MARVELL®

Tuning the Performance of the GT-64260A

Doc. No. MV-S300111-00, Rev. C

October 1, 2002

MOVING FORWARD FASTER®

Document Status	
Advanced Information	This document contains design specifications for initial product development. Specifications may change without notice. Contact Marvell Field Application Engineers for more information.
Preliminary Information	This document 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 document 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:

Table of Contents

List of Tables

List of Figures

Revision History

Table 1: Revision History

Document Type	Revision	Date
Preliminary Application Note	A	NA
Application Note	B	April 29, 2002
Application Note	C	October 1, 2002
Added the following diagrams to 4.1.1 "Performance Measurements" on page 17: • Figure 5, “PCI Read Bandwidth from SDRAM (Aggressive Prefetch),” on page 17 • Figure 6, “PCI Write Bandwidth to SDRAM,” on page 18		

Section 1. Introduction

This document describes the performance of the GT-64260A, including performance capabilities and tuning.

It discusses the following interfaces:

- CPU
- SDRAM
- PCI
- IDMA
- Device
- Crossbar

For each interface, this document generally provides:

- Performance measurements: This section shows the latency and/or bandwidth measured on an EV-64260A system.
- Performance tuning: This section describes the features of the EV-64260A that can be used to maximize the performance of that interface.
- Waveforms: This section provides waveforms captured on an EV-64260A system and the relevant register settings that were used to generate each waveform.

This document is written as only a supplement to the GT-64260A datasheet. Refer to the GT-64260A datasheet for a full description of features and configuration register settings

Note

Some of the waveforms shown in this document were captured at frequencies below the maximum frequency supported by the GT-64260A. The frequency was limited due to the additional load of the logic analyzer.

Section 2. PowerPC CPU Interface

This section describes the performance of the GT-64260A PowerPC CPU interface.

2.1 Performance Measurements

Table 2: CPU to SDRAM Latency

Transaction Type	Measurement	Figure	Latency (TClk cycles)
Read[1]	TS to TA assertion	Figure 1	11
Write[2]	TA to WR assertion on SDRAM bus	Figure 2	8

1. AACK delay was set to the minimum value (one cycle). SDRAM open pages were enabled and the access hit an SDRAM open page
2. SDRAM open pages were enabled and the access hit an SDRAM open page. TA Delay was disabled.

Table 3: CPU Read Bandwidth from SDRAM

CPU	Figure	Maximum Possible Bandwidth (MB/s)[1]	Measured Bandwidth (MB/s)[2]	Bus Utilization[3]
MPC7450	Figure 4	851 @ 133 MHz	848	99%

1. Total bandwidth for the CPU bus is 8 bytes x 133 MHz = 1064 MB/s (or 800 MB/s @ 100MHz). However, since there must be one turnaround cycle (due to the bus protocol) for every cache line read (which is four data cycles), the Maximum Possible Bandwidth is (4/5) = 80% of the Total Bandwidth. At 133 MHz, the Maximum Possible Bandwidth is 80% of 1064 MB/s = 851 MB/s.
2. Measured Bandwidth was calculated as: (data cycles/total cycles) x (total bandwidth). For the MPC7450, this was (512/642) x 1064 MB/s = 848 MB/s.
3. Bus utilization = (Measured Bandwidth) / (Maximum Possible Bandwidth).

2.2 Performance Tuning

The following features can be used to maximize the performance of the CPU interface:

- AACK delay
- Pipelining
- MPX and out-of-order mode

Additionally, disabling cache coherency can improve performance by eliminating the penalty associated with maintaining coherency.

These features are discussed in the following sections.

2.2.1 AACK Delay

Registers affected

CPU Configuration register (0x0) bits [25] (AACK_delay2), [23] (FastClk), [11] (AACK_delay)

Minimizing the AACK (Address Acknowledge) delay can reduce the latency of each CPU bus transaction and improve overall address bus utilization. AACK delay is defined as the minimum number of cycles before the GT-64260A can assert AACK after the CPU asserts TS (Transfer Start). The GT-64260A asserts AACK when it has room in its transaction queue to absorb the current transaction.

AACK delay is typically configured to two or three cycles. AACK delay can be configured to just one cycle if all of the following conditions are met:

- The system has one CPU and this CPU does not issue self-ARTRY
- TClk frequency is 100 MHz or less
- Multi-GT mode is not used

For best performance, configure AACK delay to the minimum value allowed by the system configuration and CPU used. Refer to the CPU's documentation for more information.

AACK delay is configured using three bits in the CPU Configuration Register: AACK_delay, AACK_delay2, and FastClk. Table 4 provides guidelines on how to program these bits based on the system configuration. For example, if the system meets the four requirements listed above, AACK delay can be configured to one cycle by setting the AACK_delay, AACK_delay2, and FastClk bits to 0.

Table 4: AACK Delay Programming Guidelines for Various System Configurations

System Configuration	AACK Delay	Required CPU Configuration Settings[1]	
		AACK_Delay	FastClk
Single CPU; this CPU does not issue self-ARTRY and TClk frequency is 100 MHz or less	1	0	0
Multi-CPU or Multi-GT mode or there is a chance the CPU will issue self-ARTRY and TClk frequency is 100MHz or less	2	1	0
TClk frequency is greater than 100MHz	2	X	1

1. If the CPU has a delayed ARTRY window, such as the MPC7450 (if its core clock to bus clock ratio is less than 5 to 1), set AACK delay to three cycles; set the AACK_delay2 bit to 1. This is the default setting. Change this setting if your system does not require it.

2.2.2 Pipelining

Registers affected

CPU Configuration register (0x0) bit [13]

Pipelining can significantly improve CPU bus bandwidth. Enabling the pipelining feature changes the GT-64260A queue depth from one to the maximum number of entries. Pipelining is disabled by default; enable it for maximum performance. The GT-64260A can pipeline at least six transactions. It has the capability to pipeline up to 14 consecutive reads in 60x mode.

This section describes a performance test that was used to measure the effects of pipelining in 60x bus mode. The test measured the CPU bus bandwidth on a series of cache line reads from SDRAM. This test was run with pipelining enabled and disabled. The MPC7450 CPU was used for this test.

Table 5 shows the CPU bus bandwidth that was achieved in this performance test. As shown in the table, the GT-64260A can provide 848 MB/s of read data from SDRAM, which uses 99% of the maximum possible bandwidth allowed by the 60x bus protocol. This performance is significantly better than that achieved when pipelining is not used.

The code used in this performance test is included in Appendix A. on page 36.

Table 5: Effect of Pipelining on CPU Bus Bandwidth

Pipelining Mode	Total Cycles[1]	Figure	Maximum Possible Bandwidth (MB/s)[2]	Measured Bandwidth (MB/s)[3]	Bus Utilization[4]
Disabled	1240	Figure 3	851 @ 133 MHz	439	52%
Enabled	642	Figure 4	851 @ 133 MHz	848	99%

1. This field shows the total number of cycles it took to complete 512 data cycles (512 reads from SDRAM) during the performance test.
2. Total bandwidth for the CPU bus is 8 bytes x 133 MHz = 1064 MB/s (or 800 MB/s @ 100MHz). However, since there must be one turnaround cycle (due to the bus protocol) for every cache line read (which is four data cycles), the Maximum Possible Bandwidth is (4/5) = 80% of the Total Bandwidth. At 133 MHz, the Maximum Possible Bandwidth is 80% of 1064 MB/s = 851 MB/s.
3. Measured Bandwidth was calculated as: (data cycles/total cycles) x (total bandwidth). For example, if the cycle count is 642, then the bandwidth is (512/642) x 1064 MB/s = 848 MB/s.
4. Bus utilization = (Measured Bandwidth) / (Maximum Possible Bandwidth).

2.2.3 MPX Mode

Registers affected

CPU Mode Register (0x120) bits [7:4]

The GT-64260A CPU interface supports both the 60x and MPX bus protocols. MPX bus mode provides performance advantages over 60x bus mode because it allows address and data streaming, where there are no turn-around cycles between consecutive address or data tenures. Additionally, MPX mode also allows out-of-order completions. Refer to the datasheet for a full description.

MPX bus mode can be used only in a system with a single MPC74xx CPU. It cannot be used in multi-GT or multi-CPU systems.

2.2.4 MPX Out-of-Order Mode

Registers affected

CPU Configuration register (0x0) bit [16]

Out-of-Order mode allows read data from a fast target device to pass data from a previous read to a slow device. Hence the read from the fast device is not held up, waiting for the previous transaction to complete. This feature is especially useful when there are many accesses to different target devices (SDRAM, Device, PCI, etc.).

When most of the CPU transactions are targeted to a single target interface, such as SDRAM, using this feature may reduce performance. This performance reduction occurs as a result of the bus protocol. There is a latency penalty of two cycles, in comparison to the read latency in 60x bus compatible mode, due to the DTI and DBG bus cycles. Refer to the datasheet for more details.

2.3 Waveforms

Figure 1 shows a CPU cache line read from SDRAM where open pages are enabled and the access hits an open page. Table 6 shows the register settings used to generate this waveform.

Figure 1: CPU Read from SDRAM

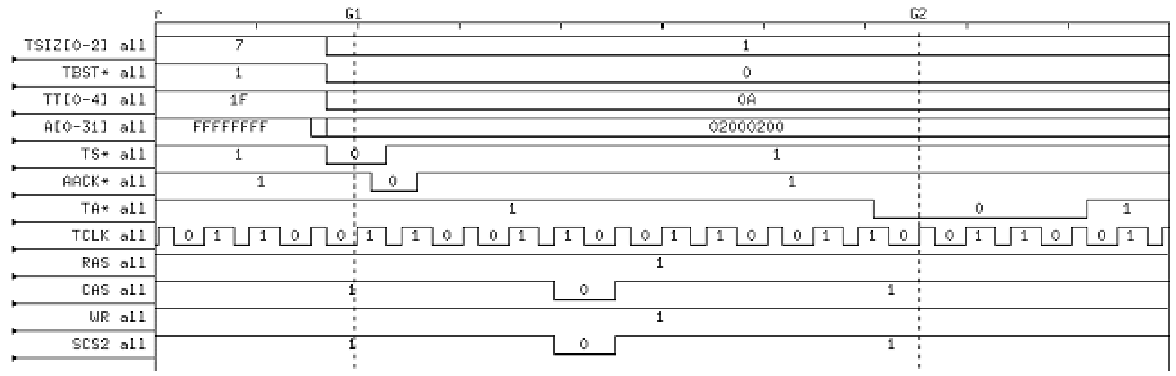


Table 6: Register Settings for CPU Read from SDRAM

Register Name	Register Offset	Register Value	Comments

CPU Configuration	0x000	0x700020FF	AACKDelay = 0
			AACKDelay2 = 0
			FastClk = 0
			No Multi-GT

SDRAM Timing Parameters	0x4B4	0x00000515	CAS Latency = 2
SDRAM Bank0 Parameters	0x44C	0x000F8000	Open pages enabled

Figure 2 shows a waveform for a CPU single-beat write to SDRAM. Table 7 shows the register settings used to generate this waveform.

Figure 2: CPU Write to SDRAM

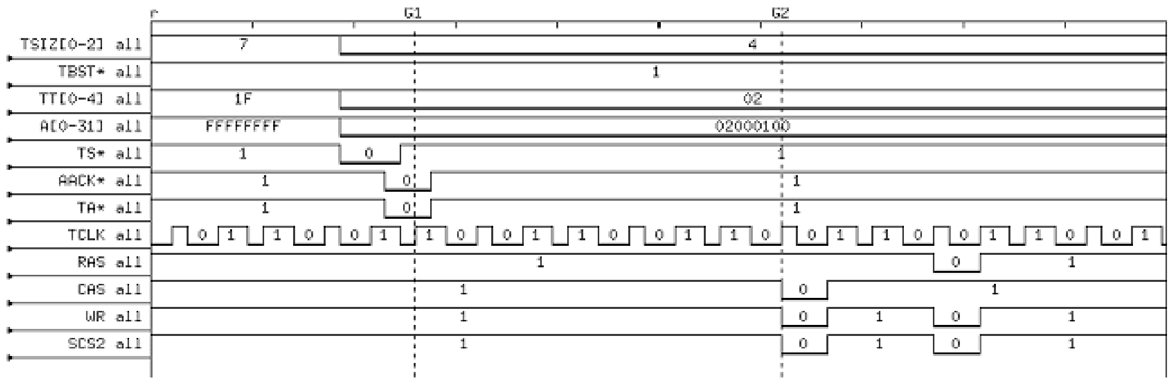


Table 7: Register Settings for CPU Write to SDRAM

Register Name	Register Offset	Register Value	Comments
CPU Configuration	0x000	0x700020FF	TA Delay = 0
SDRAM Timing Parameters	0x4B4	0x00000515	CAS Latency = 2
SDRAM Bank0 Parameters	0x44C	0x000F8000	SDRAM open pages enabled

Figure 3 illustrates the CPU read bandwidth from SDRAM with pipelining disabled. Table 8 shows the register settings used to generate this waveform.

Figure 3: CPU Read Bandwidth from SDRAM Without Pipelining

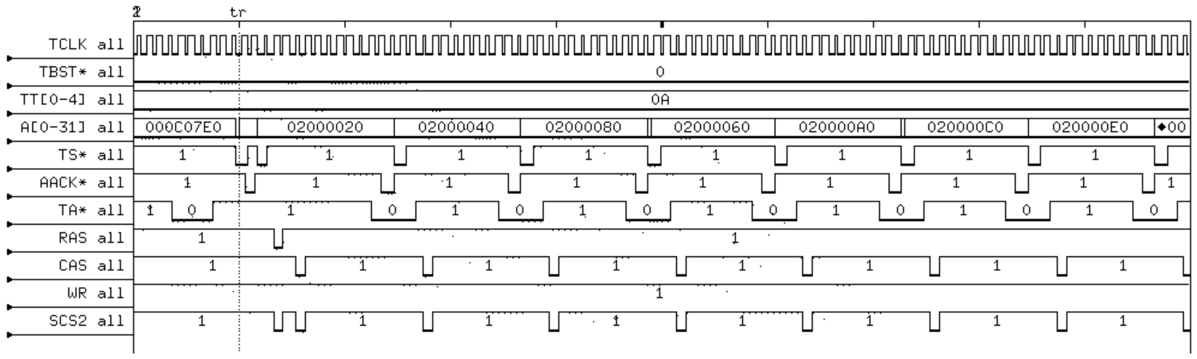


Table 8: Register Settings for CPU Read Bandwidth from SDRAM Without Pipelining

Register Name	Register Offset	Register Value	Comments
CPU Configuration	0x000	0x400100FF	Pipelining disabled
SDRAM Timing Parameters	0x4B4	0x00000515	CAS Latency = 2
SDRAM Bank0 Parameters	0x44C	0x000F8000	SDRAM open pages enabled

Figure 4 shows the CPU read bandwidth from SDRAM with pipelining enabled. Note that the throughput is much higher than that shown in Figure 3. Table 9 shows the register settings used to capture this waveform.

Figure 4: CPU Read Bandwidth from SDRAM With Pipelining

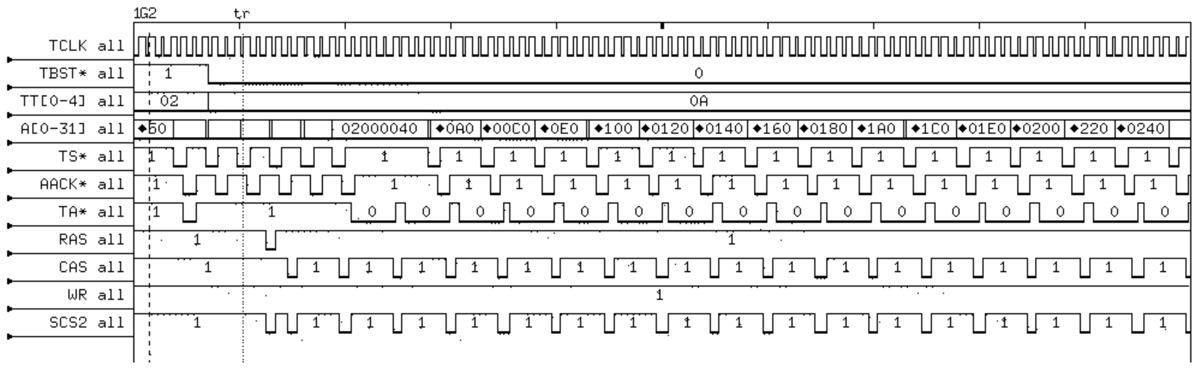


Table 9: Register Settings for CPU Read Bandwidth from SDRAM with Pipelining

Register Name	Register Offset	Register Value	Comments
CPU Configuration	0x000	0x700020FF	Pipelining Enabled
SDRAM Timing Parameters	0x4B4	0x00000515	CAS Latency = 2
SDRAM Bank0 Parameters	0x44C	0x000F8000	SDRAM open pages enabled

Section 3. SDRAM Interface

This section describes the performance of the SDRAM interface.

A typical SDRAM access requires a sequence of several cycles:

1. Activate (RAS) cycle followed by one or more idle cycles
2. Command (CAS) cycle followed by one or more idle cycles
3. One or more data cycles
4. Precharge cycle

The GT-64260A has several features that can improve SDRAM interface performance by eliminating or hiding some of these cycles, including:

- Open pages
- SDRAM timing parameters
- Interleaving

Additionally, the GT-64260A allows read buffer tuning to maximize performance. The following sections describe these features in detail.

3.1 Performance Tuning

3.1.1 Open Pages

Registers affected

SDRAM Bank0, Bank1, Bank2, Bank3 Parameters (0x44C, 0x450, 0x454, 0x458) bits [19:16]

Open pages improve SDRAM latency and bus utilization by eliminating most of the Activate and Precharge cycles. If open pages are enabled, there is no Precharge cycle at the end of a transaction and the page is thus kept open.

An open page "hit" occurs if a following transaction accesses the same virtual bank and row. In this case, there is no need for an Activate cycle. Each open page hit can save several cycles as there are no Precharge or Activate cycles.

An open page "miss" occurs if a following transaction to the same virtual bank accesses a different row. In this case, there is a performance penalty of closing the current open page and opening a new page.

In general, if open page hits are more likely than open page misses, enable open pages to maximize performance.

Refer to the datasheet for more information and waveforms.

3.1.2 SDRAM Timing

Registers affected

SDRAM Timing Parameters (0x4B4) bits [18:11], [5:0]

Set the SDRAM timing parameters, such as CAS Latency, to the minimum allowed values to improve performance. Doing so reduces the number of wait states before data can be read from or written to SDRAM. Refer to the datasheet for more details and waveforms. Make sure that the timing parameters used are also supported by the SDRAM used in the system.

3.1.3 Bank Interleaving

Registers affected

SDRAM Configuration (0x448) bits [15:14]

The GT-64260A supports both physical and virtual bank interleaving. Interleaving provides higher system performance by hiding a new transaction's Activate and Command cycles during a previous transaction's data cycles. This reduces the number of wait states before data can be read from or written to SDRAM, which increases bandwidth. Refer to the datasheet for a sample waveform.

The following section describes how to tune the SDRAM address control register to improve the likelihood of interleaving.

3.1.3.1 Benefiting from Interleaving

Registers affected

SDRAM Address Control (0x47C) bits [3:0]

To gain the maximum benefit from interleaving, determine the address decoding scheme that gives the highest likelihood of interleaving occurring. Interleaving occurs when there are multiple pending accesses to different SDRAM banks. If there are two accesses to the same virtual bank, then the Activate and Command cycles of the second transaction do not occur until after the data cycles of the first transaction. However, if the two accesses are to different virtual banks, then the Activate and Command cycles of the second transaction are issued while the first transaction is receiving read data.

For example, if both the CPU and a PCI master device access the same physical bank (SCS) but different 1KB regions within that bank, the CPU reads from addresses 0x700 – 0x7FF and the PCI master reads from addresses 0x800 – 0x8FF. Note that for CPU accesses, address bits [12:11] = 01, but for PCI accesses, address bits [12:11] = 00.

To take advantage of interleaving in this case, choose an address decoding scheme that maps these two memory regions to different virtual banks. In this case, it would be wise to select address bits [12:11] as the BankSel[1:0] bits. In other words, set the Address Control field to '0010'. Refer to the datasheet for a description of this register.

3.1.4 Read Buffer Assignment

Registers affected

SDRAM Configuration (0x448) bits [31:26]

The GT-64260A has two read buffers to store read data from SDRAM. Each requesting interface (CPU, PCI, IDMA, and Communication ports) can be assigned to use one of the two buffers. The buffer assignment scheme can favor one requesting interface over the others. For example, if CPU read latency is important and should not be delayed due to some PCI read data waiting in the buffer head, assigning one buffer for the CPU interface and the other buffer to the other interfaces guarantees the minimum CPU read latency.

Section 4. PCI Interface

This section discusses the performance of the PCI Master and Slave interfaces. The performance of the PCI Master affects IDMA transactions involving the PCI bus, CPU to PCI transactions, and P2P (PCI to PCI) transactions. The PCI Slave section focuses on the performance of PCI to SDRAM transactions.

The performance measurements provided in this section were taken from a Hewlett-Packard E2920 PCI Analyzer.

4.1 PCI Slave Performance

This section discusses the performance of PCI transactions to SDRAM. In this scenario, a PCI device acts as the PCI master (initiator) and the GT-64260A acts as the PCI slave (target). To maximize performance for these transactions, tune the GT-64260A PCI Slave appropriately.

4.1.1 Performance Measurements

Figure 5 shows the PCI read bandwidth from SDRAM for various read sizes and MBurst settings for both coherent and non-coherent reads.

Figure 5: PCI Read Bandwidth from SDRAM (Aggressive Prefetch)

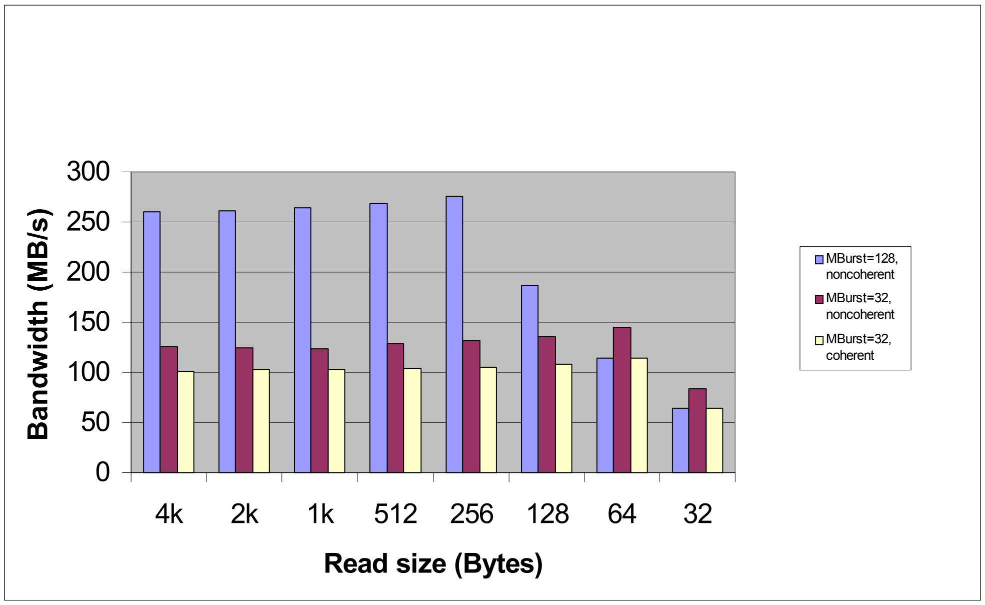


Figure 6 shows the PCI write bandwidth to SDRAM for various write burst sizes and MBurst settings for both coherent and nonco herent writes.

Figure 6: PCI Write Bandwidth to SDRAM

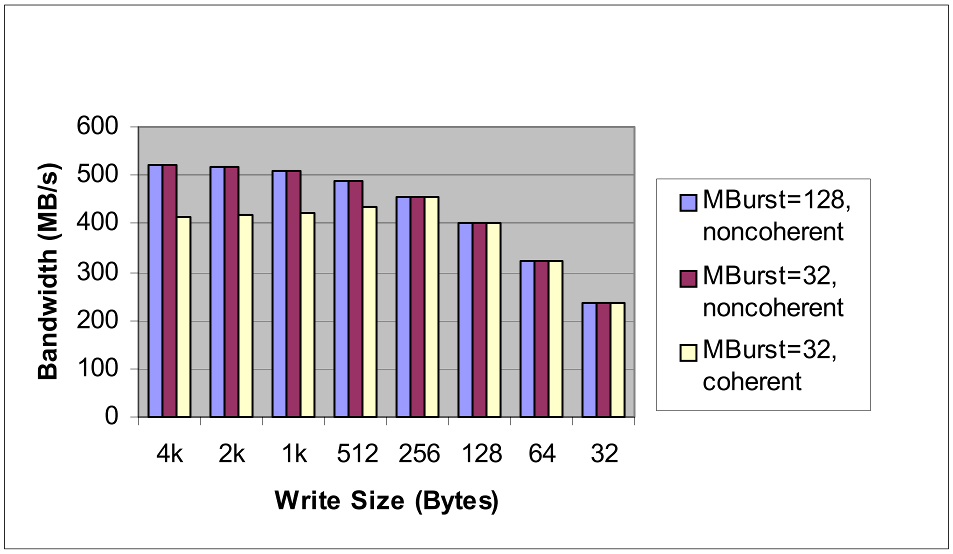


Table 10: PCI to SDRAM Latency

Transaction	Measurement	Figure	Value (PCI Clk cycles)
Read (Delayed Reads disabled)[1]	FRAME to TRDY assertion	Figure 8	11
Write	Data sampled on PCI bus to RAS assertion	Figure 7	10

1. Note that if delayed reads are enabled, the GT-64260A issues a Retry immediately. When the transaction is tried later, the GT-64260A can provide the data immediately without any wait states.

Table 11: PCI Read Bandwidth from SDRAM[1]

MBurst Size	Aggressive Prefetch	Figure	Measured Bandwidth (MB/s)[2]	Bus Utilization[3]
4	No	Figure 10	72	13%
16	No	Figure 11	171	32%
16	Yes	Figure 12	257	48%

1. Delayed reads are enabled
2. The bandwidth was measured by a Hewlett Packet E2920 PCI analyzer.
3. Bus Utilization = (Measured bandwidth) / (Maximum bandwidth). The maximum bandwidth of the PCI bus is 8B x 66 MHz = 528 MB/s.

Table 12: PCI Write Bandwidth to SDRAM

Transaction	Figure	Measured Bandwidth (MB/s)	Bus Utilization[1]
PCI write to SDRAM	Figure 7	528	100%

1. Bus Utilization = (Measured bandwidth) / (Maximum bandwidth). The maximum bandwidth of the PCI bus is 8B x 66 MHz = 528 MB/s.

4.1.2 Performance Tuning

A PCI write to SDRAM generally provides higher bandwidth than a PCI read from SDRAM, as writes avoid the latency associated with reads. On writes, the GT-64260A can absorb data continuously, as shown in Figure 7

The performance of PCI reads to SDRAM can be tuned by configuring the GT-64260A PCI Slave Unit. The features that can be used include:

- Delayed Reads
- Pre-fetching
- Aggressive Pre-fetching

The following sections discuss these features in detail.

Note

An IDMA transaction can also be used as an alternative mechanism to move data between the PCI interface and SDRAM. In some cases, an IDMA transaction can provide higher bandwidth than a PCI read from SDRAM.

4.1.2.1 Delayed Reads

Registers affected

PCI Access Control Registers. Refer to the datasheet for details.

Delayed reads allow for efficient use of the PCI bus, especially when there are multiple PCI masters.

If delayed reads are not enabled, the GT-64260A inserts wait states (by de-asserting TRDY) while it fetches data from SDRAM. During these wait states, the PCI bus is occupied and therefore another PCI master cannot initiate a transaction.

If delayed reads are enabled, the GT-64260A indicates a Retry and the bus is released. The GT-64260A fetches the data from SDRAM. When the initiator retries the transaction, the GT-64260A can provide the data with no wait states (not to first data nor to subsequent data). The bus is released quickly, allowing other PCI masters use the bus.

Figure 8 and Figure 9 show waveforms of PCI reads with delayed reads disabled and enabled, respectively.

4.1.2.2 Pre-fetching and Aggressive Pre-fetching

Registers affected

PCI Access Control Registers. Refer to the datasheet for details.

Pre-fetching can improve PCI bandwidth on reads. The amount of data that is pre-fetched for each transaction depends on the setting of the MBurst parameter. The MBurst parameter can be set to 4, 8, or 16 64-bit words (32B, 64B, or 128B). If the PCI device typically requires long read bursts and there are frequent disconnects during reads, increasing the MBurst size can improve bandwidth.

Aggressive pre-fetch can be used to improve bandwidth further. If aggressive prefetch is enabled, two bursts are pre-fetched at a time instead of one. However, the latency can be longer as more data must first be prefetched before it is returned to the PCI bus.

Note

If a memory region is marked as cache coherent, MBurst must be set to 4 for this region.

Do not use pre-fetching on a memory region if reads from this area are destructive (e.g. the target device is a FIFO). Additionally, pre-fetching lage amounts of data from SDRAM can take much of the available SDRAM bandwidth.

Table 11 summarizes PCI read bandwidth from SDRAM with various MBurst and aggressive pre-fetch settings

4.1.3 Waveforms

Figure 7 shows a PCI write to SDRAM. Note that in this waveform, the GT-64260A is able to continuously absorb data from PCI and write it to SDRAM. Table 13 shows the register settings for this waveform.

Figure 7: PCI Write to SDRAM

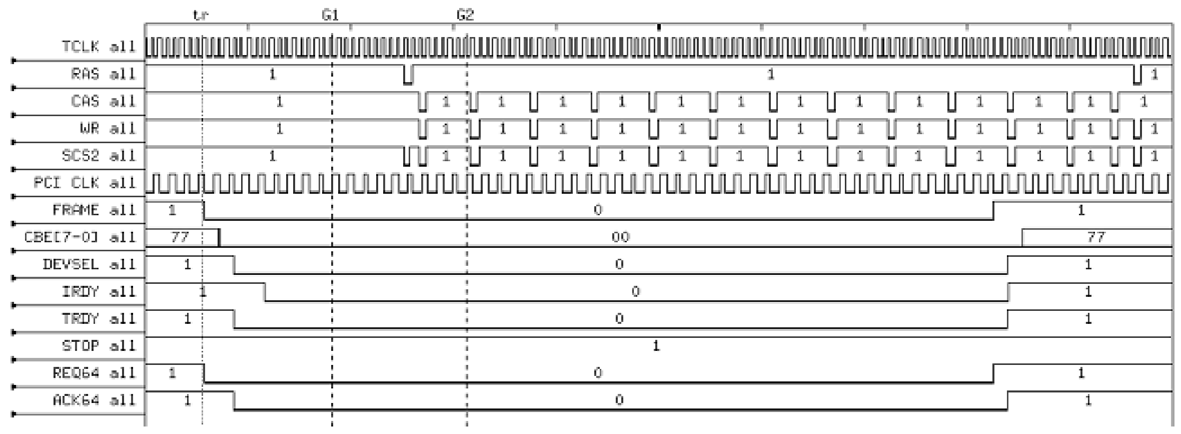


Table 13: Register Settings for PCI Write to SDRAM

Register Name	Register Offset	Register Value	Comments
SDRAM Timing Parameters	0x4B4	0x00000515	CAS Latency = 2
SDRAM Bank0 Parameters	0x44C	0x000F8000	SDRAM open pages enabled

Figure 8 shows a PCI read from SDRAM with delayed reads disabled. Notice that there are several wait cycles (IRDY is asserted by the PCI device but TRDY is not asserted) while the PCI slave fetches data from memory. During these wait states, the bus is occupied, which prevents other masters from using the bus.

Figure 8: PCI Read From SDRAM (Delayed Reads Disabled)

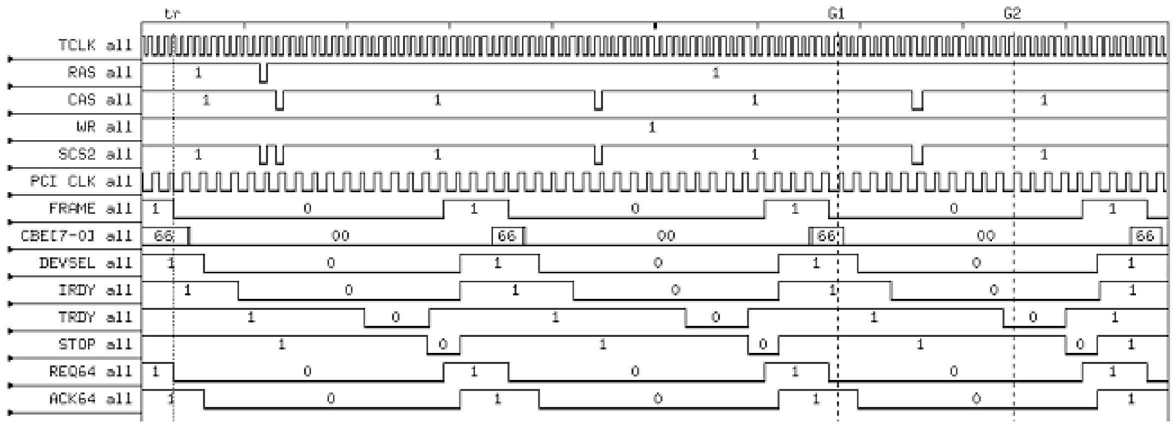


Table 14: Register Settings for PCI Read From SDRAM (Delayed Reads Disabled)

Register Name	Register Offset	Register Value	Comments
PCI Access Control Base 0 (Low)	0x1E00	0x00001000	Prefetch enabled Delayed reads disabled MBurst = 4 Aggressive prefetch disabled
PCI Access Control Top 0	0x1E08	0x0000001F	
SDRAM Timing Parameters	0x4B4	0x00000515	CAS Latency = 2
SDRAM Bank0 Parameters	0x44C	0x000F8000	Open pages enabled

Figure 9 shows a PCI read from SDRAM with delayed reads enabled. The bus is used more efficiently here than in Figure 8, where delayed reads are disabled. When the PCI device first requests a read, the GT-64260A ends the transaction with a retry (note the STOP assertion), which releases the PCI bus while it fetches data from SDRAM. In this case, another PCI master has the opportunity to issue a transaction instead of waiting for the original transaction to complete. Later on, the GT-64260A returns the data with no wait states.

Figure 9: PCI Read from SDRAM (Delayed Reads Enabled)

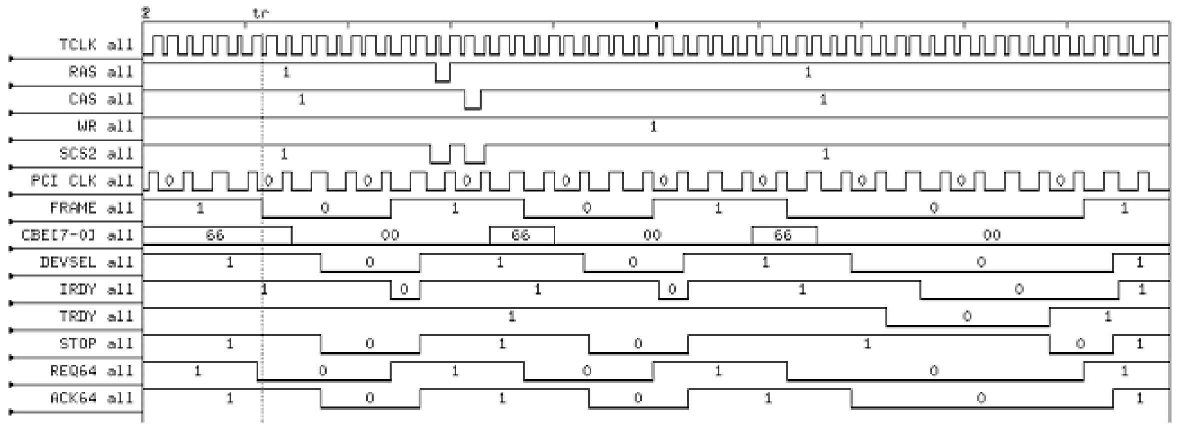


Table 15: Register Settings for PCI Read from SDRAM (Delayed Reads Enabled)

Register Name	Register Offset	Register Value	Comments
PCI Access Control Base 0 (Low)	0x1E00	0x00003000	Prefetch enabled Delayed reads enabled MBurst = 4 Aggressive prefetch disabled
PCI Access Control Top 0	0x1E08	0x0000001f	
SDRAM Timing Parameters	0x4B4	0x00000515	CAS Latency = 2
SDRAM Bank0 Parameters	0x44C	0x000F8000	Open pages enabled

Figure 10 shows PCI reads from SDRAM with pre-fetching enabled (aggressive pre-fetching disabled) and MBurst set to 4. Table 16 shows the register settings used to generate these waveforms.

Figure 10: PCI Reads from SDRAM (MBurst = 4)

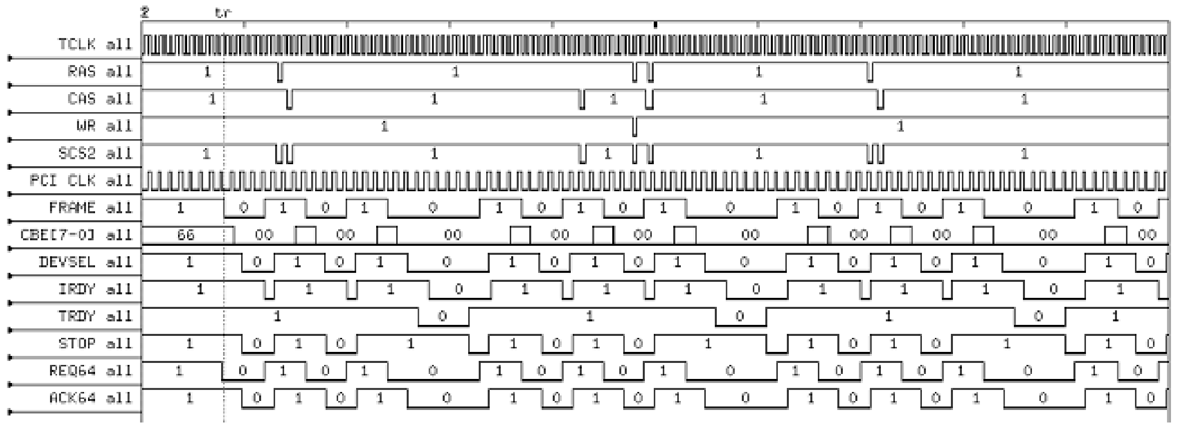


Table 16: Register Settings for PCI Reads from SDRAM (MBurst = 4)

Register Name	Register Offset	Register Value	Comments
PCI Access Control Base 0 (Low)	0x1E00	0x00003000	Prefetch enabled Delayed reads enabled MBurst = 4 Aggressive prefetch disabled
PCI Access Control Top 0	0x1E08	0x0000003F	
SDRAM Timing Parameters	0x4B4	0x00000515	CAS Latency = 2
SDRAM Bank0 Parameters	0x44C	0x000F8000	SDRAM open pages enabled

Figure 11 shows a waveform of PCI reads from SDRAM with pre-fetching enabled (aggressive pre-fetch disabled) and MBurst = 16. Compared with Figure 10 , where MBurst = 4, this waveform shows that the GT-64260A can provide a much longer data burst since it prefetches more data. Table 17 shows the register settings used to generate this waveform.

Figure 11: PCI Read from SDRAM (MBurst = 16)

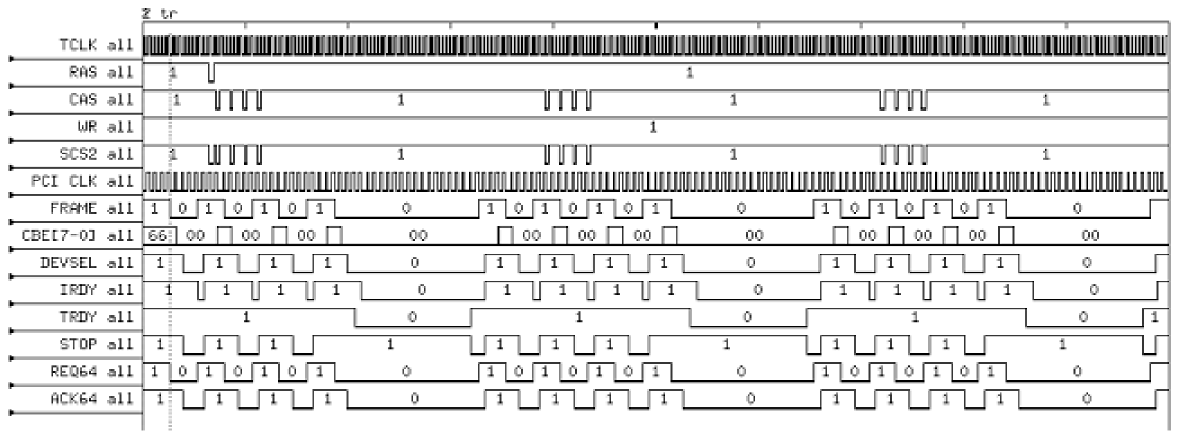


Table 17: Register Settings for PCI Read from SDRAM (MBurst = 16)

Register Name	Register Offset	Register Value	Comments
PCI Access Control Base 0 (Low)	0x1E00	0x01203000	Pre-fetching enabled Delayed reads enabled MBurst = 16 Aggressive pre-fetch disabled
PCI Access Control Top 0	0x1E08	0x0000003f	
SDRAM Timing Parameters	0x4B4	0x00000515	CAS Latency = 2
SDRAM Bank0 Parameters	0x44C	0x000F8000	Pages are kept open at the end of an access

Figure 12 shows a waveform of PCI reads from SDRAM with aggressive prefetch enabled and MBurst = 16. Compared with Figure 11 , where aggressive prefetch is disabled, this waveform shows that the GT-64260A can provide a much longer data burst and thus the bandwidth achieved is much higher. Table 18 shows the register settings for this waveform.

Figure 12: PCI Reads from SDRAM with Aggressive Prefetch (MBurst = 16)

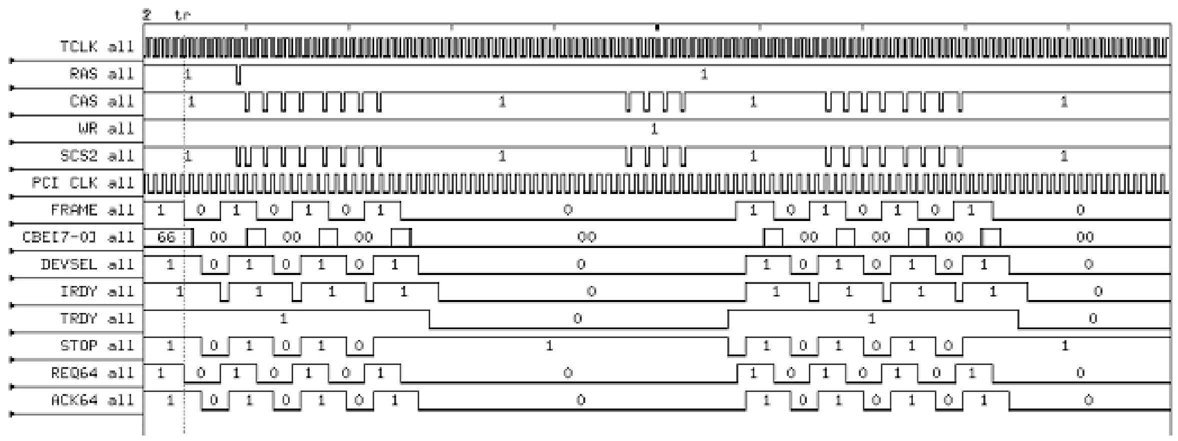


Table 18: Register Settings for PCI Reads from SDRAM with Aggressive Prefetch (MBurst = 16)

Register Name	Register Offset	Register Value	Comments
PCI Access Control Base 0 (Low)	0x1E00	0x01273000	Delayed reads enabled MBurst = 16 Aggressive prefetch enabled
PCI Access Control Top 0	0x1E08	0x0000003F	
SDRAM Timing Parameters	0x4B4	0x00000515	CAS Latency = 2
SDRAM Bank0 Parameters	0x44C	0x000F8000	Pages are kept open at the end of an access

4.2 PCI Master

The PCI Master interface is used on IDMA transactions, CPU to PCI transactions, and P2P (PCI to PCI) transactions.

4.2.1 Performance Measurements

4.2.1.1 IDMA Performance

Table 19: PCI Bandwidth on IDMA Transactions

Source	Destination	Burst Limit (Bytes)	Figure	Measured Bandwidth (MB/s)[1]	Bus Utilization[2]
SDRAM	PCI	128	Figure 13	495	93%
PCI	SDRAM	128	Figure 14	226	42%

1. The bandwidth was measured with a Hewlett Packet E2920 PCI analyzer.
2. Bus Utilization = (Measured bandwidth) / (Maximum bandwidth). The maximum bandwidth of the PCI bus is 8B x 66 MHz = 528 MB/s.

4.2.1.2 CPU to PCI Performance

Table 20 shows the measured latency from the CPU to PCI. Note that the latency of a CPU to PCI transaction is affected by the performance of the target PCI device. As shown in the table, the minimum total latency of a CPU read from PCI is the sum of 18 TClk cycles plus the number of TClk cycles it takes to complete the transaction on the PCI bus.

Table 20: CPU to PCI Latency[1]

Transaction	Measurement	Figure	Value (TClk cycles)
Read	TS to TA assertion	Figure 15	18 + PCI transaction duration
Write	TA to FRAME assertion	Figure 16	10

1. The internal PCI arbiter was used for this test.

4.2.2 Performance Tuning

The features that can be used to improve the performance of the PCI Master interface include:

- Read and write combining
- Fast back-to-back transactions
- 64-bit PCI
- Bus parking

The following sections discuss these features in more detail.

Note

A PCI write to SDRAM can deliver higher bandwidth than an IDMA transaction from PCI to SDRAM and is thus to be considered as an alternative method for moving data from PCI to SDRAM.

4.2.2.1 Combining

Registers affected

PCI Command (0xC00 & 0xC80) bits [5:4]

Read and write combining can eliminate some turnaround cycles on the PCI bus. Consecutive read or write transactions can be combined into a single read or write transaction if they meet certain criteria. This is especially useful for long DMA transfers, where a long-burst read or write is needed.

4.2.2.2 Fast Back-to-Back Transactions

Registers affected

PCI Status and Command (offset 0x4 in the PCI header) bit [9]

If fast back-to-back transactions are enabled, then the PCI master can start a new PCI transaction immediately after the previous transaction, without an idle cycle in between.

Note

Do not use this feature if any of the target devices on the bus do not support it.

4.2.2.3 64-bit PCI

Registers affected

PCI Command (0xC00 & 0xC80) bit [15]

For maximum performance, configure the PCI master to use 64-bit PCI transactions, if possible.

4.2.2.4 Bus Parking

Registers affected

PCI Arbiter Control (0x1D00 & 0x1D80) bits [20:14]

For maximum performance, configure the arbiter to use bus parking, to avoid delays due to bus arbitration.

4.2.3 Waveforms

Figure 13 shows a waveform of an IDMA from SDRAM to PCI. Table 21 shows the register settings used.

Figure 13: IDMA from SDRAM to PCI

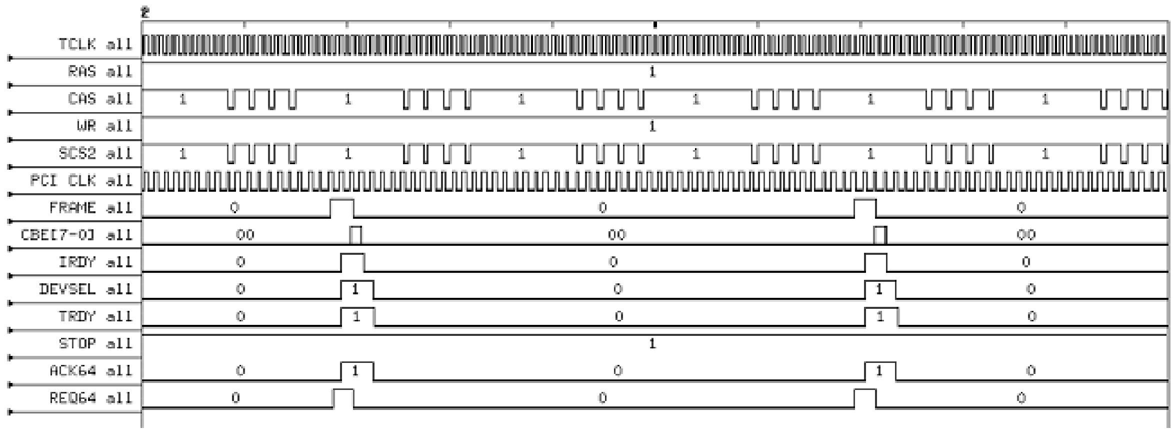


Table 21: Register Settings for IDMA from SDRAM to PCI

Register Name	Register Offset	Register Value	Comments
Channel 0 DMA Control	0x840	0x00001D00	IDMA with 128-byte burst limit (set to 0x00001C00 if using 32-byte burst limit)
PCI Command	0xC00	0x00069370	Combining and Req64 enabled
SDRAM Timing Parameters	0x4B4	0x00000515	CAS Latency = 2
SDRAM Bank2 Parameters	0x454	0x000F8000	SDRAM open pages enabled

Figure 14 shows an IDMA transaction from PCI to SDRAM.

Figure 14: IDMA from PCI to SDRAM

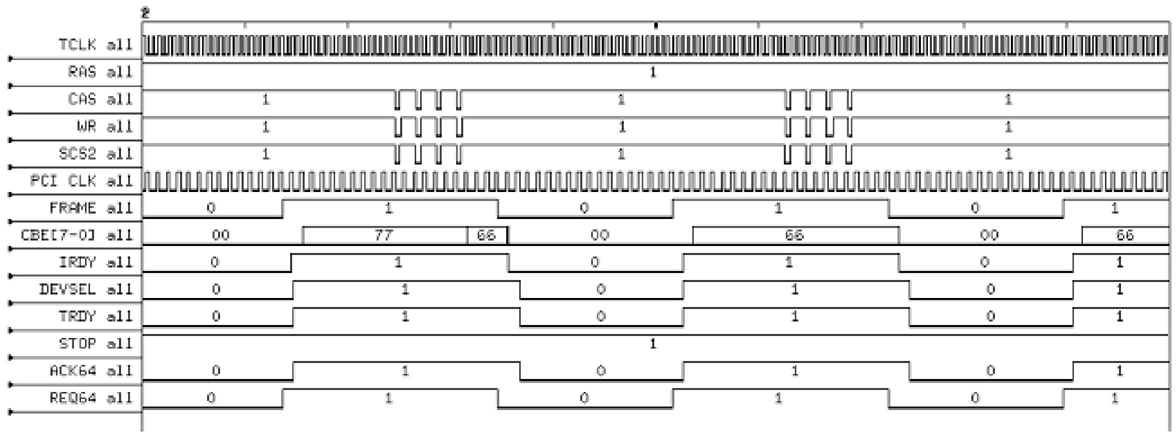


Table 22: Register Settings for IDMA from PCI to SDRAM

Register Name	Register Offset	Register Value	Comments
Channel 0 DMA Control	0x840	0x00001D00	IDMA with 128-byte burst limit (set to 0x00001C00 if using 32-byte burst limit)
PCI Command	0xC00	0x00069370	Combining and Req64 enabled
SDRAM Timing Parameters	0x4B4	0x00000515	CAS Latency = 2
SDRAM Bank2 Parameters	0x454	0x000F8000	SDRAM open pages enabled

Figure 15 shows a waveform for a CPU read from PCI.

Figure 15: CPU Read from PCI

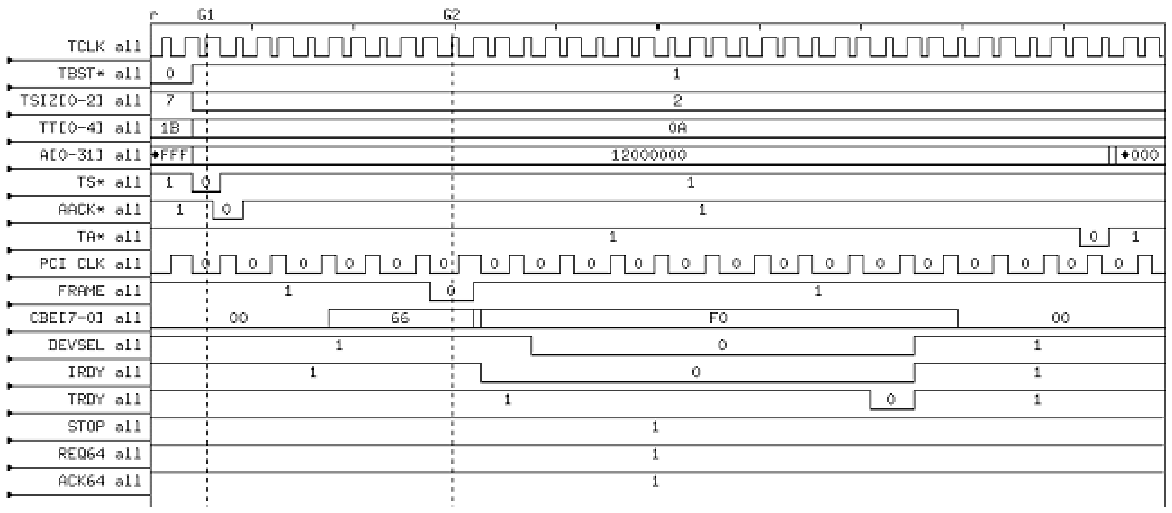


Figure 16 shows a waveform for a CPU write to PCI.

Figure 16: CPU Write to PCI

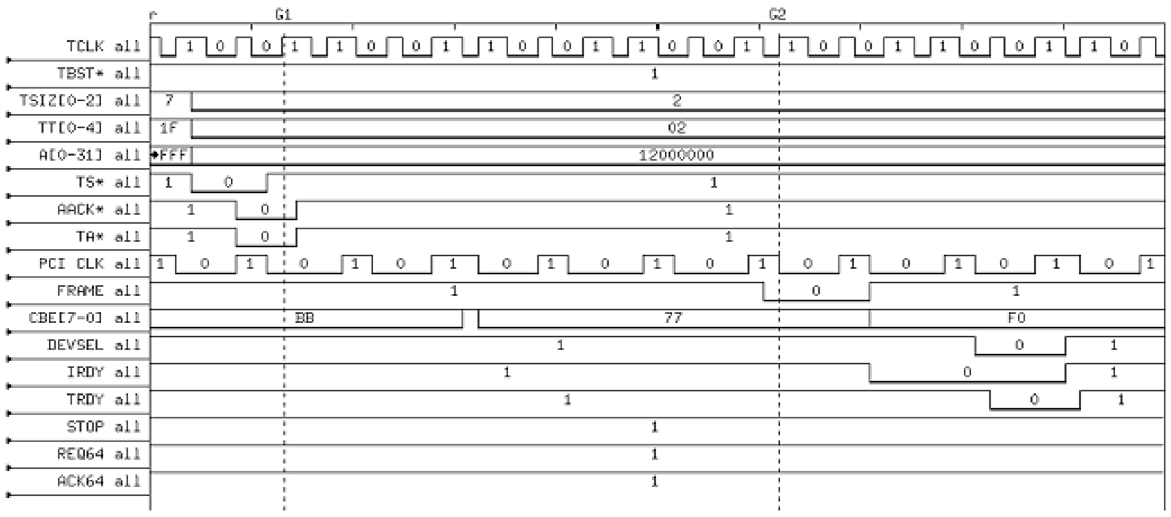
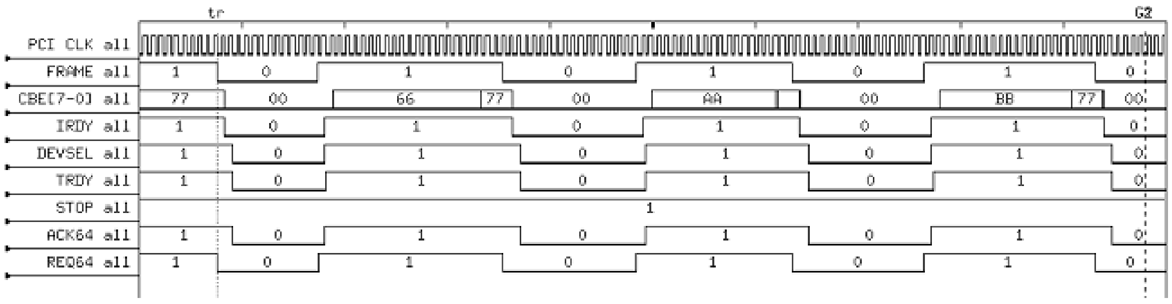


Figure 17 shows the destination PCI bus on an IDMA from PCI1 to PCI0. The IDMA burst limit is 128B.

Figure 17: DMA from PCI1 to PCI0



Section 5. IDMA

This section discusses IDMA performance. Refer to Section 4.2 for specific information on IDMA transactions involving the PCI bus.

5.1 Performance Tuning

To get the maximum IDMA performance, first tune the performance of the interfaces involved (PCI Master, SDRAM, etc.). In addition, the following features can be used to tune IDMA performance:

- Burst limit
- IDMA arbitration

The following sections discuss these features in detail.

5.1.1 Burst Limit

Registers affected

Bits [8:6], [2:0] of all Channel Control registers

Increasing the burst limit can increase the IDMA bandwidth achieved. However, large bursts take much of the bandwidth of the respective interfaces.

Note

If an IDMA transaction accesses a cache coherence SDRAM region, the burst limit must not exceed 32 bytes.

5.1.2 IDMA Arbitration

Registers affected

DMA Channels 0-3 and 4-7 Arbiter Control (0x860 & 0x960)

The IDMA controller has two programmable round-robin arbiters per the two channels groups. Each channel can be configured to have different bandwidth allocation. The user can define each of the 16 slices of this “pizza arbiter". At each clock cycle, the arbiter samples all channel requests and gives the bus to the next agent according to the “pizza arbiter".

Section 6. Device Interface

This section describes the performance of the device interface. The GT-64260A can interface with high performance devices, such as 32-bit syncBurst SRAM, and provide bursts of up to 32B.

6.1 Performance Measurements

Table 23: CPU to Device Latency[1]

Transaction	Measurement	Figure	Latency (TClk cycles)
Read	TS assertion to ALE low	Figure 18	5
	TS to TA assertion	Figure 18	13
Write	TA to ALE low	Figure 19	6

1. This table shows the latency to a 32-bit device with the highest performance device timing parameters.

6.2 Performance Tuning

To maximize device bus performance, tune the device timing parameters to meet the needs of the devices used.

6.2.1 Device Timing Parameters

Registers affected

Device Bank0, Bank1, Bank2, Bank3, and Boot Device Parameters (0x45C, 0x460, 0x464, 0x468, 0x46C)

To maximize performance, set the device timing parameters to the highest performance setting allowed by the device being used.

6.3 Waveforms

Figure 18 shows a waveform for a CPU read from a 32-bit device. The minimum device timing parameters were used. Table 24 shows the register settings used to generate this waveform.

Figure 18: CPU Read from 32-bit Device

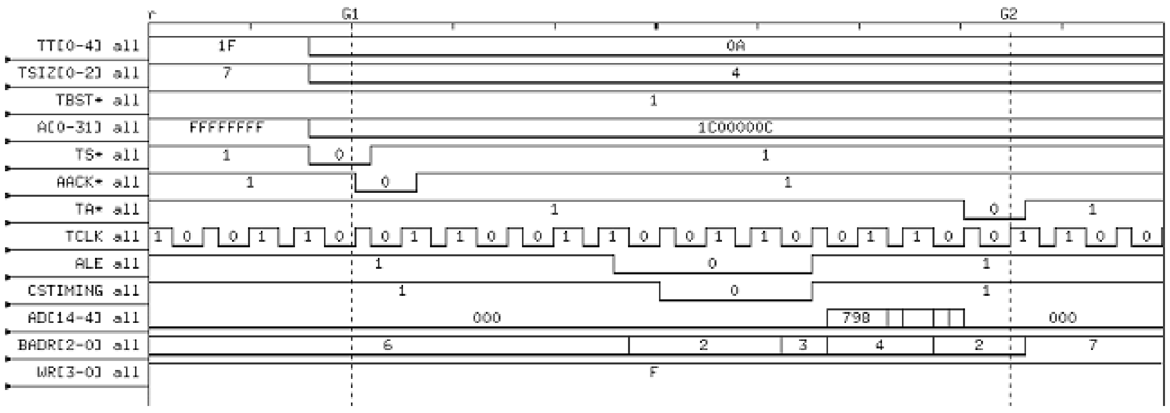


Table 24: Register Settings for CPU Read from 32-bit Device

Register Name	Register Offset	Register Value	Comments

Device Bank0 Parameters	0x45C	0xC0205899	Acc2First = 3
			Acc2Next = 1
			BadrSkew = 0
			ALE2Wr = 3
			WrLow = 1
			WrHigh = 0

Figure 19 shows a CPU write to a 32-bit device. The minimum device timing parameters were used. Table 25 shows the register settings used in generating this waveform.

Figure 19: CPU Write to 32-bit Device

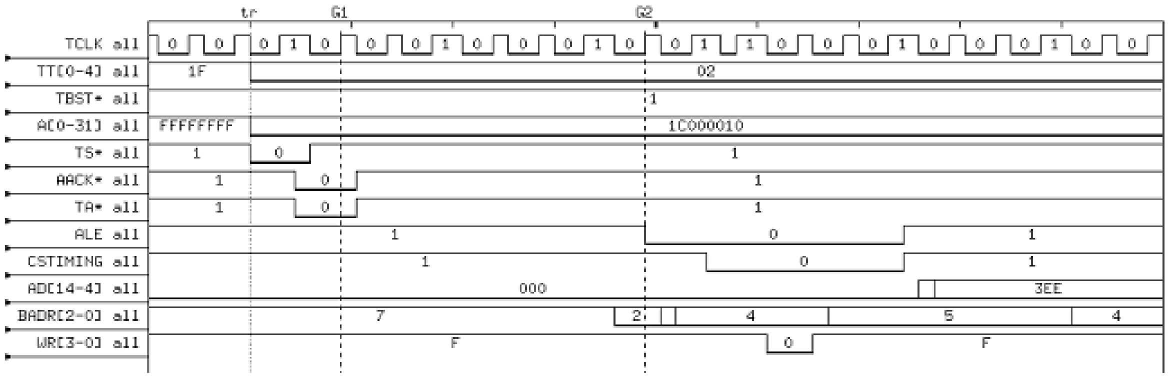


Table 25: Register Settings for CPU Write to 32-bit Device

Register Name	Register Offset	Register Value	Comments

Device Bank0 Parameters	0x45C	0xC0205899	Acc2First = 3
			Acc2Next = 1
			BadrSkew = 0
			ALE2Wr = 3
			WrLow = 1
			WrHigh = 0

Section 7. Crossbar

The crossbar control registers, also referred to as the “pizza arbiter,” allow performance tuning of the various interfaces in situations where conflicts occur over resources. Refer to the datasheet for more details.

This subject will be discussed further in a future revision of this document.

Appendix A. CPU Bus Bandwidth Performance Test Code

This section provides the code used in the CPU bus bandwidth performance test in Section 2.2.2. This code is included as a reference.

```
void CPUBandwidthTest()
{
    unsigned int temp;

    cntmrStart(CNTMR_0,0xffffffff,COUNTER);
    MemoryLoop();
    temp = cntmrDisable(CNTMR_0);
    temp = 0xffffffff - temp;
    printf("timer/cntr = 0x%x, %d\n",temp,temp);
}

void MemoryLoop()
{
    unsigned int temp, counter, numberOfLoops, address;

    numberOfLoops = 0x20;
    address = 0x82000000;

    for (counter = 0; counter < numberOfLoops; counter++)
    {
        temp = READWORD(address);
        temp = READWORD(address + 0x20);
        temp = READWORD(address + 0x40);
        temp = READWORD(address + 0x60);
        address = address + 0x80;
    }
}
```