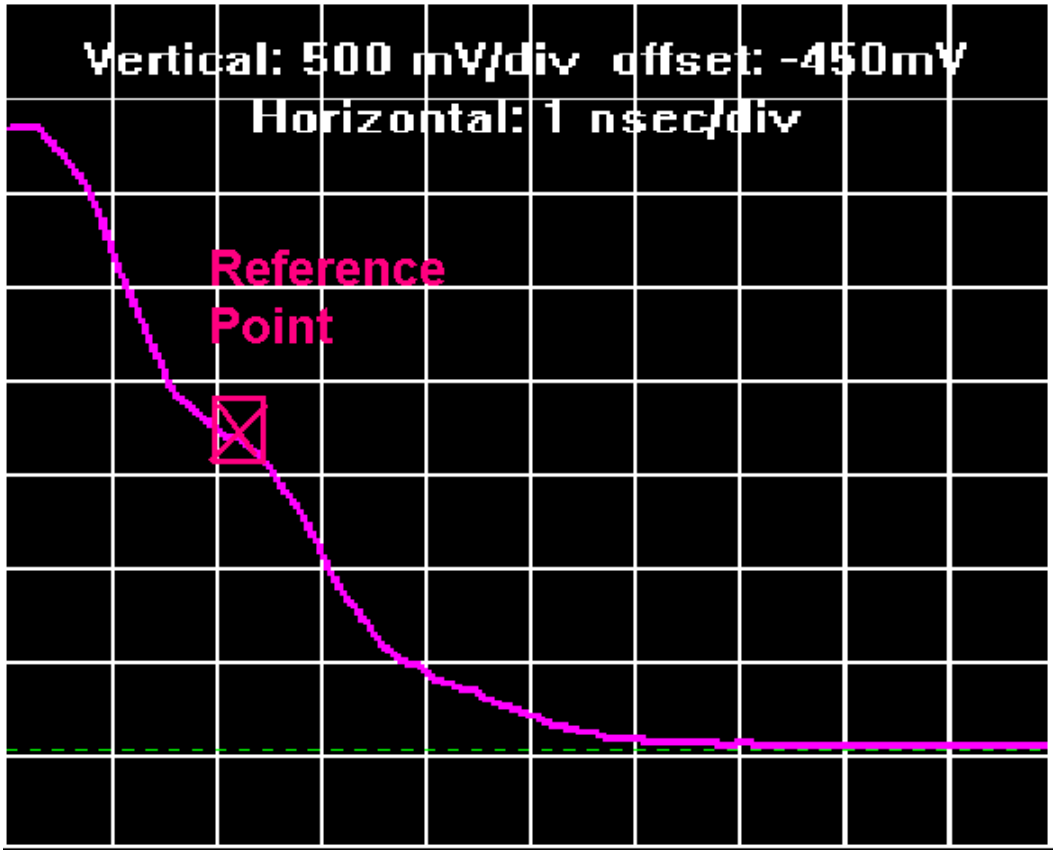
The multiplier in the following timing numbers is 1 for 100Mb/s operation, and 10 for 10Mb/s operation

Signals	Description	133MHz		Units	Loading
RMII Clock	Frequency		50	Mhz	
RMII Clock	Clock Period	20		ns	
DvCRS, RxD[1:0]	Setup	2		ns	
DvCRS, RxD[1:0]	Hold	1		ns	
TxD[1:0],TxEN	Output Delay	3	10	ns	20pf

The RMII signals output delay is measured using the test circuit in Figure 77 on page 131.

Simulating the GT-64260A RMII signals test circuit gives a reference point of 2.1 ns. (See Figure 82.)

Figure 82: GT-64260A RMII Signals Reference Point



GT-64260A to RMII PHY timing calculations:

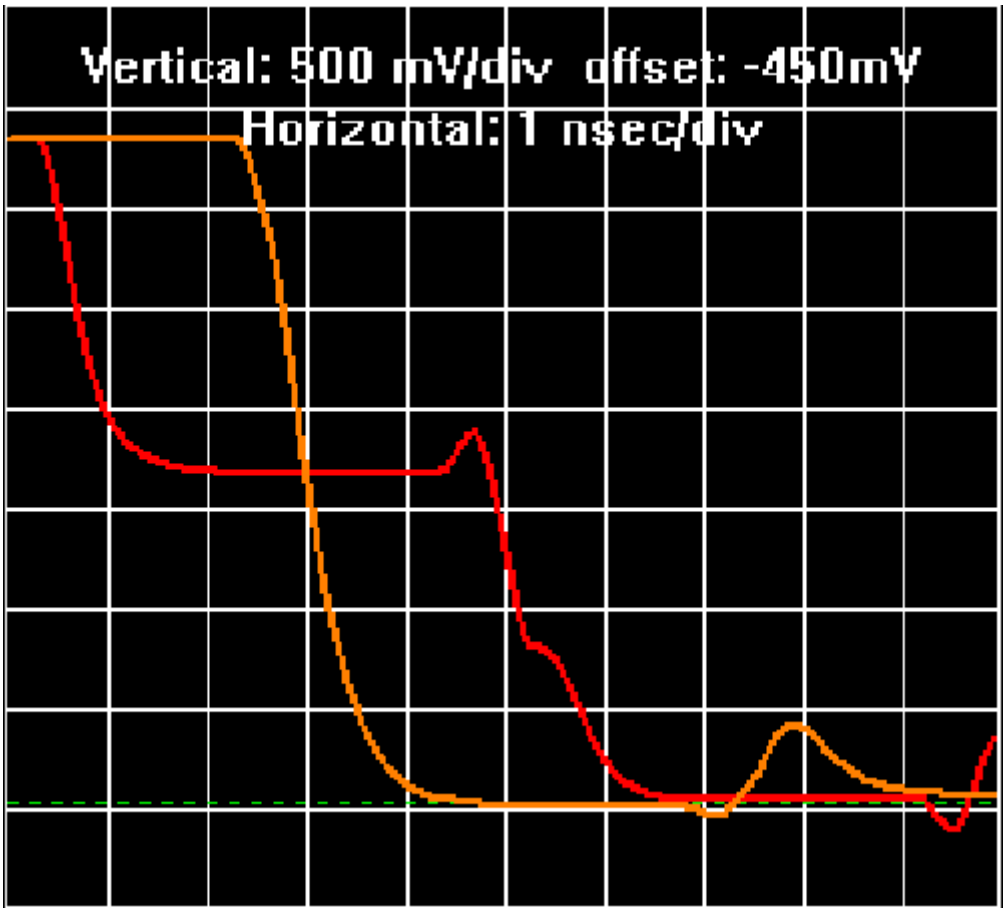
Tcycle > Toutput_delay(GT-64260A) + Tsetup(PHY) + Tdelay(fly_time) + Tclock_skew

20 > 10 + 4 + Tdelay(fly_time) + 0.5

Tdelay(fly_time) < 5.5 ns

The fly time is measured from the GT-64260A reference point in Figure 82 (2.1 ns) to the Vil measured on the SDRAM pin (3.3 ns) in the figure below (board simulation).

Figure 83: 2.1 ns Fly Time Reference Point



Note

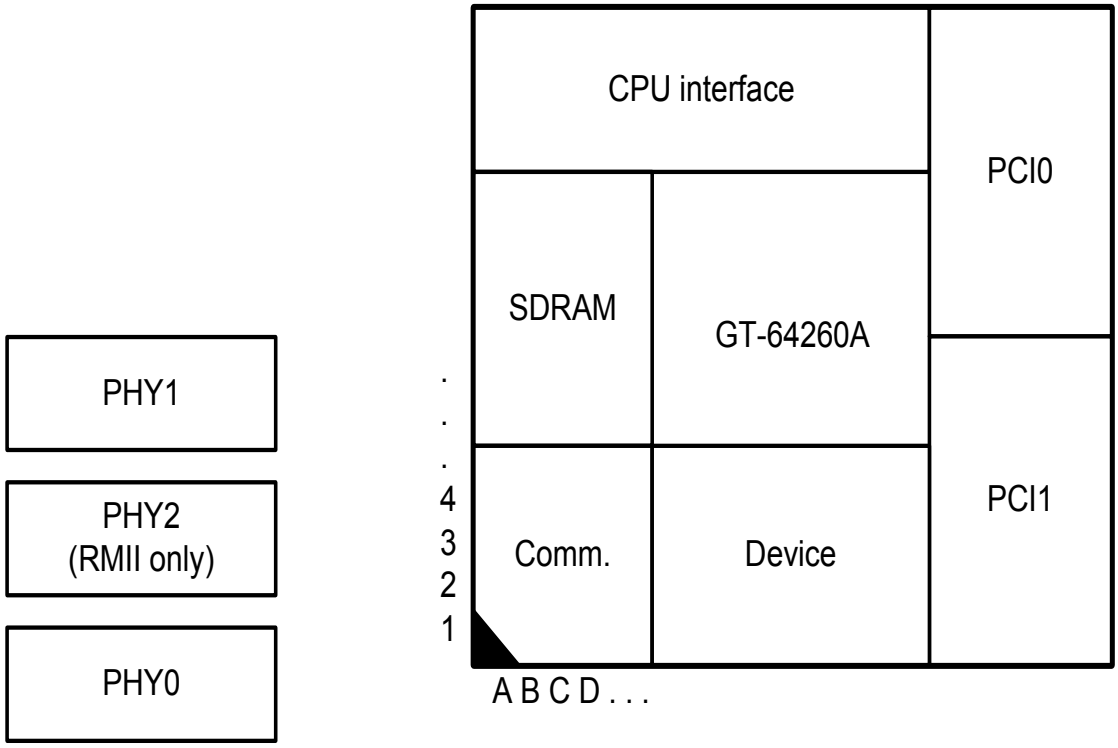
In this topology, the fall time is slower than the test circuit because the load is smaller.

17.5 Layout Instructions

17.5.1 Placement

Figure 84 illustrate how the PHYs must be placed.

Figure 84: PHY Placement



Note

The ports on the Packet Processor are arranged in the following order (from bottom to top) 0,2,1.

17.5.2 Routing

RMII traces must be 50 - 60 Ohm impedance.

MTx and MRx must be routed on separate layers.

In addition, each port must be separated from the other ports by at least 15 mm of clearance, when using 5 mm traces.

Section 18. Power Supply

The GT-64260A uses low-core voltages to attain higher operating speeds and lower power consumption.

It also divides the I/O power pins of various signals into logical groups. Each group may be set to different voltages. This allows the use of lower-voltage (faster) processor buses. This allows clamping PCI signals to 5V if required (dictated by the PCI bus), 5V tolerance for some interfaces, and 2.5V/3.3V IO voltage support for the CPU interface.

Table 32 describes the various GT-64260A voltages.

Table 32: GT-64260A Voltages

Power Group	Function	No. of Pins	Nominal Voltage	Notes
Vcc 2.5V	I/O Supply Voltage	10	3.3V or 2.5V	CPU interface
Vcc 3.3V	I/O Supply Voltage	33	3.3V	SDRAM, Communication, PCI and device interfaces
Vcc core	Core Supply Voltage	11	1.8V	
VREF	PCI Clamp Voltage	2	3.3V or 5V	
AVCC	PLL filtered power	1	1.8	Separate filters required
AGND	PLL filtered ground	1	GND	Separate filters required
GND	Ground, common	58	GND	

Note

For more information on the PLL power supply filtering, see the GT-64260A datasheet's "PLL Power Filter Circuit" section.

All power planes must be de-coupled to ground to provide for reasonable management of the switching currents (dI/dt) to which the plane and its supply path are subjected. This is platform dependent and not detailed in the specification in a manner consistent with high speed signaling techniques.

18.1 De-coupling Recommendations

Due to large address and data buses and high operating frequencies, the GT-64260A can generate transient power surges and high frequency noise in its power supply, especially while driving large capacitive loads. This noise must be prevented from reaching other components in the GT-64260A system.

To avoid noise affecting other components, ensure that there is sufficient capacitance in the power distribution system to supply the current and support voltage.

18.1.1 VCC3.3 De-coupling

Connect one 100 nf capacitor to each VCC3.3 power pin. The minimum capacitors/pin must not be smaller than 2:3.

Place the capacitors as close as possible to the power pins. It is also recommended to place two 4.7 uf capacitors close to the GT-64260A. See Figure 85.

18.1.2 VCC2.5 De-coupling

Connect one 100 nf capacitor to each VCC2.5 power pin. The minimum capacitors/pin must not be smaller than 1:2.

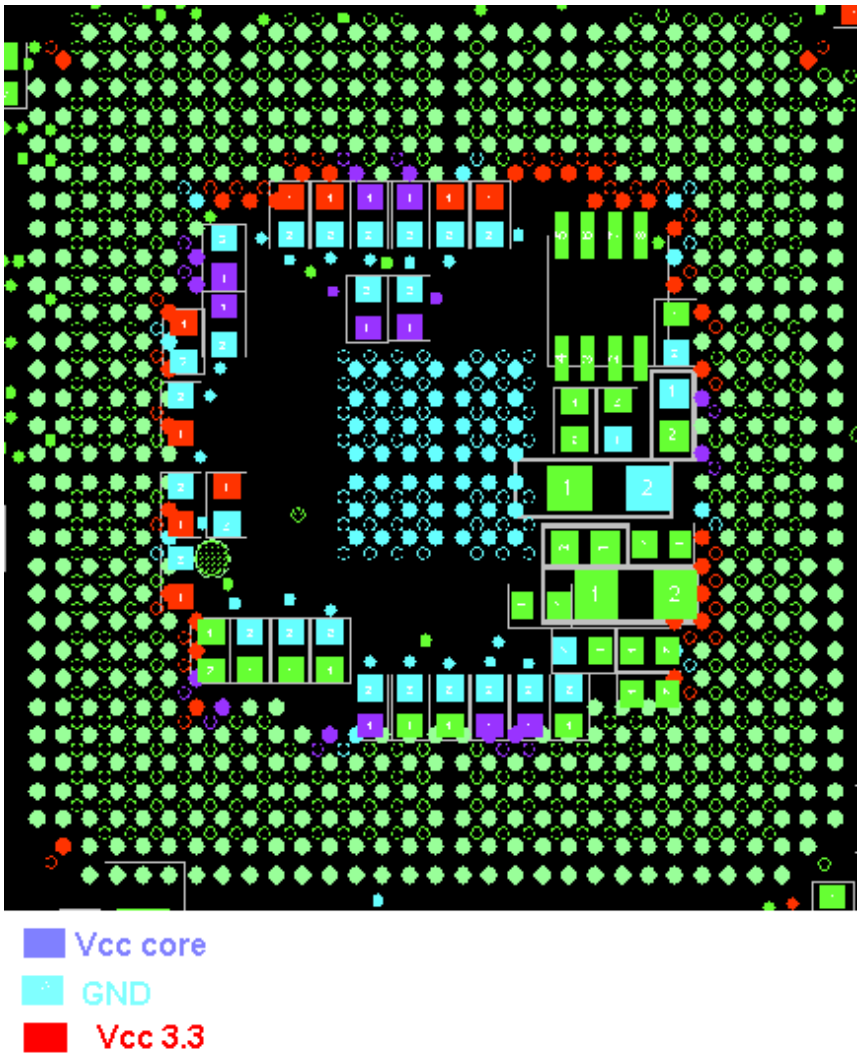
Place the capacitors as close as possible to the power pins. It is also recommended to place one 4.7 uf capacitor close to the GT-64260A. See Figure 85.

18.1.3 VCCcore De-coupling

Connect one 100nf capacitor to each VCC2.5 power pin. The minimum capacitors/pin must not be smaller than 1:2.

Place the capacitors as close as possible to the power pins. It is also recommended to place two 4.7 uf capacitors close to the GT-64260A. See Figure 85.

Figure 85: GT-64260A Power Supply Pin Map



Note

For Power sequencing information see *AN-67: Powering Up/Powering Down Galileo Technology Devices with Multiple Power Supplies of Different Voltages*.

Section 19. Clocks

The GT-64260A device implements several clock domains:

- TClk is the internal source clock. If AD[5] is pulled HIGH at reset de-assertion, TClk is also used as the clock source for the CPU interface.
- SysClk functionality is configured at reset. If AD[5] is pulled LOW at reset, SysClk is used as the clock source for the CPU interface, running asynchronously to TClk.
- PClk0 is the clock source for PCI0 interface.
- PClk1 is the clock source for PCI1 interface.
- SDClkIn/Out.

Note

For the SDRAM clocking scheme, it is not recommended to use the SDClkOut configuration. For more information on the SDRAM interface clocking, see the GT-64260A datasheet and *AN- 82: SDRAM Clocking Schemes in the GT-642xx/A Devices* on the secure website.

- BClkIn can be used as a clock input to the baud rate generator. It is multiplexed on the MPP interface. See Section 8. "Multi-Purpose Pin Interface Functional Overview" on page 74.
- BClkOut can be used as a clock output from the baud rate generator. It is multiplexed on the MPP interface. See Section 8. "Multi-Purpose Pin Interface Functional Overview" on page 74.

Notes

- For more information on the GT-64260A clocks, see the GT-64260A datasheet's "GT-64260A Clocking" section.
- The maximum allowed jitter on the TClk clock input is +/-100 pS.

Section 20. Reset

The GT-64260A implements three reset pins.

Note

Since the read and write pointers to the PCI interface unit FIFOs must be synchronized, it is necessary to assert the SysRst when the Rst signal is asserted. Some of the pointers are reset with the SysRst signal and some of them are reset with the Rst signal.

Some systems require the PCI devices to be reset, but not the GT-64260A and the CPU. For example, the CPU gets data from the GT-64260A communication ports and routes it to other GT-64260A interfaces (PCI, MPSC, etc.). Even if the PCI is in reset state, the system can still receive and send packets to the network. The system must notify the CPU that the PCI interface is currently not available. It then resets the PCI interface. To complete this reset sequence, the GT-64260A PCI reset signals must be separated from all other PCI devices and it must not be asserted when all of the PCI devices are being reset.

The recommended sequence for this example above is as follows:

1. In the PCI Status and Command register (Offset: 0x04 and 0x84), disable the IOEn bit [0], MasEn bit [2], by setting these bits to ’0’.
2. For the PCI slave, disable the same bits.
3. In the CPU Configuration register (Offset: 0x000), disable the GT-64260A synchronization barrier capabilities by setting ConfSBDis and IOSBDis bits [29:28] to ’1’.
4. Reset all the PCI devices, except for the GT-64260A. No configuration cycles are allowed prior to the PCI reset assertion.
5. After the PCI reset de-asserts the PCI master, the PCI slave and the synchronization barrier capabilities must be enabled.

20.1 Reset Configurations

The GT-64260A is configured at reset de-assertion via AD[31:0]. Additionally, it can be configured via Serial ROM initilization. For more information, see the GT-64260A datasheet’s "Reset Configuration" section.

Note

There cannot be any bus holds on the AD[31:0] bus. The GT-64260A must sample the configuration signals at a valid voltage level.

Section 21. Bringing Up the System (Debugging)

21.1 Communication unit

21.1.1 MPSC Packets Transmission Stops

The Shadow Byte Count filled (see "Tx Descriptors" on page 64) can be used to debug this problem. Each time a transmit descriptor is fetched, the Shadow Byte Count is loaded with the Byte Count field value. Each transmit burst decrements the Shadow Byte Count field value.

When the Shadow Byte Count field reaches zero, the descriptor is closed and the pointer moves to the next descriptor. For normal operation, the Shadow Byte Count field value must be initialized to the same value as the Byte Count field value.

For debugging purposes, the Shadow Byte Count field can be initialized to an unreasonable value (e.g 0xFFFF). If the Shadow Byte Count field remains with this value, it means that the descriptor has not been fetched. In addition, the Tx descriptor pointer and the transmit demand value must be checked to confirm that the packet was supposed to be transmitted.

The conclusion of such debugging is that the MPSC machine initilization is wrong.

Section 22. Revision History

Table 33: Revision History

Document Type	Revision	Date
Preliminary Revision	Rev. A	May 21, 2002

Appendix A. I2C EEPROM Example

The following is an I2C EEPROM example for serial ROM initialization on applications with the Motorola MPC745x CPU.

```
/* CPU Mode */
0x14000120 /* 0x40 - 64-bit PowerPC CPU, 60x bus */
0x00000040
/* CPU Configuration */
0x14000000
0x4000a8ff
0x14000160
0x00003035
0x14000d00
0x80000000
0x14000d80
0x80000001
/* 8-bit flash */
0x1400046c
0xf00b5e7c
/* 32-bit SRAM */
0x1400045c
0xf02051a9
0x14000460
0xf587d234
0x14000464
0xf0059bd4
0x14000468
0xf4a82f1c
0x14000448
0xd8E90200
0x1400044c
0x00004000
/* DMA to copy the 8 bit flash to the boot SRAM */
0x14000900
0x80080000 /* Byte count = 0.5MB */
0x14000910
0xfff00000   /* Source address */
0x14000920
```

```
0x1c000000 /* Destination address */
0x14000930
0x00000000 /* Next descriptor address = NULL */
0x14000940
0x80001a00
/* Dummy writes to stall the system until the DMA finishes */
0x14000c00
0x00000000
:
:
0x14000c00
0x00000000
/* Enable Boot from SRAM, switch between the memory windows */
0x14000238
0x000001c0
0x14000240
0x000001c7
0x14000028
0x00000fff
0x14000030
0x00000fff
/* Configure MPP 0-7, The InitAck signal is driven on MPP7 on the EV64260BP */
0x1400f000
0X50000000
/* The Serial ROM Last Data register contains the expected value of  two 0xffffffff).*/
0xffffffff
0xffffffff
```

Appendix B. SDRAM Mode Register/Code

```
/* configuration Sdram Operation Mode */
        lis  r5,  INTERNAL_BASE
        ori     r5,r5, SDRAM_MODE              ! r5 holds the Sdram operation Mode
        lis     r6, 0x0                                    ! r6 <= 0x0
        ori     r6, r6, 0x3                               ! r6 <= 0x3
        stwbrx  r6, 0, (r5)                              ! ( 0x14000474 ) <= 0x3
/* A dummy write to bank0 */
        sync
       lis     r6, 0x0                                     ! r6 <= 0x0
        stwbrx  r0, 0, (r6)                            ! ( 0x0 ) <= 0x0
        sync
/* A dummy write to bank1 */
          lis   r6, 0x0080                              ! r6 <= 0x0080.0000
        stwbrx  r0, 0, (r6)                            ! ( 0x80.0000 ) <= 0x0
/* A dummy write to bank2 */
        sync
        lis   r6, 0x0100                              ! r6 <= 0x0100.0000
        stwbrx  r0, 0, (r6)                           ! ( 0x180.0000 ) <= 0x0
 /* A dummy write to bank3 */
        sync
        lis   r6, 0x0180                              ! r6 <= 0x0180.0000
        stwbrx  r0, 0, (r6)                           ! ( 0x200.0000 ) <= 0x0
/* poll active bit */
       lis r6, 0x8000
        lis r5, r0, INTERNAL_BASE
        ori     r5,r5, SDRAM_MODE           ! r5 holds the Sdram operation Mode
active_poll_loop:
        lwbrx  r7, 0 , (r5)
       sync
      and r7, r7, r6
     cmp 0,0,r6,r7
     bne active_poll_loop

/* configuration Sdram Operation Mode */
         lis r5, r0, INTERNAL_BASE
        ori     r5,r5, SDRAM_MODE                    ! r5 holds the Sdram operation Mode
        lis    r6, 0x0                                          ! r6 <= 0
        stwbrx  r6, 0, (r5)                                 ! ( 0x14000474 ) <= 0x0
```

Appendix C. ECC Initialization- Example Code

C.1 Assembler Code

The following is the Assembler Code to Copy the Boot Device to SDRAM.

```
          ! Copy dink to memory....
    addisr3,0,0    ! Load dram address (0) into r3.
    addisr4,0,RESET_BASE! Load r4 with base eprom address
                      ! of 0xFFF00000.
    addisr6,0,0x10       ! Load r6 with upper half of 1M.
    ori r6,r6,0x0000! Load r6 with lower half of 1M.
        lis     r5,0x1400               ! DMA source address register
        ori     r5,r5,0x0910            ! loaded with flash base address
        stwbrx  r4,0,(r5)
        lis     r5,0x1400              ! DMA destination address register
        ori     r5,r5,0x0920            ! loaded with dram base address
        stwbrx  r3,0,(r5)
        lis     r5,0x1400               ! DMA ByteCount register
        ori     r5,r5,0x0900            ! loaded with block size
        stwbrx  r6,0,(r5)
        lis     r5,0x1400               ! DMA control register
        ori     r5,r5,0x0940

        lis     r3,0x8000                  ! DMA control register loaded
        ori     r3,r3,0x1a00            ! with block mode, 8bytes limit
        stwbrx  r3,0,(r5)
        lis     r4,0x0
        sync
    lis     r6,0x0
chanAct
    lis r4,0x0
        ori     r4,r4,0x4000               ! Is channel active?
    lwbrx   r6,0,(r5)
        and.   r4,r4,r6
    bnz     chanAct                ! Loop while DMA transfer continues
```

C.2 C Code Example

```
for (offset = 0x100000 ; offset < (SCS0SIZE+SCS1SIZE+SCS2SIZE+SCS3SIZE) ; offset+=0x100000)
    {
        dmaTransfer(DMA_ENG_4,offset,offset,0x100000,DTL_128BYTES |
BLOCK_TRANSFER_MODE,NULL);
        while (dmaIsChannelActive(DMA_ENG_4));
    }
```

Appendix D. Big and Little Endian Support

The GT-64260A core supports two Endianess settings, each with an 8 byte width.

Table 34: Big and Little Endian Bit Ordering

Little Endian the LSB is on the right.	7	6	5	4	3	2	1	0
Big Endian the LSB is on the left.	0	1	2	3	4	5	6	7

If the PCI Command register's (Offset: 0xc00. and 0xc80) MSwapEn bit [21] is set to '0', the swap settings are ignored. To use any kind of data swap, it must be set to '1'.

If the PCI is being accessed from the IDMA, start the correct swapping in the Channel Control High register (Offset: 0x880).

If accessing the PCI from the CPU, set the correct data swap in the corresponding PCI Low Decode Address register's PCISwap bits [26:24]. For example, if the PCI access hits the PCI0 Memory 0 window, set the data swap at the PCI_0 Memory 0 Low Decode Address Register (Offset: 0x058).

D.1 Internal Register

The GT-64260A internal register space is set to Little Endian. This means the byte order of significance has the Most Significant Byte (MSB) in the leftmost ordering and the Least Significant Byte (LSB) is the rightmost. The data written to an internal register is byte swapped by the GT-64260A CPU interface.

To write to the GT-64260A internal register, the data on the CPU bus must be driven as Big Endian (the order of significance is that the LSB is the leftmost and the MSB is the rightmost). One way of driving the data as Big Endian on the CPU bus is for the CPU's general register to hold the data in Big Endian and use a simple load instruction.

A C language programer can write data in this code as Big Endian or work with Little Endian and use a SW swap mechanism. This is the way the Low Level Drivers write to the GT-64260A internal register.

Another way to drive the data as Big Endian on the CPU bus is to use data in the code as Little Endian and only the store/load operation to the internal register will use the load/store word with byte swap instructions. In this way, the C programer has better code interface by using Little Endian data and better performance.

D.2 Communication Descriptors

The GT-64260A does not swap the data on CPU to SDRAM accesses (CPU data = SDRAM data). This also applies to the SDMA engines of the communication interfaces that take the data from the SDRAM.

This means that the data to descriptors in the SDRAM must be placed on the CPU bus as Little Endian but word swapped.

An example is the following descriptor:

```
address 0x0
              command/status
 address 0x4
              buffer size/byte count
address 0x8
              buffer pointer
 address 0xC
              next descriptor pointer
must be written to memory on the CPU bus as followed:
   address 0x0
              buffer size/byte count
 address 0x4
              command/status
 address 0x8
              next descriptor pointer
 address 0xC
              buffer pointer
```

Note

All the data in the descriptor must be written on the CPU bus as Little Endian. The word swap is done only by the software. There is no hardware support for this swap. The programer must switch the polarity of bit 2 of the address (0x0 <=> 0x4).

D.3 PCI Interface

The PCI interface only supports Little Endianess.

The problem with working in Big Endian is that the PCI is a 32-bit bus. Even with the extension to PCI64, every transaction ends as a 32-bit transaction depending on the master and slave.

Big Endianess depends on the bus width. Because of that, the GT-64260A must work on the PCI in Big-32 mode.

Table 35: PCI Big Endian Bit Ordering

Big Endian on a 32bit bus width.	4	5	6	7	-	0	1	2	3

If it is necessary to work with the PCI with Big-64, all of the transactions will have 8 byte width.

D.4 Swapping Options

Data swapping is made in the Lunit slave master.

There are two options for swapping. When these options are combined, they produce four cases. (See Table 36.)

Table 36: Data Swapping
NOTE: Data Before Swapping 0x 01020304.05060708

Option	Swap Mode	Data After Swapping	Word Bit	Byte Bit
1	No Swap	0x 01020304.05060708	0	1
2	Byte Swap	0x 08070605.04030201	0	0
3	Word Swap	0x 05060708.01020304	1	1
4	Byte and Word Swap	0x 04030201.08070605	1	0

D.4.1 Non PCI-to-PCI Swapping (CPU, DMA, SDMA)

Because the GT-64260A works in Dwords (8 Byte) and the PCI works in Words(4 Byte), a restriction exists when working with less than Dword transactions.

There are four reasonable cases when working with Big and Little Endian on the core and on the PCI.

Core Little: Option 1, 4 (2, 3 are Restricted on non-Dword Transactions)

- PCI Little swap mode - No Swap (1).
- PCI Big swap mode - Byte and Word Swap (4).

Core Big: Option 2, 3, 5, 8 (1, 4 are Restricted on non-Dword Transactions)

- PCI Little swap mode - Byte swap (2).
- PCI Big swap mode - Word Swap. (3).

D.4.2 PCI-to-PCI Swapping (PCI0-to-PCI1, PCI1-to-PCI0)

Because both PCIs works in Words, a restriction exists when working with less than Dwords transactions.

If the PCIs work with the same Endian setting, the swap mode must be No Swap (1).

If the PCIs use different Endianess settings, then the mode must be Byte and Word Swap (4).

Notes

- Options 2 and 3 are restricted on non-Dword transactions.
- On P2P transactions, swapping is executed on the slave and the master.The swapping is the combination of both swapping.

D.4.3 Master Swapping

The Master's swapping setting is determined by the PCI Command registers (Offset: 0xc00 and 0xc80) MByteSwap bit [0] and MWordSwap [10] and the xbar command bit [14:12], which determine the initiating unit (PCI/CPU/DMA).

Table 37: Master Swapping

Swap	Xbar Command	PCI Command Register	Result
Byte	bit 12	bit 0	active low (0-swap)
Word	bit 13	bit 10	active high (1-swap)

Reg 0xC00 MSwapEn bit [21] PCI Master Swap Enable

- 0 - PCI master data swapping is determined via MByteSwap and MWordSwap bits (bits 0, 10 in 0xc00) as in the GT-64120/130
- 1 - PCI master data swapping is determine via PCISwap bits in CPU to PCI Address Decoding registers or PCISwap bits in PCI to PCI Address decoding registers. (On the xbar command, bits 14-12 determine the initiator unit PCI/CPU/DMA)

To use the PCI as a 64-bit bus (for Big64), the option of force64 is available to force the PCI master to assert the REQ64# signal. When asserting REQ64#, ensure that the slave always responds with ACK64#.

D.4.4 Slave Swapping

The slave performs data swapping determined by the PCI Command registers (Offset: 0xc00 and 0xc80) or according to swap bits in the access.

There are four cases:

- SSwapEn bit [20]) is set to '0'. The swapping is according to SWordSwap bit [11]) and SbyteSwap bit [16]).
- SswapEn bit [20]) is set to '1' and there is a hit in the access. The swapping is according to the PCISwap bits [26:24] in the access.
- SswapEn [20]) is set to '1' and there is no hit in the access. The swapping is according to SWordSwap bit [11] and SbyteSwap bit [16]).
- There is an access to internal registers or to configuration registers. The swapping is according to SintSwap bits [26:24]), regardless of any other configuration bits.

In the following example, the data 0x7766554433221100 is written, with all byte enabled valid (all '0'), from the PCI to the SDRAM.

```
Here is how it looks on a 64 bit PCI bus:
    In case of starting address with offset 0:
        Pad[31:0] = 0x33221100
        Pad[63:32] = 0x77665544
    In case of starting address with offset 4:
        First phase  -  Pad[31:0] = don't care
                              Pad[63:32] = 0x33221100
```

```
          Second phase  -  Pad[31:0] = 0x77665544
                        Pad[63:32] = don't care
Here is how it looks on a 32 bit PCI bus:
       In case of starting address with offset 0:
          First phase  - Pad[31:0] = 0x33221100
          Second phase  -  Pad[31:0] = 0x77665544
       In case of starting address with offset 4:
          First phase  -  Pad[31:0] = 0x33221100
          Second phase  -  Pad[31:0] = 0x77665544
```

Table 38 shows how the data looks on the SDRAM bus.

Table 38: Master Swapping on the SDRAM Bus

Swap Type	Starting Address with Offset 0	Starting Address with Offset 4
Byte Swap	Sdata[63:0] – 0x0011223344556677	Phase 1 - Sdata[63:0] – 0xxxxxxxxx00112233 Phase 2 - Sdata[63:0] – 0x44556677xxxxxxxx
Byte and Word Swap	Sdata[63:0] – 0x4455667700112233	Phase 1 – Sdata[63:0] – 0x00112233xxxxxxxx Phase 2 – Sdata[63:0]– 0xxxxxxxxx44556677
Word Swap	Sdata[63:0] – 0x3322110077665544	Phase 1 – Sdata[63:0] – 0xxxxxxxxx33221100 Phase 2 – Sdata[63:0] – 0x77665544xxxxxxxx
No Swap	Sdata[63:0] – 0x7766554433221100	Phase 1 – Sdata[63:0] – 0x33221100xxxxxxxx Phase 2 – Sdata[63:0] – 0xxxxxxxxx77665544

Table 39 shows the necessary swaps for all eight options.

Table 39: Swapping for All Eight Options

PCI Width	PCI Endianess	Core Endianess	Swap Needed
64	Little	Little	No swap
64	Little	Big	Byte swap
64	Big	Little	Byte swap
64	Big	Big	No swap

Table 39: Swapping for All Eight Options (Continued)

PCI Width	PCI Endianess	Core Endianess	Swap Needed
32	Little	Little	No swap
32	Little	Big	Byte swap
32	Big	Little	Byte & word swap
32	Big	Big	Word swap

Appendix E. Communication Example Code

E.1 Ethernet Initialization

```
    /* Initialize the Communication Unit memory pool */
    usrMemInit();
/* connect the MPSC and Ethernet as RMII ports */
    GT_REG_WRITE(SERIAL_PORT_MULTIPLEX, 0x1102);

    /* initialize the address table to filter other MAC addresses */
    initAddressTable(ETHERNET_DOWNLOADING_PORT,0,1,0);
    macH = (ifWanMacAddress[0] << 8) | (ifWanMacAddress[1]);
    macL = (ifWanMacAddress[5] << 0) | (ifWanMacAddress[4] << 8) |
           (ifWanMacAddress[3] << 16) | (ifWanMacAddress[2] << 24);

    addAddressTableEntry(ETHERNET_DOWNLOADING_PORT,macH,macL,1,0);
/* rx prio0 */
    allocStruct.bufferSize = BUFFER_SIZE_FOR_ETHERNET;
    allocStruct.numberOfDescriptors = MIN_NUMBER_OF_ETHERNET_RX_DESC_ALLOC_PRIO0;
    allocStruct.portNumber = ETHERNET_DOWNLOADING_PORT;
    allocStruct.priority = PRIO0;
    allocStruct.rxOrTx = RX_DESCRIPTOR;
    allocStruct.protocolType = ETHERNET_PROTOCOL;
    sdmaAllocateDescriptorsForOnePort(&allocStruct);
    sdmaInitRxDescriptors(ETHERNET_DOWNLOADING_PORT,ETHERNET_PROTOCOL,PRIO0);

    /* tx prio0 */
    allocStruct.bufferSize = 0;
    allocStruct.numberOfDescriptors = MIN_NUMBER_OF_ETHERNET_TX_DESC_ALLOC_PRIO0;
    allocStruct.portNumber = ETHERNET_DOWNLOADING_PORT;
    allocStruct.priority = PRIO0;
    allocStruct.protocolType = ETHERNET_PROTOCOL;
    allocStruct.rxOrTx = TX_DESCRIPTOR;
    sdmaAllocateDescriptorsForOnePort(&allocStruct);
    sdmaInitTxDescriptors(ETHERNET_DOWNLOADING_PORT,ETHERNET_PROTOCOL,PRIO0);

    /* Set the buffers as Big Endian (rx and tx)  */
    GT_REG_WRITE(ETHERNET0_SDMA_CONFIGURATION_REGISTER +
                 ETHERNET_PORTS_DIFFERENCE_OFFSETS*ETHERNET_DOWNLOADING_PORT,
```

```
                0x3000);

    /* alocates buffers to all the tx descriptors */
    allocBuffersForNicTxDescriptors(ETHERNET_DOWNLOADING_TX_PORT);

    /* override the rx priority to receive every thing in the lowest priority */
    etherGetDefaultPortConfig(&portConfig,ETHERNET_DOWNLOADING_PORT);
    portConfig.portext |= OVERRIDE_RX_PRIORITY;
    etherInitPort(&portConfig,ETHERNET_DOWNLOADING_PORT);
```

E.2 Ethernet API

ethernetLow.c: Ethernet register manipulation and MIB counter API + MII register manipulation (PHY).
ethernetLow.h: Ethernet function and structure declaration.
ethernet.c : Ethernet port initialization code.
sdmaLow.c: SDMA register manipulation and initializations.
sdmaLow.h: SDMA functions and structures declaration.
sdma.c: SDMA memory allocation and transmit packet routines.
sdma.h: SDMA structs and function declaration.

```
STATUS wanTxPacket (STRUCT_TX*  Transmit)
{
    int i;
    TX_DESC* pCurrTxDesc;
    TX_DESC* pLastTxDesc;
    TX_DESC* pFirstTxDesc;

    /* get the current tx descriptor from a table (in the cache) */
    pCurrTxDesc = (TX_DESC*)getCurrentTxDescriptor(ETHERNET_DOWNLOADING_TX_PORT);

    /* if the owner is the GT return ERROR to prevant destroy data */
    if(pCurrTxDesc->cmd_sts & OWNER_BY_GT)
    {
        return ERROR;
    }

    pFirstTxDesc = pCurrTxDesc;
    pLastTxDesc = pCurrTxDesc;
    *(UINT32*)(&(pFirstTxDesc->cmd_sts)) = FIRST | ENABLE_INTERRUPT | PADDING;
```

```
    for(i = 0 ; i <Transmit->bufferNum ; i++)
    {
        pCurrTxDesc->bytecnt = Transmit->bufferLen[i];
        pCurrTxDesc->buf_ptr = VIRTUAL_TO_PHY(Transmit->buffer[i]);
        pCurrTxDesc->shadowOwner = SHADOW_OWNER_BY_CPU;
        pCurrTxDesc->pointerToRxQueue = (UINT32)Transmit;

        if(i != 0)
            *(UINT32*)(&(pCurrTxDesc->cmd_sts)) = ENABLE_INTERRUPT | OWNER_BY_GT;
        pLastTxDesc = pCurrTxDesc;
        pCurrTxDesc = (TX_DESC*)PHY_TO_VIRTUAL(pCurrTxDesc->next_desc_ptr);
    }

    *(UINT32*)(&(pLastTxDesc->cmd_sts)) = *(UINT32*)(&(pLastTxDesc->cmd_sts)) | LAST |
OWNER_BY_GT | GENERATE_CRC;
    *(UINT32*)(&(pFirstTxDesc->cmd_sts)) = *(UINT32*)(&(pFirstTxDesc->cmd_sts)) |
OWNER_BY_GT;

    /* send the packet out using the Ethernet SDMA machine */
sdmaSendPackets(ETHERNET_PROTOCOL,PRIO0,
ETHERNET_DOWNLOADING_PORT,(UINT32*)(&(pFirstTxDesc->cmd_sts)));
    /* update the current descriptor */
    sdmaTxCurrentDescriptors[ETHERNET_DOWNLOADING_TX_PORT] =
(UINT32*)PHY_TO_VIRTUAL(pLastTxDesc->next_desc_ptr);

    return OK;
}
```

E.3 MPSC API

mpscLow.c: MPSC register manipulation.

mpscLow.h: MPSC function and structs declaration.

mpsc.c: Initialize MPSC port engines.

mpsc.h: MPSC structs and functions declaration.

brgLow.c , brg.c : BRG Register manipulation and initializations.

brgLow.h, brg.h: BRG function and structure declaration.

sdmaLow.c: SDMA register manipulation and initializations.

sdmaLow.h: SDMA functions and structures declaration.

sdma.c: SDMA memory allocation and transmit packet routines.

sdma.h: SDMA structs and function declaration.

```
void gtUartDevInit(MPSC_UART_CHAN* pChan)
{
    UINT32 tempRegValue;
    short mpscPortsExist[NUMBER_OF_MPSC_PORTS];
    PORT_ALLOCATION_STRUCT allocStruct;
    MPSC_PORT_CONFIG TempPortConfig; /* Initial config of MPSC channel */
    BRG_CONFIG_TUNING cfgTunBrg;  /* Initial configuration of the BRG engine */
    MPSC_SDCMR commandMpsc;

    /* Initialize the Communication Unit memory pool */
    usrMemInit();
    mpscPortsExist[MPSC_AS_UART_CONSOLE] = MPSC_PORT_AS_UART;

    /* set all mpscs direct to their ports */
    GT_REG_WRITE(MAIN_ROUTING_REGISTER, 0x7ffe38);

    /* connect the MPSC and Ethernet as RMII ports */
    GT_REG_WRITE(SERIAL_PORT_MULTIPLEX, 0x1102);

    UARTPort = MPSC_AS_UART_CONSOLE;

    /* Set UARTPort to work with BRG 0 */
    tempRegValue = GetDefaultRegisterValue(RCRR);
    tempRegValue = tempRegValue & (~(0xf << (UARTPort*8)));
    tempRegValue = tempRegValue | (UART_PORT << (UARTPort*8));
    /* set the register value in an internal table */
    SetDefaultRegisterValue(RCRR,tempRegValue);
    GT_REG_WRITE(RECEIVE_CLOCK_ROUTING_REGISTER,tempRegValue);

    tempRegValue = GetDefaultRegisterValue(TCRR);
    tempRegValue = tempRegValue & (~(0xf << (UARTPort*8)));
    tempRegValue = tempRegValue | (UART_PORT << (UARTPort*8));
    /* set the register value in an internal table */
    SetDefaultRegisterValue(TCRR,tempRegValue);
    GT_REG_WRITE(TRANSMIT_CLOCK_ROUTING_REGISTER,tempRegValue);

    /* initialization of the BRG engine */
    BRG_DEFAULT_CONFIG_GET(&(cfgTunBrg.config),UART_PORT);
```

```
BRG_DEFAULT_BAUD_TUNING_GET(&(cfgTunBrg.tuning),UART_PORT);
brgSetConfig(&cfgTunBrg,UART_PORT); /* set the BRG with defaults values */

/* Halt any Rx/Tx activities */
/* stop the MPSC Rx machines */
GT_REG_WRITE(CHR(UARTPort,2) , ABORT_RECEPTION);

/* workaround for wrong data read from MPSC internal registers. */
/* wait for the Rx to abort in the MPSC machine */
for(tempRegValue=0; tempRegValue<1000; tempRegValue++);

/* Abort SDMA Rx machine */
GT_REG_WRITE(sdmaRxCommandRegister[UARTPort], ABORT_RECEIVE);

/* check that the Rx aborted in the SDMA machine */
GT_REG_READ(sdmaRxCommandRegister[UARTPort],((UINT32*)&commandMpsc));
while(commandMpsc & ABORT_RECEIVE)
{
GT_REG_READ(sdmaRxCommandRegister[UARTPort],((UINT32*)&commandMpsc));
}

/* Abort the SDMA Tx machine */
GT_REG_WRITE(sdmaRxCommandRegister[UARTPort], ABORT_TRANSMIT);

/* Check that the Tx stopped in the SDMA machine */
GT_REG_READ(sdmaRxCommandRegister[UARTPort],((UINT32*)&commandMpsc));
while(commandMpsc & ABORT_TRANSMIT)
{
GT_REG_READ(sdmaRxCommandRegister[UARTPort],((UINT32*)&commandMpsc));
}


/* Set port 0 protocol to be UART */
mpscGetDefaultPortConfig(&TempPortConfig,UART_PORT);
TempPortConfig.protocol = UART_PROTOCOL;
mpscSetPortConfig(&TempPortConfig,UARTPort);

allocStruct.portNumber = UARTPort;
allocStruct.numberOfDescriptors = MIN_NUMBER_OF_UART_RX_DESC_ALLOC;
allocStruct.bufferSize = BUFFER_SIZE_FOR_UART;
allocStruct.priority = PRIO0;
```

```
allocStruct.protocolType = MPSC_PROTOCOL;
allocStruct.rxOrTx = RX_DESCRIPTOR;
sdmaAllocateDescriptorsForOnePort(&allocStruct);
sdmaInitRxDescriptors(UARTPort,MPSC_PROTOCOL,PRIO0);

/* TX */
allocStruct.portNumber = UARTPort;
allocStruct.numberOfDescriptors = MIN_NUMBER_OF_UART_TX_DESC_ALLOC;
allocStruct.bufferSize = 0;
allocStruct.priority = PRIO0;
allocStruct.protocolType = MPSC_PROTOCOL;
allocStruct.rxOrTx = TX_DESCRIPTOR;
sdmaAllocateDescriptorsForOnePort(&allocStruct);
sdmaInitTxDescriptors(UARTPort,MPSC_PROTOCOL,PRIO0);

allocBuffersForTxDescriptors();
}
```