INTRODUCTION

This document provides FAQs or Frequently Asked Questions (and answers) for the GT-642xx Discovery devices.

For easy reference, the Table of Contents is based on the organization of Galileo's data sheets. If viewing this document electronically, click on any entry in the Table of Contents to jump to that section. Each section may contain multiple FAQs.

The content of this document is not guaranteed to be error free. The respective Galileo data sheet is the authoritative document over these FAQs. If there are any questions or concerns please contact Galileo's technical support.

Each FAQ answer has a date showing when it was last updated. Please check the Galileo website periodically for the latest release and for any updates.

TABLE OF CONTENTS

GT-64260 FAQs

Section 1. Address Space Decoding

Address Space Decoding Errors - page 2

Section 2. CPU Interface:

CPU Big and Little Endian Support - page 3

Section 3. Device Controller:

Ready* Support - page 3

Section 4. IDMA Controller:

IDMA Channel Control - page 4

Section 5. PowerPC Cache Coherency:

Background - page 5

Section 6. Multi Protocol Serial Controller (MPSC):

UART Mode - page 6

Section 7. General Purpose Port:

GPP Interrupts - page 6

Section 8. Internal Arbitration Control:

Example - page 7

Section 9. Verilog:

CPU Interface - page 8

Device Controller - page 10

GT-64240 FAQs

Section 10. CPU Interface:

MIPS 64-bit Multiplexed Address/Data Bus Interface - page 10

CPU Synchronization Barrier - page 10

Section 11. Device Controller:

Interfacing With 8/16/32-Bit Devices - page 11

Section 12. PCI Interface:

Data Endianess - page 12

64-bit PCI Interface - page 12

Section 13. IDMA Controller:

IDMA Channel Control - page 13

Section 14. AC Timing:

Restrictions - page 14

GT-64260 FAQs

1. Address Space Decoding

1.1 Address Space Decoding Errors

Question:

A write to the GT-64260 will go to one region and the read will come from the second. I see the problem when accessing address 0xf2000000. I changed the base address programming for CS3 region to 4000_0000 to 4100_0000 and I got the same result. Attached is the transcript file with all register access. Region 5XXX_XXXX is the DISCO,region 7XXX_XXXX is an 8 bit Device bus region and region FXXX_XXXX is a 16 bit bootflash region. It is possible I have a region crossed since I've moved stuff around. All of the other regions seem to be working OK.

These are the sequence of writes to the GT-64260 registers:

1250 ns: Wrote address: 50000028 with data: 00000600

1260 ns: Wrote address: 50000030 with data: 000006ff

1270 ns: Wrote address: 50000228 with data: 00000700

1280 ns: Wrote address: 50000230 with data: 000007ff

1290 ns: Wrote address: 50000248 with data: 00000800

1300 ns: Wrote address: 50000250 with data: 000008ff

1310 ns: Wrote address: 50000038 with data: 00000400

1320 ns: Wrote address: 50000040 with data: 00000410

1330 ns: Wrote address: 50000238 with data: 00000f00

1340 ns: Wrote address: 50000240 with data: 00000fff

1940 ns: Read from address: 50000028 Recieved data: 00000600

2017 ns: Read from address: 50000030 Recieved data: 000006ff

2087 ns: Read from address: 50000228 Recieved data: 00000700

2157 ns: Read from address: 50000230 Recieved data: 000007ff

2227 ns: Read from address: 50000248 Recieved data: 00000800

2297 ns: Read from address: 50000250 Recieved data: 000008ff

2367 ns: Read from address: 50000038 Recieved data: 00000400

2437 ns: Read from address: 50000040 Recieved data: 00000410

2507 ns: Read from address: 50000238 Recieved data: 00000f00

2577 ns: Read from address: 50000240 Recieved data: 00000fff

2587 ns: Wrote address: 5000045c with data: 002fffff

2597 ns: Wrote address: 50000460 with data: 000fffff

2607 ns: Wrote address: 50000464 with data: 000fffff

2617 ns: Wrote address: 5000046c with data: 001fffff

2934 ns: Read from address: 5000045c Recieved data: 002fffff

3060 ns: Read from address: 50000460 Recieved data: 000fffff

3186 ns: Read from address: 50000464 Recieved data: 000fffff

3312 ns: Read from address: 5000046c Recieved data: 001fffff

3312 ns: **** Device Bus Configuration Complete ****

Answer (last updated January 12, 2001):
It looks like you have some address windows crossing. In such case the Disco will behave very unstable. You see the problem when accessing address 0xf2000000 which is the defaults value for PCI0 Mem1 region. You actually have an overlap between the BootCs and Mem1, Mem2 and Mem3 of PCI0.

2. CPU Interface:

2.1 CPU Big and Little Endian Support

Question:
Assume I have a 32-bit Motorola MPC7400 CPU running in little endian mode, a GT-64260 chip, and a 32 bit PCI bus.

First case: The CPU initiates a 32-bit write of value 0x11223344 to address 0x1000 in local memory range. According to Motorola docs, the address munging on the CPU would cause such a write to appear in local memory at addresses 1004,1005,1006,1007 and contain the values 11,22,33,44 respectively (i.e. actually in big endian order). OK so far.

Second case: The CPU initiates a 32-bit write of value 0x11223344 to address 0x10000000 in PCI range. I would assume that this write appears on the MPC bus as a 32-bit write to addresses 0x10000004-0x10000007, containing data 11, 22, 33, 44. Is there a byte swap register configuration on the GT that would cause this 32-bit write to be transmogrified into a 32-bit write to PCI addresses 0x10000000-0x10000003, containing data 44,33,22,11?

Answer (last updated January 11, 2001):
Their is a basic problem with your assumption. The GT-64260 doesn't support little endian with the MPC7400. The GT-64260 only supports Big Endian CPU bus.

3. Device Controller:

3.1 Ready* Support

Question:
Please refer to “Ready* Extending WrLow Parameter“ Figure of GT64260 spec. Please look at WR* and Ready* signal on that timing diagram. The datasheet says the Ready* is sampled on the last rising edge of the WrActive (WrLow)

phase during write cycle. If that's the case, how can Ready* be DON'T-CARE on the rising edge of the (extended) WrLow phase as shown? Won't Ready* be sampled 'X' when that WR* is going high?

What will make sense is if WR* is shown going high one TClk cycle too late. If the WR* goes high a clock cycle earlier, then THAT will be the last rising edge of WrActive, and in that case, Ready* will be sampled low.

Answer (last updated January 10, 2001):
The GT-64260 Ready* signal has no meaning after it has been sampled active. From this point on, the cycle is a normal cycle that doesn't need to take Ready* into account. When WR* goes high, the cycle is already finished - Ready* need not be sampled any more, it had already "done it's job" by extending the cycle for two clocks.

The WR* stays low two cycles more than it was meant too - this is due to Ready* signal activity. The GT regards the access as starting only when the device is ready to accept it - when Ready* is sampled low.

4. IDMA Controller:

4.1 IDMA Channel Control

Question:
Our application plugs into the PCI interface of GT64260 and is responsible for moving packets across this interface to/from CPU. I need validation from you that these scenarios are indeed doable:

<Scenario A> From CPU -> PCI Application:

while (there is packet to trasfer) {

0. CPU will use available IDMA Channels. The availability is found by reading GT64260's DMA (ChanAct bit) control reg.

1. CPU will setup packet in system memory (hanging off GT64260)

2. CPU will program the IDMA control registers, and enable DMA

}

<Scenario B> From PCI Application->CPU:

Ideally, what we would like to do:

0. At init time, cpu sets up a chained-list of descriptors in memory. Each descriptor points to a buffer big enough to capture an entire packet. The byte-count indicates max bytes that the buffer can take. The channel en bit is set, and is setup for chained-dma, and demand-dma. The source address in these descriptors are always the same, pointing to the PCI bus; and held the same throughout the DMA xfer.

1. CPU programs the receive IDMA channel with the base pointer to the first descriptor.

2. The application will assert DMAReq, indicating it has a packet to send. Discovery will initiates the transfer. Note that the byte-count in the descriptor memory does not indicate the true length of transfer.

I need to understand better the relationship between DMAReq and DMAAck. How will the application know that it's request has been processed. How long should DMAReq be set for? I am assuming the following:

- DMAReq is set for 1 cycle
- Application waits for DMAAck to indicate xfer is done
- If more data needs to be xfer, repeat.

3. During the last burst, the application will assert EOT. This will cause the DMA to finish this burst, and then close the descriptor; thus changing the ownership bit to CPU. Also, since we are in chained-mode, it will fetch the next descriptor.

4. Repeat steps 2,3.

Are there any side-effects on using a PCI interrupt to the application. What other scenarios can cause this interrupt to trigger. Can we use an MPP to trigger DMA-Done? Also, I am assuming that the DMA-Done/interrupt is per BurstLim transfer (which could be 128-bytes in our application). Is this correct?

Will the GT64260 closes the descriptor and returns ownership to CPU in the event EOT is asserted on IDMA channel?

Answer (last updated January 9, 2001):
Scenario A looks fine.

Scenario B step 2: DMAReq* can be set to edge triggered in your case. Once a falling edge is detected on the pin, the DMA will be initiated. DMAAck* will be asserted for one Tclk and only at initiation of the DMA, not when it is done (there is a choice to assert DMAAck* during the transaction but only if the access is to the device bus). Instead, you could enable a PCI interrupt on completion of the DMA.

Scenario B setp 3: Asserting EOT* closes the descriptor, fetches the next descriptor and return ownership to CPU. This feature will not work in the first Discovery silicon. This bug will be fixed in the second revision. Without EOT*, the DMA will fetch the max byte count of data from before stopping.

All interrupt in the 64260 can be routed to any of the seven interrupt pins; one for each PCI bus so a total of two(open drain outputs), one for the CPU and four MPP pins. You can enable one of the MPP pins to interrupt your application.

DMA done interrupt is effected when the DMA finishes the last descriptor, one with the null pointer.

Question:
Regarding 32 bit device bus operations, does the bus operate the same way when the transfer is initiated from the DMA engine instead of the PPC interface? For illustration purposes, lets assume that the DMA engine is requesting 4 32 bit words from a beginning address which is 0 based, what will the DISCO perform for transfers? (my assumption is 0, 4, 8, C). Now, lets take the same example except the transfer base starting address is 4. What will the GT-64260 perform for transfers? (my assumption is 4, 8, c, F).

Answer (last updated January 11, 2001):
The DMA splits the total byte count it needs to transfer, to "small" transactions to the Device interface unit, based on the Burst Limit the DMA is configured to, and based on the address alignment.

In case of DMA transfer of 16 bytes from offset 0x4 of the Device bus, if Burst Limit is 8, the DMA issues three requests to the Device unit - 1st is read of 4 bytes from offset 0x4, 2nd is read of 8 bytes from offset 0x8, and 3rd read of 4 bytes from offset 0x10.

As you already know, there is a restriction in the GT-64240 Device interface unit, that any access to a 32-bit device, is executed as two bursts. You will see all 3 transactions as reads of 8 bytes (burst of two 32-bit words) to offsets 0x0,0x8 and 0x10 respectively.

5. PowerPC Cache Coherency:

5.1 Background

Question:
I'm confused by the PCI Snoop registers which appear to be different than the CPU snoop registers. Can you find out how these are used, and what their relation is to the CPU snoop registers. In our application, we expect to see a lot of PCI-to-SDRAM accesses which must be cache-coherent (snooped by the processor).

Answer (last updated January 11, 2001):
The snoop regs of the CPU are used by the DMA engines and the communication unit. The PCI snoop regs are used by the PCI. The actual snooping operation is done in the SDRAM unit, but it gets the indication that snooping is required from the accessing unit - either the DMA and COMM (using the CPU regs) or the PCI unit.

6. Multi Protocol Serial Controller (MPSC):

6.1 UART Mode

Question:
It is not clear to me how the MPSC hardware flow control works in UART Mode. I would expect, the GT-64260 asserts RTS to indicate okay to send to the GT-64260. GT-64260 interprits CTS as okay to send.

If NLM = 0, do both CD and CTS need to be asserted to transmit data out of the discovery? How does the discovery back pressure the other guy?

If NLM = 1, only CTS is used to determine when the Discovery can transmit? How does the discovery back pressure the other guy?

If there is no hardware back pressure to the other guy, is SW back pressure required? Is there some maximum speed up to which no back pressure is needed?

Answer (last updated January 11, 2001):
Lets simplify this:

Normal mode (NLM bit = 0)

Transmiter side :

When want to transmit :

Asserts RTS, waits for CTS and then transmits.

Receiver side :

When preparing for receive :

Waits for CD and only when CD asserted sample the line.

When the receiver have to initiate back-pressure it should use one of the GPPs (MPPs in the disco)

and connect is to the CTS of the Receiver.

So, the connection should be like this :

Tx:TXD --------> Rx:RXD

Tx:RTS --------> Rx:CD

TX:CTS<-------- Rx:GPP (for back pressure).

(the gpp behavior is defined by the receivers SW and normaly it should be

deasserted only when the Receiver runs out of descriptors.)

- There is no maximum speed for backpressure it depends on the traffic and the descriptors management.

There is a limitation on the UART speed.

7. General Purpose Port:

7.1 GPP Interrupts

Question:
How do the Discovery GPP interrupts work? Are they suitable for edge triggered sensitive interrupts?

Answer (last updated January 11, 2001):
When a GPP input pin is asserted (toggle from 0 to 1 if configured to active high, or toggle from 1 to 0 if configured to active low), the corresponding bit in the GPP cause register is set, and an interrupt is generated. The interrupt will be kept asserted until interrupt handler clears the bit in the GPP cause register. If using multiple GPP inputs as interrupts,

if when clearing one cause bit, some other cause bit is still active, interrupt is kept asserted (the interrupt is an OR of the cause bits). Once a cause bit is asserted, there is no way to register new interrupts from the same GPP input, until the corresponding GPP cause register bit is cleared.

Question:
How does a Discovery-based system handle other interrupt signals? The GT-64260. does not appear to have any interrupt inputs which would allow interrupts to be routed to either processor. Ideally, there would be 4 to 8 inputs for board use. As it stands, we need to implement an OpenPIC externally, which is a significant design effort.

Interrupt Destinations:

CPU #0

CPU #1

Interrupt Sources:

PCI 0, INTA# to INTD# -- PCI bus #0

PCI 1, INTA# to INTD# -- PCI bus #1

SIOINT -- Legacy 8259 interrupt sources

PHY0INT*, PHY1INT* -- interrupts from the ethernet PHYs

Internal -- Discovery internal resources

Discovery lists the following physical pins:

CPUINT* -- interrupt output to "the" CPU

INT0* -- PCI 0 INTA# output (for agent mode, I assume)

INT1* -- PCI 1 " " "

MPP[x:y] -- INT[0:3]* outputs for CPU interrupt pins. CPU #1 could be one, I assume.

Answer (last updated January 11, 2001):
The GT-64260 should do all you are looking for:

External interrupts - All the GPP pins (32 pins) can be configured to be interrupt inputs.

Second interrupt pin to the CPU - You have two options here:

1 - You can use one of the PCI interrupt pins (If they are not used for

generating interrupts over the PCI). In this case All the interrupts logged in the main low and main high

registers (if not masked) can generate an interrupt to the 2nd CPU

2 - Use one of the GPPs as an interrupt output - in this case you need to

chouse which of the main interrupt registers is "connected to this interrupt output - main high or main low.

8. Internal Arbitration Control:

8.1 Example

Question:
Regarding the 'pizza arbiters' in the crossbar router. In the example in the documentation, The IDMA units are not included in the arbitration scheme. Does this mean that they can never connect, or are just considered to be on the bottom of the priority list

i.e. If none of the listed subsystems are requesting use of the crossbar, the IDMAs will be allowed in. In a similar manner, if the pizza slices only contained CPU, would any other system be connected, and if so, how would they be arbited?

Answer (last updated January 11, 2001):
The figure in the “Internal Arbitration Control“ section is just an example and not the default values, to see the defaults values please look in the appropriate registers. If a unit has no slice in the pizza arbiter it won't get connection. One can remove all slices for a unit only if it is not activated. In this way if all slices are for the CPU then no other interfaces will get connection.

9. Verilog:

9.1 CPU Interface

Question:
We are running the GT-64260 verilog with Bit 4 tied low (CPU Big Endian). I just want to make sure that I understand the connections so we wire it correctly on our schematics. PPC_A[0] on the 7400 is wired to PPC_A of the GT-64260 and so on (as in the reference schematic) right? If I simply define a 32 bit wire (as in the case of the PPC_A bus) and connect up from 7400 pin PPC_A to GT-64260 PPC_A pin, do I get a bit swap due to the declaration of the pin? Is that why we need to implicitly wire them up bit by bit (i.e. it is just a verilog thing?). It works but I want to understand why it seems logically different than the real life hookup (which I think is just a Verilog thing). Also, when I write GT-64240 registers I think I'll need to endian Byte swap it as well right?

Please explain how you work with the PPC convention.

Answer (last updated January 11, 2001):
The GT-64260 buses are in the PPC convention but declared in verilog using Galileo's own convention. This means that our bus is declared as [31:0] but 31 is the LSB and 0 is the MSB, so should give the PPC the address as written in the data sheet (1400_0000) but will see it on your wave as 0000_0028. The same goes with all other buses between the PPC and the GT-64260.

You are right about the bit swap. It is really a Verilog thing. If you connect whole buses it depends on how they where declared. By implicitly wiring bit by bit the connection becomes bit 0 to bit 0 and so on. This is for all the buses between the GT-64260 and the PPC.

You don't need to byte swap the data bus when working in big endian, BUT you will receive the data on the bus swapped according the endianess. Also note that the registers in the data sheet are described in little endian only so when you read an internal register you should see it byte swapped from what is written in the data sheet.

Here is an example of a particular GT-642260 register read-write sequence from the PPC perspective showing what you would see on the PPC buses for each case:

write(DISCO_ADDR, write_data); //Write to address DISCO_ADDR to set some functionality

read(DISCO_ADDR, response_data); //Read from address DISCO_ADDR to get some value

The write_data and response_data will always be the same regardless of what endianes you are using. Take the CPU_CONFIGURATION register at offset 0x000 as an example. Lets say that you want to write to it its default value. The default value of this register as stated in the data sheet is 000008ff this is with an assumption that the values that sampled at reset are:

1. Big endian - bit 12 is zero.

2. Normal address decoding - bit 18 is zero.

3. Clocks are asynchronous - bit 30 is zero.

Now if you read the value of this register in big endian you will see:

dh[0:31] (PPC data bus 0:31]) is ff080000.

dl[0:31] (PPC data bus [32:63]) is don't care.

Now (to continue with this example) if you want to change bit 30 to one (you should only change it by the sampling option or serial initialization) the value of the register in little endian will be 400008ff.

To write this value in big endian mode you should write ff080040 and on the bus you will see:

dh[0:31] (PPC data bus 0:31]) is ff080040.

dl[0:31] (PPC data bus [32:63]) is don't care.

Question:
I have the GT-64260 simulations working but I want to double check the bus wiring. I think some of the confusion revolves around the verilog world versus schematics but here it goes: The PPC_D bus is defined as [0:63] where bit 0 is the MSB. This is wired to GT-6426 DH and DL busses (both defined as [31:0]) and are connected as follows:

PPC_D[63:32] -> DL[31:0] and PPC_D[31:0] -> DH[31:0] which implies from a verilog perspective that Bit 31 is the MSB of the bus from the GT-64260's perspective. Now I know that Bit 0 is really the MSB which matches the way the reference design is connected as well as matches the bit-bit mapping from the PPC that I would expect. So, my assumption is that somehow, beneath the top level wrapper file of the GT-64260 model the PPC_D inputs are munged to reverse the bit order, no big deal.

If I look at the reference design you have a boot device hanging directly off of a buffered version of the device_ad bus on bits 7:0 so that would lead me to believe that this is the LSByte which doesn't track with the endian nature of the bus?

The simulation works (eventhough I don't understand the connections) and my assumption is that if I connect bit 0 of PPC to bit 0 of DH and so forth as well as hookup device_ad to my CPLD and assume that bits 31:24 are LSByte and so forth that I will be all set in the lab. It seems to me that my PLD would be wrong since this appears different than what I have in the testbench.

Is bit 0 of device bus MSB or LSB? From simulation it appears as if it is LSB but big endian definition would lead one to think it is MSB? Phrased another way, why is the 8 bit device of the reference design tied into bits [7:0] if it is a big endian bus structure? I would expect it to be tied to [31:24].

This in itself seems like a contradiction. If I am accessing an 8 bit device, bit 0 is the LSB. If I am accessing a 32 bit device, bits 7:0 become the MSByte?

Why would a 32 bit device provide its MSByte of response data on bits [7:0]? I understand that bit 0 is the LSB of your bus which will translate to PPC[63] on the PPC bus, but this doesn't make sense. I am assuming that device bus bit 31 is the MSB and bit 0 is the LSB. If I am performing a byte operation my logic should get and provide data from bits 7:0 of the device bus (bit 7 is MSB). If I am doing a 16 bit access, my logic gets and provides data on bits 15:0 of the device bus (bit 15 is MSB) and if I am performing a 32 bit operation my logic would get and proivde data on bits 31:0 with bit 31 being the MSB. From the last 2 E-mails it isn't clear to me that this is how this works.

Answer (last updated January 12, 2001):
You may be confused between the name convention of the bits in a bus and the byte order between big/little endianess. The PPC convention is when spreading a bus the most significant bit is 0, we are using the bus naming convention in which 0 is the list significant bit. Now the endianess is that the most significant BYTE is on the list significant bit lane, i.e. in PPC it will be on [56:63] but in our bus naming convention is on [7:0].

When connected to the PPC we are using the PPC convention but when connected to the device we are using our regular convention. Actually you can look at it as just how we call the bits in the bus and it is different between the PPC interface and the device and all other interfaces.

I think that I understand your confusion. It goes this way:

1. On the PPC bus bit 0 is the MSB.

2. In all other buses including the device bus bit 31/63 is the MSB.

So if you map the bits one to one bit 63 of the PPC bus is reflected to bit 0 on the device bus, this is the reason that you don't have problems in simulation. When using 8 bit device you need to use bits [7:0] (they are also duplicated on the

other bytes). Actually when using an 8 bit device endianes doesn't mater. If you would use a 32 bit device you should see on bits [7:0] the MSB byte. Note also that endianes is for byte order and not bit order.

9.2 Device Controller

Question:
Could you check the programming of the GT-64260 CS0 Device Bank Parameter register (address offset 45c) in my simulation? I wrote a 32'h002F_FFFF to this register to configure it to a 32 bit region however when I do a 32 bit access to the region the GT-6420 is doing a burst, which breaks the interface. I forced the READY low and high and it appears to me that the GT-64260 thinks it is a 64 bit region since it appears to be doing a burst of 2 (the BADR increments from 0 to 1, ALE stays low and device bus value doesn't change. Here is the write command to CS0:

Load_verify(address_base, 64'hAA55_00FF, 4, 64'hFFFF_FFFF_FFFF_FFFF);

Load_verify(address_base + 4, 64'h1122_3344, 4, 64'hFFFF_FFFF_FFFF_FFFF);

The data that comes across the device bus when BADR=0 is AA55_00FF and the data that comes across when BADR=1 is all 0's (which is why I think the GT-64260 thinks its a burst since this is what is on the MSBs of the PPC_Data bus on the 32 bit write).

Answer (last updated January 12, 2001):
This is the actual behavior of the GT-64260 device controller. Since the internal data bus is 64 bit wide the controller makes an pack/unpack operations when accessing the devices. In case the data is 4 byte or less the GT-64260 will still make two accesses but on the second the WR* signal won't be activated, so the write is not done although consuming a cycle. Please ensure that in that this is the case (second write is not activated).

GT-64240 FAQs

10. CPU Interface:

10.1 MIPS 64-bit Multiplexed Address/Data Bus Interface

Question:
The datasheet does not address what to do with Preq* when the uP does not have a SysAD bus mastership request line, nor what to do with the ternary cache lines when you're not using an L3 cache. I have tied Preq* and TcTCE* high, and TcMatch low, but then again I'd tied the SysRdyIn* lines high, and it turned out I was wrong. Can you get an answer for me on Preq*, TcTCE* and TcMatch when unused?

Answer (last updated January 11, 2001):
See QED's (PMC-Sierra's) external cache Application Note for QED's CPU for more details: “External Cache for the RM5271”, RM5271-AN1161010002.

Preq* - it is recommended to connect it to the CPU but when not available pulled high.

TcMatch - it is recommended to pull low. but when not using the external cache just do not set bit 14 (R7KL3) of the CPU Configuration register (Offset:0x000) and the GT64240 will not look on this signal (high or low).

TcTCE* - it is recommended to connect this signal to the cpu TcTCE* (ScTCE* depends on the cpu you are using), see page 9 of the appnote referenced above.

10.2 CPU Synchronization Barrier

Question:

Whenever you do a Memory Write, does the GT-64240 actually write it to the actual memory location immediately or does it store in a buffer some where and actually waits for a read operation and then writes it into the actual memory location. The problem i am facing is this :

Give a 2 GB SDRAM memory, upon doing 14 Front door (via Verilog and the given BFM) writes, and then doing back door reads (through Denali back door interface), I find the data to be the old data and it looks like the write transaction never took place. HOWEVER if throw in a front door read (from any address) before those back door reads, i see the new data. in the memory through back door.

Answer (last updated January 11, 2001):
I am not sure on how you are activating the GT-64240. By front door and back door do you mean accesses from CPU and from PCI? If yes then we have a posting fifo internally of the accesses and the data will go to the DRAM without need of reads, but the order between CPU and PCI is not defined and depends on the time of initialization the clocks ratio and internal arbitration. In some cases you might see new data and in some old one. If you need to ensure order you should use one of the synchronization mechanisms like sync barrier or others.

11. Device Controller:

11.1 Interfacing With 8/16/32-Bit Devices

Question:
I am accessing a device bus region configured as 32 bits wide in the following way: write to address 6000_0000 (CS0 region as programmed via S/W into the GT-64240) write to address 6000_000C. On the latched device bus (latched on ALE falling edge) and BADR, I am seeing the following on each access: latched_address[27:2] = 0, badr = 0. On second access I am seeing: latched_address[27:2] = 1, badr = 2.

The GT-64240 32 bit interfaces is hooked up as follows: {latched_addr[26:4], badr} as the address to the devices. This does not seem right.

Answer (last updated January 11, 2001):
In the GT-64240 rev 1.1 datasheet, there is the following table, that explains how to connect 32-bit device:

Table 149: 32-bit Devices

Connection	Connect...	To...
Device Address	BAdr[2:0] AD[27:4] ALE Latch Outputs	Device Address Bits [2:0] Address Latch Inputs Address LE Device Address Bits [26:3]
Device Data	AD[31:0]	Device Data Bits [31:0]
Device Control Pins	ALE AD[1] AD[0] AD[31:28]	Control latch LE Becomes DevRW* Becomes BootCS* Becomes CS[3:0]*
Write Strobes	Wr[0]* Wr[1]* Wr[2]* Wr[3]*	Device Data Bits[7:0] Write Strobe Device Data Bits[15:8] Write Strobe Device Data Bits[23:16] Write Strobe Device Data Bits[31:24] Write Strobe

As you can see from the table, the 32-bit device address is build up of:

DevAddr[2:0] is driven by the GT-64240 BAdr[2:0]

DevAddr[26:3] is driven by the latched GT-64240 AD[27:4]

GT-64240 AD[3:2] are not used for the 32-bit device address generation.

Now, when access offset 0x0, we expect DevAddr[26:0] to be 0x0, and this is what you get. When access offset 0xc, we expect DevAddr[26:0] to be 0x3. However, you get 0x2. As you already know, there is a restriction in the GT-64240 Device interface unit, that any access to a 32-bit device, is executed as a burst of two. So, in the case of access to offset 0xc, you get a burst, BAdr for the first dummy data is 0x2, and BAdr for the second valid data is 0x3 as expected.

By the way, although the GT-64240 AD[3:2] is not used, you might find it usefull as a workaround to the restriction. During the address phase (ALE falling), AD[27:2] are driven with the CPU address[28:3]. When CPU access offset 0xc, CPU address bit[3] is '1', and this is the reason why you see value of '1' on latched AD[2].

12. PCI Interface:

12.1 Data Endianess

Question:
Will the GT-64240 correctly byte-swap data from SDRAM even if the PCI access is not 64-bit aligned?

Answer (last updated January 11, 2001):
The GT-64240 PCI (target) interface will correctly byte-swap data even if your access is not 64-bit aligned. What happens is the GT-64240 always fetches 64-bit words from the SDRAM for transfer to the PCI bus. It performs the correct byte-swap internal and masks out the unwanted bytes (via CBE) before it ships it out to the PCI. Let say your DMA (of your PCI master) starts at byte 5. GT-64240 fetches the whole 64-bit word, byte 0 to byte 7, from the SDRAM, swap the bytes, then puts the data on the PCI bus with byte 0 to byte 4 masked out. The subsequent accesses will be aligned, until the last transfer. If the no. of byte transferred is even, then the last transfer will be unaligned, in which case GT-64240 handles it in the same fashion as described.

12.2 64-bit PCI Interface

Question:
I have initialized the PCI_1 mem low decode registers to enable PCIReq64*, But I dont see it getting asserted when I write into a PCI device's internal register(addr: 0xf083_4008). Since the PCI device responds only for 64bit transactions it is not asserting Trdy_1_. Also I see initially during configuration cycles, the upper 32bits of PCI_1 bus are "X"s and lower 32 bits have the actual data. I assume this is also related to PCIReq64.

I am using the same initialization routine which I have for assembler. PCI_0 is working fine and GT asserts PCIReq64* properly for PCI_0 read/write. Please let me know if need to set anything specifically for PCI Bus_1 to force PCIReq64*.

Answer (last updated January 11, 2001):
In order to operate in 64 bit you must configure the PCI bus. For PCI0 use the 64en0 pin but for PCI1 use the REQ641* pin. This pin should be sampled low during reset.

Question:
If Force Req64* is enabled, all pci transactions initiated by the GT-64240 will be 64bits even when data is less than 64bits? Does using this feature affect pci config cycles as well?

Let's assume one of the GT-64240's dma channel is programmed to read 10 bytes of data starting from sdram address 0x0000...07 (not quadword aligned), and destination address is in PCI space. And DestHold=1 in control register, so all 10 bytes will be transferred to the same pci address.

After data is read from sdram to GT-64240's internal buffer, I am assuming the following write transactions will take place for transferring data.

Let's assume data in sdram is as follows, and dma source address is 0x00...07 :

address	data [0..63] in sdram
0x00...00	B0 B1 B2 B3 B4 B5 B6 B7
0x00...08	B8 B9 B10 B11 B12 B13 B14 B15

0x00...10 B16

So, with ForceReq64 feature on, and no swapping in GT, following transaction will take place on PCI:

1. 64bit write: pci[63:0] : B7 B8 B9 B10 B11 B12 B13 B14 B15

2. 64bit write: pci[63:0] : x x x x x x B16 B17 (only 2 bytes are enabled on bus, and B16 and B17 are on pci_bus[15:0])

Is this correct?

Answer (last updated January 12, 2001):
Normally, GT's PCI will drive transactions of or less than 64-bit as two 32-bit transactions (no gain by driving 64-bit since one cycle is wasted for waiting the ACK64). If you have the PCIReq64 bit enabled, then all transactions to PCI will assert Req64*. This is set in all the cpu to PCI_x Mem x Low decoders, the DMA Channel x Control registers, the MPSCx and Ethernet x Address Control registers. The target is expected to return Ack64*.

It is there for the support of Big Endian 64-bit PCI. When a master puts out a big-endian 64-bit transaction, the lower double word (bytes 0-3) is on the upper half of the PCI bus (AD[63:32]). If the target fails to returned an ACK64, however, the data is expected on the lower half of the bus (AD[31:0]). This is problematic, so for big-endian support, GT is forcing all transactions to be 64-bit.

This should not affect configuration cycles.

13. IDMA Controller:

13.1 IDMA Channel Control

Question:
Let's assume one of the GT-64240's dma channel is programmed to read 10 bytes of data starting from sdram address 0x0000...07 (not quadword aligned), and destination address is in PCI space. And DestHold=1 in control register, so all 10 bytes will be transferred to the same pci address. After data is read from sdram to GT-64240's internal buffer, I am assuming the following 3 write transactions will take place for transferring data.

1. 64 bit write, with only one byte enabled, cbe[7]=0, cbe[6:0]=1

2. 64bit write , with all bytes enabled, cbe[7:0]=0

3. 64bit write, with only one byte enabled, cbe[0]=0, cbe[7:1]=1

Or does the GT-64240 perform 32bit write for step#1 and #3 , and place data to lsb always?

Would the alignment of destination address change any of this?

Answer (last updated January 12, 2001):
To transfer 10 byte you should use Burst Limit of 8bytes. Now if you use hold, the destination address must be 8 byte aligned. In this case the reads will be:

1. 1byte read.

2. 8byte read.

3. 1byte read.

But the writes on the PCI will be:

1. 8 byte write.

2. 2 byte write.

In this case the source/destination alignment won't change.

Our PCI master policy is to drive transactions of up to 8 bytes as 32-bit PCI (don't assert REQ64*), so the 8byte write is executed as a burst of two 32-bit words.

If you use the Force Req64* feature, the master will drive it as a 64-bit transaction. Please see the “64-bit PCI Interface” section of the data sheet for more information regarding this feature.

Question:
It is mentioned in the data sheet that:burst limit should be smaller than byte count in the descriptor. We will probably use 64bytes as burst limit. How will this work for packets smaller than 64 bytes? Also, how will this affect the residual data:i.e if packet is 100 bytes, the second dma burst is only 36bytes long?

- ChanAct bit:Is this bit set after the completion of the entire DMA or after a burst transfer?

Answer (last updated January 3, 2001):
The system shall attempt to DMA a full packet. In this case, the byte count shall be programmed to a higher value than the burst limit. At the end, when there is some "left over", this amount shall be transferred to the destination unit. This last transfer shall be a smaller transfer than the burst limit. No problem with that.

The ChanAct bit indicates the time that the DMA engine is Active in reading data into it's FIFOs. While writing from the FIFO to the destination internal unit or when idle, this bit shall indicate that the port is inactive.

14. AC Timing:

14.1 Restrictions

Question:
I am trying to produce a 100 MHz or 133MHz clock for Tclk and Sysclk much the same as your reference design. The problem I am having is insuring a maximum skew between Tclk, Sysclk, and the SDRAM clks when all are synchronized at 100 MHz or 133MHz.

Answer (last updated January 10, 2001):
If all the clocks are synchronized, then TClk drives all of the internal clocks in this case and Sysclk just needs to be pulled high or low.

Question:
If TClk and Sysclk are synchronized, must Sysclk be pulled up or down or can it still have an active input clock?

Answer (last updated January 10, 2001):
The TClk and Sysclk synchronized means there is no restriction on the Sysclk input (as long as it is not floating).

Question:
What about the skew between TClk and the SDRAM clocks? Since the clock for the SDRAM is generated externally , there must still be a max skew between it and TClk. What is this maximum skew?

Answer (last updated January 10, 2001):
This maximum clock skew must be calculated at the system level using the GT-64240’s AC timing numbers, SDRAM AC timing and your printed circuit board signal fly time.