FREQUENTLY ASKED QUESTIONS GT-640xx, GT-641xx, GT-960xx and GT-961xx Products

January 12, 2001
FAQ-11

INTRODUCTION

This document provides FAQs or Frequently Asked Questions (and answers) for the GT-640xx, GT-641xx, GT-960xx and GT-961xx devices. This document replaces “FREQUENTLY ASKED QUESTIONS GT-64xxx System Controller Products” dated March 09, 2000.

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-64011, 64111 & GT-64115 FAQs

1. Overview:

1.1 Differences Between the GT-64011 and GT-64111

Question:
What are the differences between the GT-64111 and the GT-64011 devices?

Answer (last updated March 9, 2000):

There are minimal difference between the GT-64111 and the GT-64011. The GT-64111 pinout is the same as the GT-64011, except the power pins become 3.3V and the Vref pin becomes the Vio on the PCI bus. Following are changes to GT-64011 based designs:

1) Vcc = 3.3V. All I/O is 5V tolerant. Universal PCI support (3.3 or 5V).

(2) Valid Out* pulled up to CPU's Vcc (via 4.7Kohm):

An additional 4.7Kohm pull-up is required on the ValidOut* signal from the CPU to the GT-64111. This often already exists on many GT-64011 based designs. The GT-64111 detects the cpu SysCmd type (IDT/QED or NEC) on first valid command from the cpu. If ValidOut* is floating on power up, then the GT-64111 may incorrectly read invalid cpu command and configure itself in the wrong mode.

(3) Vref for CPU becomes Vio on PCI:

Connect the Vio directly to 3.3V, 5V or the Vio pin on the PCI bus. The voltage divider used to generate a 4.0V Vref on the GT-64011 may be used to provide a zero ohm stuffing option to support 3.3V or 5V Vio on the GT-64111.

(4) CPU Clock may run up to 66MHz, the PCI Clock may run up to 66MHz. Existing GT-64011 clocks may be used to drive the GT-64111. The PCI Status and Command Register 0x004 66MhzEn bit 21 is set to 1 to advertise 66MHz PCI capability.

(5) The GT-64111 consumes 50% less current (one third the power) compared to the GT-64011.

2. CPU/Local Master Interface:

2.1 CPU/Local Master Interface

Question:
What is the relation between RST signal and CPU reset ? is it important to finish GT-64111 reset before we release the CPU reset? Can we release RESET of the CPU with the GT-64111 chip?

Answer (last modified December 12, 2000):
Let me describe how we reset the GT-64111 on our evaluation board (EV-64111). RST* is received from the PCI. This signal goes directly into the GT64111 and into the reset PAL. The PAL then generates the Reset Sequence for the CPU and releases the CPU ColdReset and WarmReset after about ["SystemClk" divided by 256 (which is "Mode Clock") and this is again divided by 256 Mode clocks] (See PAL equations). Bottom line: GT64111 is released from Reset (i.e. Rst* rising edge) about 1.5mS before CPU.

The relation between the GT64111 reset and CPU reset should comply with the requirement listed in chapter 3.8 of the GT-64111 spec: "The CPU should not attempt to access the GT64111 before 10 Tclk Cycles from the deassertion of the RST* signal has expired."

3. Address Space Decoding:

3.1 PCI Decoding Process

Question:
What is the purpose of the GT-64115 PCI bank size registers? It seems like the PCI Bars do the same thing? What is the purpose of the PCI I/O bank size registers? It seems like the PCI Bars do the same thing?

Answer (last updated January 12, 2001):
The purpose for the PCI bank size registers is to have adjustable PCI memory windows. The limit on each of the PCI memory windows is 256MByte. Since there are two of these windows, it enables lager PCI memory address space (512Mbyte). PCI I/O size is the same but for I/O address space.

3.2 Address Space Decoding Errors

Question:
I can not set "7F" to 0x14000060 and only can set up to "3F". Is it mean GT64111 can not support to 256MByte?(Bit 27 can not compared) If set "7F",GT64111 be hang up.

Answer (last modified December 12, 2000):
GT64111 does support 256Mbyte, but in order to perform the action you are attempting to do, you must move Cs[0..3] and BootCs from that area. If not, this may cause a double hit in the GT64111, as both the PCI and the Cs will answer in this area.

Question:
GT64111s 0x14000060 register defaule value be 0x1f and it support to 64MByte (not 32MByte as shown in the data sheet).

Answer (last modified December 12, 2000):
True, this a mistake in the data sheet “Address Space Decoding” section’s Default Memory Map.

3.3 PCI Address Remapping

Question:
There are four GT-64115 registers that I do not know how to configure:

- PCI SCS[1:0] BASE AR (0xc48) They have 0x0
- PCI SCS[2:3] BASE AR (0xc4C) They have 0x0
- PCI CS[2:0] BASE AR (0xc50) They have 0xe4000000
- PCI CS[3] & Bootcs AR (0xc54) They have 0xe6000000

Should they match the BAR? I am still struggling with Base Address Remaps. Do we have any examples to review?

Answer (last updated January 12, 2001):
These registers are duplicate of the BASE AR when you rewrite the BASE AR. This is the example on how I would use them. For example: If I want to put an address 0x12000000 on the PCI bus and I want this address to let me access the memory at location 0x0. When I change the BAR to 0x12000000 the offset 0xc48 also have 0x12000000. Since I want to access the memory at 0x0, I would reset 0xc48 to 0x0. So when GT see the 0x12000000, it will accept this transaction and set this equal to 0x0.

4. Memory Controller:

4.1 Memory Controller

Question:
How long are the time slices on the GT-64115 Memory Controller arbiter? Lets say the PCI interface moves data and it is 2K bytes to local memory and the CPU wants to access memory. Is it allowed to move all data and stall the CPU? Can this duration be programmed?

Answer (last updated January 12, 2001):
This is not the time slice arbiter and it is not programmable. As your example above if there is the CPU request for memory access, the PCI will finish it current FIFO and give the access to CPU and If CPU also has a long burst then CPU will finish with it FIFO and give back the control to the PCI and the whole thing will start all over again until all done.

Question:
Can GT-64115 Memory Controller priorities be changed? I only see priority assignments in the DMA controller.

Answer (last updated January 12, 2001):
This is a round robin arbiter so there is no priority needed or supported. The priority in the DMA controller is for the prioritize among the DMA engines not for the GT-64115 Memory Controller arbitor.

4.2 Device Controller

Question:
What is the byte order on the GT-64115 device bus for the unpacking of data from the SysAD bus?

Answer (last updated February 1, 2000):
The unpacking process takes into consideration the GT-64115 endianess (which is the same as the CPU endianess) and the device width.

Examples:

a) CPU LITTLE endian:

Data from CPU   3 2 1 0

unpacking towards 8 bit device: 0, 1, 2, 3 (Where 0 = first, 3 = last)

unpacking towards 16 bit device: 10, 32 (Where 10 = first, 32 = last)

write towards 32 bit device: 3210

b) CPU BIG endian:

Data from CPU   0 1 2 3

unpacking towards 8 bit device: 0, 1, 2, 3 (Where 0 = first, 3 = last)

unpacking towards 16 bit device: 01, 23 (Where 01 = first, 23 = last)

write towards 32 bit device: 0123

5. DMA Controllers:

5.1 DMA Channel Registers

Question:
What is the maximum number of bytes a GT-64111 DMA channel may be configured to transfer?

Answer (last updated February 1, 2000):
The GT-64111 data sheet (section 7.1 DMA Channel Registers) mentions that “The maximum number of bytes a DMA channel can be configured to transfer is 64K.” The Byte Count Register 0x800 - 0x80C is programmable with a 16-bit field. Actually, the largest value that can be programmed (0xFFFF) will result in a maximum transfer length of 64K-1 (65,535) bytes.

5.2 Design Information

Question:
When working in chain mode DMA, should the start address dma internal register in the GT-64111 contain the virtual address that the MIPS CPU knows or the physical address defined by the hardware?

Answer (last modified December 12, 2000):
Always use PHYSICAL addresses, the MIPS adds an artificial offset of 0xA0000000 to kseg space, so the MIPS address is for example 0xB4000000 when you want to access 0x14000000.

Question:
In FlyBy DMA the device chip select signal does not seem to function. Is this a GT-64115 bug?

Answer (last modified December 15, 2000):
The GT-64115 was desinged like this. In demand mode, the GT-64115 doesn’t drive the CS_ signal, but the DMAack_ signal is driven which enables the following work around:

The application should drive the CS_ signal from the CStiming_ and DMAack_ signals:

CS[1]_ = CStiming_ or DMAack[1]_ (CStiming_ || DMAack[1]_).

6. PCI Bus:

6.1 PCI Master Operation

Question:
What is GT64111's PCI bus politics when GT64111 DMA accesses the PCI bus? It will release the bus until DMA access is over or one PCI burst access is over ?

Answer (last modified December 12, 2000):
In general, the 64111 will burst a SINGLE TRANSACTION and then free the PCI for another agent to use the bus.

6.2 PCI Target Interface

Question:
Any pointers on setting up the GT-64115 PCI Timeout& retry registers for optimal performance? I understand it varies across applications but am looking for help here.

Answer (last updated January 12, 2001):
Most customers just set these fields is to their maximum value to avoid retires on the PCI bus.

6.3 PCI Parity Support

Question:
Will the GT-64111 complete a PCI master cycle normally if it detects a parity error?

Answer (last updated February 1, 2000):
The PCI specification (revision 2.1) indicates that a master may either complete the cycle or terminate it, although it recommends letting it complete. The GT-64111 as a PCI master completes a PCI cycle normally even when there is a parity error.

7. Reset Configuration:

7.1 Reset Configuration Table

Question:
In the data sheet, Reset Configuration of GT-64111, pin DAdr[7] controls the 'External Latches Presence' configuration. What are these latches? Are these address latches or data latches ?

Answer (last modified December 12, 2000):
The latches are needed for 64 bit wide interfaces. This means that for these systems latches are needed for the data lines. They are also present on the Dadr lines for interleaving between the DRAMs. See the illustrations in the "application" section of the datasheet.

7.2 PLL Application Notes

Question:
What are the default PLL settings of the GT-64115?

Answer (last updated February 1, 2000):
The GT-64115 PLL setting is defined by pull-ups or pull-down resistors sampled at reset on the MPP[3:0] and CS_Timing pins. The default values are shown in the GT-64115 data sheet Reset Configuration chapter 10.

Since the PLL is programmable it may be useful to provide a dual layout allowing for pull-ups or pull-downs on each pin. In case any board specific timing issues occur, providing a stuffing option for either a pull-up or pull-down resistor on each pin may be useful.

8. Big and Little Endian:

8.1 Background

Question:
We are using the GT64111 with IDT MIPS 4640. Both devices are configured to big indian. When we write and read to the 64111 internal registers of the dma, we notice that they are in little indian. Is this fact true, and are the 64111 internal registers always in little indian regardless to big/little indian configure?

Answer (last modified December 12, 2000):
The word is endian. (An indian is a person that comes from India, or a native American). The internal registers are always accessed in little endian, even if the CPU is in big endian.

8.2 Configuring a System for Big and Little Endian

Question:
No matter if GT-64111 is in big endian or little endian mode, connecting 8 bit memory device will be to to AD[7:0], and connecting 16bit memory device will be to AD[15:0]. Is this correct ?

Answer (last modified December 12, 2000):
Yes.

Question:
The CPU (in big endian mode) writes one byte to an 8 bit memory device and puts data on SysAD[31:24]. The GT64111 will write this byte on AD[7:0]. Is it correct ?

Answer (last modified December 12, 2000):
Yes.

Question:
The CPU (in big endian mode) writes one byte to a 32bit memory device and puts data on SysAD[31:24], GT64111 will write this byte to AD[31:24]. Is it correct ?

Answer (last modified December 12, 2000):
Yes.

9. Connecting the Memory Controller to DRAM and Devices:

9.1 Working Without Data Latches

Question:
If GT64111 is configured to big endian through interrupt* pin, and one of the memory device is 8bit wide, should I connect device d0~7 to AD[7:0] ? or to AD[31:24]?

Answer (last modified December 12, 2000):
AD[7:0]

10. Evaluation Boards:

10.1 BOM

Question:
We are designing an I/O add-in card for your EV64111 evaluation board, that whould be connected to P4 and P5 SMT connectors. Could you indicate which manufacturer and part number is the parts you use for P4 and P5 connectors in EV64111?

Answer (last modified December 12, 2000):
The connectors is manufactured by amp and are called 0.05" (1.27mm) 2x30 SMT. cvilux and samtec and probably 10 other vendors have the same part.The part numbers are:

female AMP 104652-6 or CVILUX CB55-60AV100

male AMP 104656-6

male SAMTEC (very high) TFM-130-32-SDA

10.2 PCI Bus Access

Question:
Why won't the EV-64115 access the PCI as a bus master? The EV-64115 evaluation board works except for GT-64115 access to PCI. The GT-64115 will not even assert Req# on the PCI bus. Why?

Answer (last updated February 1, 2000):
Enable the GT-64115 as a PCI Master. This feature is programmable by setting bit 2 “MASEn” of the PCI Configuration register located at 0x004 to '1'.

On the Galileo website there is an application note “Basic Examples with IDT/sim on Galileo Eval Boards” that details configuration of Galileo system controllers to access the PCI space. This application note is located in the GT-64010A section of the Technical Library.

10.3 85C30 UART

Question:
Why do sequential accesses to the 85C30 lock up the Galileo-9 (GT-64011) evaluation board?

Answer (last updated February 1, 2000):
The 85C30 (10MHz) requires a 240nS delay between consecutive reads & writes.

The CPU may execute NOPs to delay sequential accesses to the UART. The 16MHz 85C30 provide improved performance over the 10MHz version, but may also require delays in software.

A PLD could de-assert the GT-64111 Ready* signal insuring sequential access to the UART does not violate the 85C30 specification.

10.4 PLD Reprogramming

Question:
How is the EV-64115 PLD reprogrammed? The GT-64115 BSP documentation says rev 1.01 PLD is required on the EV-64115 eval board, but we were shipped a board with the rev 1.0 PLD. What should we do?

Answer (last updated February 1, 2000):
In most cases the existing PLD on the eval board works for evaluation purposes. Evaluation of the GT-64115 BSP is the exception. Since the PLD is not socketed, an Altera byte blaster + MAX 8.0 or higher is required to reprogram the PLD through the 10 pin connector marked “con 1".

10.5 Software Support

Question:
What Win32 drivers are available for the GT-64115 for PC applications?

Answer (last updated February 1, 2000):
KRF Tech has a product line that automates and simplifies writing device drivers for many operating systems. Galileo has worked with KRF Tech to develop drivers for Galileo system controllers including the GT-64115. More information is available at KRF Tech's website www.krftech.com.

GT-64120 & GT-64120A FAQs

11. Evaluation Boards:

11.1 Product Description - Bill Of Materials

Question:
Does the EV-64120 schematics have pull-ups on the SysAD[63:0] bus or on the AD[63:0] bus?

Answer (last updated March 9, 2000):
The EV-64120 evaluation board schematic has pull-ups on the SysAD bus signals. During the delay between the release of the GT-64120 and CPU from reset, the SysAD bus signals are not driven by the CPU. If these signals are inputs to the GT-64120 and allowed to float (no pull-ups), problems may result.

Pull-ups are also required on the SysAD bus for applications that do not have a local CPU installed. There is a section in the GT-64120 data sheet that describes how the CPU bus should be terminated. Please see chapter 14 “Using the GT-64120 Without the CPU Interface”.

The EV-64120 evaluation board schematic does have pull-ups on the AD bus, on the rdata[63:0] signals. The purpose of these pull-ups is to protect the 501 transceivers from receiving a floating input. The “bushold” technology for buffers and transceivers may be used to eliminate the need for pull-ups / pull-downs. If bushold technology is used, be sure to check the application for any potential conflicts with reset configuration pull-ups / pull-downs in the system.

11.2 Product Description - Schematics

Question:
Why are GT-64120 pins U26(VREF_0) and L24(VREF_1) connected to diodes on the EV-64120-MBD schematics?

Answer (last updated February 2, 2000):
On sheet 11 of the EV-64120-MBD schematics, GT64120 pins U26(VREF_0) and L24(VREF_1) are connected through clamping diodes to a supply voltage.

There is an errata FEr#3 “Wrong PCI Pads” on all steppings of the GT-64120 device. These diodes were added to the layout of the EV-64120-MBD printed circuit board. System level testing indicates that there are no negative impacts to the operation of these devices without external clamping diodes. These diodes are no longer required on the schematic.

11.3 Product Description - PAL Equations

Question:
We bought 3 EV-64BP (5V) from you. When putting these boards into operation with a Galileo EV-64130-MPC603e board, an INTEL PRO/100+ ethernetcard and a self developed PCI card, we have some problems with the 64bit PCI slots.

It seems that there is a problme with the PCI slot #6 (IDSEL 3(silkscreen) or DEV 6). It looks like that these slot could ONLY be used from the EV-64130-MPC603e board. In all other cases, the _os_pciFindDevice function (provides from YOUR BSP) does never detect any device inserted in that slot. WHY??

Of cours you could say: Then use it in the one and only working configuration, but then we have some problems that we can't make measurements in our hardware.

I think all slots (64bit) should be identical and "exchangeable".

Answer (last updated January 3, 2001):
The problem is with Intel NIC card. The cause of the problem is after power up the BP64's arbiter park the GNT# on IDSEL_3 slot. Some of Intel's NICs can not handle GNT# without issue REQ# first. Attached here are the PAL MAX+II files so you can modify the code or the pinout to change the GNT# that will be issued after power up.

```
% PLD Function:  PCI 32 Bit BUS arbiter
  Altera part number: 7032-10 3.3V
  Circuit board:      PCI 64 Bit Motherboard
  Edit history:

DateEngeneerRev changeRev
    ----------------------        -----
11 May 96M. HaskoInitial Release     1.1
    16 JAN 97   M. Hasko    Included timeout    1.2
20 NOV 97RONEN SIMA\TAL GIL64 Bit RESET HANDLER1.3
15 feb 98 NoamL / RonenS146H GNT1 must be in one for 64bit1.4
%

TITLE "PCI 32 Bit Arbiter";

CONSTANT TIME_OUT = 15;% Time-Out between Grant change enable to FRAME assert.In PCI clocks %

CONSTANT NUMBER_OF_REQ = 5; % Number of agents which may request grant %


SUBDESIGN pciarb32
(
%inputs%

_PCIrst :INPUT;
PCIclk:INPUT;

_REQ0:INPUT;% REQ32_0 (P2) (PIN 5)  %
```

```
_REQ1:INPUT;% REQ32_1 (P1) (PIN 6)  %
_REQ2:INPUT;% REQ32_2 (P3) (PIN 8)  %
_REQ3:INPUT;% REQ32_4 (P6) (PIN 11) %
_REQ4:INPUT;% B_REQ32      (PIN 13) %

_FRAME/_REQ64:BIDIR;% (PIN 14) _FRAME ON 32 BIT BUS MODE , REQ64 ON 64 BIT BUS MODE ( DURING
RESET ) %
_IRDY:INPUT;
_TRDY:INPUT;
_STOP:INPUT;
_LOCK:INPUT;

_FR/REQ64oe:INPUT;% PIN 2 - OEN2#    CONTROLS THE O.E. OF FA/_REQ64_tri  %
ParkOnLast:INPUT;

_64BusEn:INPUT;% WHEN ASSERTED THE PAL FUNCTIONS DURING RESET ONLY ( J19 )(PIN 36)%
Mode1:INPUT;
Mode2:INPUT;
Mode3:INPUT;
Mode4:INPUT;
ARB_CON1:INPUT; % CONNECTION BETWEEN THE TWO PALS (PIN 27) %

_REQ32_5:INPUT;%pin 31- UNUSED%
_REQ32_6:INPUT;%pin 29- UNUSED%
_REQ32_3:INPUT;%pin 9- UNUSED%


%outputs%

_GNT0:OUTPUT;% (PIN 7) %
_GNT1:OUTPUT;% (PIN 12) %
_GNT2:OUTPUT;% (PIN 40) %
_GNT3:OUTPUT;% (PIN 41)  %
_GNT4:OUTPUT;% B_GNT (PIN 38) %

_GNT32_5:OUTPUT;%pin 37 - UNUSED%
_GNT32_6:OUTPUT;%pin 34- UNUSED%
```

```
_GNT32_3:OUTPUT;%pin 39- UNUSED%



ARB_CON0:OUTPUT; % CONNECTION BETWEEN THE TWO PALS (PIN 26) %



_OEdriver:OUTPUT;% (PIN4) SHORTED ON THE BOARD WITH PIN 2 (OEN2#) , CONTROLS THE O.E. OF FA/
_REQ64_tri %
)

VARIABLE

GNTstate: MACHINE OF BITS (GNTq[2..0]) WITH STATES
(
Master0,
Master1,
Master2,
Master3,
Master4
);

EnCHstate: MACHINE OF BITS (enCHq[1..0]) WITH STATES
(
   NO_REQ,
WAIT_CYCLE,
FRAMED
);

AnyREQ :NODE;
   EnChange :NODE;
_FRAME  :NODE;
InFrame   :DFF;
   MaskGNT :DFF;
FrameDeasserted  :NODE;
FR/_REQ64_tri:TRI;
   TimeOut[3..0]    :DFF;
```

```
req_ff1:DFF;
req_ff2:DFF;

BEGIN
   DEFAULTS
MaskGNT = GND;
ARB_CON0 = GND;
END DEFAULTS;
AnyREQ = !_REQ0 # !_REQ1 # !_REQ2 # !_REQ3 # !_REQ4;
EnChange = (EnCHstate == FRAMED) # (EnCHstate == NO_REQ) # (TimeOut[] == TIME_OUT);

_GNT32_5= VCC;
_GNT32_6= VCC;
_GNT32_3= VCC;

% **** LOGIC FOR THE 64 Bit BUS **** %
req_ff1.clrn=_PCIrst;
req_ff2.clrn=_PCIrst;
req_ff1.clk=GLOBAL(PCIclk);
req_ff2.clk=GLOBAL(PCIclk);
req_ff1.d=VCC;
req_ff2.d=req_ff1.q;



FR/_REQ64_tri.in = req_ff1.q ;
FR/_REQ64_tri.oe =  GLOBAL(!_FR/REQ64oe) ; % ACTUALY FR/_REQ64.oe = _OEdriver ( THE ! IS NECECERY TO
IMPLIMENT THE
 TRI STATE , NOTE THAT WE PUSH A LOW ACTIVATED SIGNAL BUT IT IS NOT
 INVERTED BY THE ! , IT IS ONLY A SYMBOL TO THE COMPILER ) %
_FRAME/_REQ64 = FR/_REQ64_tri.out ;
IF ( _64BusEn == 0 ) THEN   % 64 Bit BUS MODE  - DRIVING REQ64 DURING RESET %
_OEdriver = req_ff2.q;
_FRAME = !_64BusEn;

ELSE
_OEdriver = VCC;  % HIGH Z THE TRI %
_FRAME = _FRAME/_REQ64 ;
```

```
END IF;
% END OF LOGIC FOR THE 64 Bit BUS %

InFrame.clk = GLOBAL(PCIclk);
InFrame.clrn = GLOBAL(_PCIrst);
InFrame = !_FRAME;
FrameDeasserted = InFrame & _FRAME;

   TimeOut[].clk = GLOBAL(PCIclk);
TimeOut[].clrn = GLOBAL(_PCIrst);

   MaskGNT.clk = GLOBAL(PCIclk);
   MaskGNT.clrn = GLOBAL(_PCIrst);

% Arbitration state machine %
!_GNT0 = (GNTstate == Master0) & GLOBAL(_PCIrst) & !MaskGNT   & _64BusEn;
!_GNT1 = (GNTstate == Master1) & GLOBAL(_PCIrst) & !MaskGNT   & _64BusEn;
!_GNT2 = (GNTstate == Master2) & GLOBAL(_PCIrst) & !MaskGNT   & _64BusEn;
!_GNT3 = (GNTstate == Master3) & GLOBAL(_PCIrst) & !MaskGNT   & _64BusEn;
!_GNT4 = (GNTstate == Master4) & GLOBAL(_PCIrst) & !MaskGNT   & _64BusEn;

EnCHstate.clk = GLOBAL(PCIclk);
EnCHstate.reset = GLOBAL(!_PCIrst);

CASE EnCHstate IS
WHEN NO_REQ =>
TimeOut[] = 0;
IF (FrameDeasserted) THEN
EnCHstate = FRAMED;
ELSIF (AnyREQ) THEN
EnCHstate = WAIT_CYCLE;
ELSE
EnCHstate = NO_REQ;
END IF;

WHEN WAIT_CYCLE =>
         IF (TimeOut[] == TIME_OUT) THEN
```

```
TimeOut[] = TimeOut[];
ELSE
TimeOut[] = TimeOut[] + 1;
END IF;

IF (FrameDeasserted) THEN
EnCHstate = FRAMED;
ELSE
EnCHstate = WAIT_CYCLE;
END IF;

WHEN FRAMED =>
TimeOut[] = 0;
IF (AnyREQ) THEN
EnCHstate = WAIT_CYCLE;
ELSE
EnCHstate = NO_REQ;
   END IF;

END CASE;


GNTstate.clk = GLOBAL(PCIclk);
GNTstate.reset = GLOBAL(!_PCIrst);

CASE GNTstate IS
WHEN Master0 =>
IF (!_REQ1 & EnChange) THEN
GNTstate = Master1;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ1 & !_REQ2 & EnChange) THEN
GNTstate = Master2;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ1 & _REQ2 & !_REQ3 & EnChange) THEN
GNTstate = Master3;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ1 & _REQ2 & _REQ3 & !_REQ4 & EnChange) THEN
```

```
GNTstate = Master4;
MaskGNT = _FRAME & _IRDY;
ELSE
GNTstate = Master0;
END IF;

WHEN Master1 =>
IF (!_REQ2 & EnChange)THEN
GNTstate = Master2;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ2 & !_REQ3 & EnChange) THEN
GNTstate = Master3;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ2 & _REQ3 & !_REQ4 & EnChange) THEN
GNTstate = Master4;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ2 & _REQ3 & _REQ4 & !_REQ0 & EnChange) THEN
GNTstate = Master0;
MaskGNT = _FRAME & _IRDY;
ELSE
GNTstate = Master1;
END IF;

WHEN Master2 =>
IF (!_REQ3 & EnChange)THEN
GNTstate = Master3;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ3 & !_REQ4 & EnChange) THEN
GNTstate = Master4;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ3 & _REQ4 & !_REQ0 & EnChange) THEN
GNTstate = Master0;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ3 & _REQ4 & _REQ0 & !_REQ1 & EnChange) THEN
GNTstate = Master1;
MaskGNT = _FRAME & _IRDY;
ELSE
```

```
GNTstate = Master2;
END IF;

WHEN Master3 =>
IF (!_REQ4 & EnChange)THEN
GNTstate = Master4;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ4 & !_REQ0 & EnChange) THEN
GNTstate = Master0;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ4 & _REQ0 & !_REQ1 & EnChange) THEN
GNTstate = Master1;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ4 & _REQ0 & _REQ1 & !_REQ2 & EnChange) THEN
GNTstate = Master2;
MaskGNT = _FRAME & _IRDY;
ELSE
GNTstate = Master3;
END IF;

WHEN Master4 =>
IF (!_REQ0 & EnChange) THEN
 GNTstate = Master0;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ0 & !_REQ1 & EnChange) THEN
GNTstate = Master1;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ0 & _REQ1 & !_REQ2 & EnChange) THEN
GNTstate = Master2;
MaskGNT = _FRAME & _IRDY;
ELSIF (_REQ0 & _REQ1 & _REQ2 & !_REQ3 & EnChange) THEN
GNTstate = Master3;
MaskGNT = _FRAME & _IRDY;
ELSE
GNTstate = Master4;
END IF;
```

```
WHEN OTHERS =>
GNTstate = Master0;



END CASE;

%STAM = _TRDY & _STOP & _LOCK & Mode0 & Mode1 & Mode2 & Mode3 & Mode4;%

END;
```

11.4 Getting Started - Jumpers & Headers

Question:
PMON on our EV-64120 PCI evaluation board doesn't come up if jumper J26 is connected. It's my understanding that this jumper connects the 64120's Interrupt* pin to the CPU's Int*[2] pin. On the ev-64120 manual the default state of this jumper is "out" (disconnected). When this jumper is "in" and the board is powered up no indication of life (PMON activity) is observed. Are there any tricks to get the 64120 interrupt controller connected to a CPU interrupt line on the ev-64120 evaluation board?

Answer (last updated January 3, 2001):
The problem is that the EV-64120 has a pull down on the interrupt pin going to the GT-64120 to configure it to Big Endian. We have another pull, this time a PullUp, connected to the int5* pin of the CPU, for cases that this pin is not driven, i.e. the jumper is not connected. The problem is that when you connect the jumper you are actually connecting the two pulls. Now there is a conflict that results in a bad sampling at reset by the GT. To solve this you must disconnect R169 - the PullUp connected to the CPU. This shall result in pulling down during reset, but after reset is deasserted the 120 shall drive this pin to high (unless an interrupt is pending and it is unmasked...).

Question:
On the EV-64120 evaluation board I want the GT-64120 INTERRUPT* pin to drive one of the CPU interrupts. What modification to the EV-64120 evaluation board is required? The CPU and GT-64120 are configured for big endian mode.

Answer (last updated March 9, 2000):
The EV-64120's J32 header is provided to connect a variety of interrupt signals to any of the CPU's interrupt inputs. Included in this header is the GT-64120 INTERRUPT* signal. The J32 header default configuration does not connect any interrupt signal to any of the CPU interrupt inputs. Each of the CPU interrupt inputs are pulled high (non-active) on the evaluation board.

When Reset is de-asserted and the GT-64120 is initializing, the GT-64120 samples (as an input) the INTERRUPT* signal. This pin is pulled high or low by J7 to configure the GT-64120 endianess (which should be the same as the CPU endianess). The INTERRUPT* pin is pulled high for little endian or pulled low for big endian configuration.

If the INTERRUPT* pin is connected to one of the CPU interrupt inputs and if the GT-64120 is configured to big endian, then the pull-down on J7 will conflict with the pull-up on the respective CPU interrupt signal. The following procedure will prevent this conflict:

1. Connect the GT-64120 INTERRUPT* pin to the appropriate interrupt input to the CPU on J32.
2. Remove the pull-up resistor on the same CPU interrupt input. This pull-up would otherwise cause a conflict if J7 is pulling the INTERRUPT* pin low for little endian mode. For example, the resistor for Int0 is R255, and the resistor for Int3 is R252. These resistors are located on the back of the EV-64120 board adjacent to the J32 header.

After the GT-64120 comes out of reset, the GT-64120 drives the INTERRUPT* pin high (non-active) until there is a pending interrupt. The state of the INTERRUPT* signal is defined by the Interrupt Cause Register 0xC18 of the GT-64120.

11.5 Programming Reference - Software Tool

Question:
In looking at the EPLD designs on the EVB64120 board. Please explain the purpose of RST_control. RST_control appears to be generated by the second EPLD (U4) as a function of: [BootCS # !(ladd20)# !(ladd19)], what is the purpose of this?

Answer (last updated December 18, 2000):
The RST_control is a mechanism that allows the CPU reset to be deasserted via PCI control. U4 is responsible for the CPU reset. When in this mode (S2 asserted), only a specific access from the PCI to a certain address shall cause the CPU reset to be deasserted, so that until then, the PCI can configure the 64120 while the CPU is being reset. Deasserting the CPU reset via PCI control after the 64120 is configured, ensures that the CPU accesses the 64120 when it is ready for such access.

11.6 Customer Configurable Options - Connecting to 5V only PCI

Question:
Is it possible to run the EV-64120 Evaluation Platform on Galileo's 4PB passive backplane?

Answer (last updated December 18, 2000):
No. The 4PB passive backplane does not include the required 3.3V support. You should use the 64PB backplane.

12. Pin Information:

12.1 Pin Assignment Table

Question:
Aside from an unreckognized PCI command, the 64120's DEVSEL timing is fixed at the second clock cycle after Frame is asserted, right? Is there any condition where this timing would be any different?

Answer (last updated December 18, 2000):
The GT-64120 will always assert DevSel* two cycles following Frame.

Question:
I have a couple of questions concerning output GT-64120 driver types:

1.Perr_0*, Perr_1* Are these outputs Sustained Tri-State?

2.Serr_0*, Serr_1* Are these outputs Open Drain?

3.Int_0* Is this output open drain?

Answer (last updated December 18, 2000):
Yes on all three questions.

Question:
Does GT-64120 VREF affect the output or the input of the PCI or both? Regarding VREF0/VREF1 input pin of the GT64120, when we compare two cases(level is 3.3V or 5V), which is correct?

1. Only input threshold level is changed. Output level is not changed and it is still 3.3V level even when VREF is 5V.

2. Both input threshold level and output level are changed to match 3.3V/5V environment respectively.

If 1. is correct, is it guaranteed that receiver(5V Vcc) on PCI can work correctly when the GT64120 outputs 3.3V level signals?

Answer (last updated December 18, 2000):
#1 is the right answer. Still a 5V VCC receiver will work properly (the input threshold voltage of a 5V signaling is ~2V so we are ok).

13. Address Space Decoding:

13.1 Address Remapping

Question:
Does address remapping allow addresses to be mapped from one device to another? For example:can I remap the reset vector to SDRAM?

Regarding booting from SDRAM - I only have 1 SDRAM bank on SCS0 and I want to boot from that. In the meantime the CPU is asserting 1fc00000 etc. I can still boot from PCI memory.

Answer (last updated August 22, 2000):
The remapping is used to have the same physical address within the SDRAM or a Device that is in a different location on the PCI or CPU.

Our device does the address decoding in two stages.

1. The first stage is either on the PCI or the CPU side. Once this address is mapped to a group (i.e. SCS[1:0], SCS[3:2], CS[2:0] etc.) the external access will go to one of the members of the group.
2. The second stage selects one of the members of the group.

The remapping is done in between the first stage and the second stage. Thus if the CPU or PCI has an address that matched one group the remapping will not cause the access to go to a different group.

It can only change which address range it will go to within that group.

The CPU can boot from the SDRAM by simply holding it in reset while the PCI re-configures our device and maps the first stage registers and the second stage registers to map the CPU boot address to an SDRAM device.

The CPU can boot from anywhere you want assuming you follow the next steps in the order they are written (this example is for booting from CS[0]*) -

1. CPU is held in reset while GT64120A is released from reset.
2. From PCI access register at offset 0x038 and 0x040 and disable the group CS[3]* & BootCS* (look at section 3.2)
3. From PCI access register at offset 0x008 and 0x010 and write the values 0x00f8 and 0x7f respectively.
4. From PCI access register at offset 0x440 and 0x444 and disable the BootCS* (look at section 3.2)
5. From PCI access register at offset 0x420 and 0x424 and write the values 0xfc and 0xff respectively. The value 0xfc can be lower according to the size of your SDRAM.
6. Write from PCI to the SDRAM the boot code.
7. Release the CPU.

Question:
Regarding the GT-64120:

PCI_0 Swapped RAS[3:2] Base Address Remap: initial value seems to be 0 during simulation.

PCI_1 Swapped RAS[3:2] Base Address Remap: initial value seems to be 0 during simulation.

PCI_0 Swapped CS[3] & BootCS Base Address Remap: initial value seems to be 0 during simulation.

PCI_1 Swapped CS[3] & BootCS Base Address Remap: initial value seems to be 0 during simulation.

Answer (last updated December 18, 2000):

The reason you probably get 0 is that swap BARs are disabled through BAR enable registers (0xc3c,0xcbc). If a BAR is disabled (it's related bit in the BAR enable register is set to 1), the BAR and the BAR remap register are reserved - read only 0.

3. Address Space Decoding

3.6 Address Remapping

Question:
Regarding the GT-64120:

PCI_0 Swapped CS[2:0] Base Address Remap: missing register description 0xc60

PCI_1 Swapped CS[2:0] Base Address Remap: missing register description 0xce0

Answer (last updated December 18, 2000):
The GT-64120 doesn't have Swapped CS[2:0] BAR nor remap register. It has swap BARs and remap registers only for RAS[1:0],RAS[3:2],CS[3]&BootCS.

13.2 CPU PCI Decode Override (1 Gbyte and 2 Gbyte PCI Address Spaces)

Question:
Can a single GT-64120 have both its PCI ports connected together, thus appearing as 2 devices on the PCI bus? Electrically, I believe it's OK, but there may be internal or CPU port issues that I'm unaware. The reason to do this is to double the CPU's windows into the PCI bus (indeed, we will turn off the base address registers of the second PCI port to allow only outgoing [non target] traffic through this port).

Answer (last updated December 18, 2000):
CPU PCI Override provides a 1 Gigabyte window to PCI (see address space decoding chapter of the data sheet). However, the method you suggest is an alternative. There might be a problem if GT-64120 PCI0 master will try to access PCI1 slave through the PCI bus (or PCI1 master access to PCI0 slave). If the software programing 64120 registers can make sure this never happens.

One small issue - LOCK* is not supported on PCI1, so it can not work in a system that uses LOCK*.

There are 2 windows into PCI bus for each PCI bus in the 64120, so if you short the two PCI buses together you can reach 4 windows.

the relevant registers are:

058 - pci0_mem0_base

060 - pci0_mem0_top

080 - pci0_mem1_base

088 - pci0_mem1_top

0a0 - pci1_mem0_base

0a8 - pci1_mem0_top

0b0 - pci1_mem1_base

0b8 - pci1_mem1_top

in addition we have the PCI IO window for each PCI bus:

048 - pci0_io_base

050 - pci0_io_top

090 - pci1_io_base

098 - pci1_io_top

by the way, we have similar windows also in the 64011 chip (2 PCI mem

windows + 1 PCI io window).

14. CPU Interface Description:

14.1 SysAD, SysADC, and SysCmd Buses

Question:
What are all the events for which the GT-64120A will issue a bus error on the CPU bus:

Answer (last updated January 12, 2001):

All the events for which the GT-64120A will issue a bus error on the CPU bus:

1. Block read from internal registers.
2. Read Address miss WITHIN the CPU interface unit's decoding.
3. Read Address miss WITHIN the Memory interface unit's decoding.
4. Data Parity error during memory read (when parity is enabled in the Memory interface unit).

14.2 MIPS Write Modes and Write Patterns Supported

Question:
What does "External Hit delay" in the CPU Interface configuration register mean? For what? When we use the EV64120-CacheModule (GT-64120), it should be 0x0(Non-registered), right? And is this bit meaningful only for a R5000 native L2 cache, or for both the R5000 L2 and the R4700+GT64012 ?

Answer (last updated December 18, 2000):
This bit was implemented in order to be 64012 compatible - 64012 supports the two options of hit delay (for different versions of R4600), so we did the same in 64120. if the bit is 0, 64120 sample hit the next cycle after validout. if it is 1 it samples hit 2 cycles after validout. as far as I know, this is how R5000 works, so when working with R5000 need to set the bit. i don't know about R4700 and 64012.

14.3 Multiple GT-64120 Support

Question:
What is the initialization sequence when booting the GT-64120 in Multi-GT mode?

When booting the GT-64120 (Multi-GT mode is enabled by reset configuration) the CPU hangs while attempting to access the Low/High decode address registers in the CPU interface block. The CPU writes new values to a range of address map registers starting from offset:0x004.

Just after writing to the SCS[1]* Low decode address register (offset 0x408), the GT-64120 does not respond. When trying the same process in normal mode (Non-Multi-GT), no problem occurs.

Answer (last updated February 2, 2000):
In the GT-64120 data sheet (revision 1.2) there is a footnote in section 4.7.2 “Multi-GT Mode Enabled” that states: “As long as Multi-GT mode bit is SET, there is no access to PCI, SDRAM and Devices, and DMA internal registers. There is access only to CPU interface internal registers and to boot ROM.”

Except for the “Internal Space Decode” register 0x068, all of the address decode registers should only be programmed after coming out of Multi-GT mode.

Changing the “Internal Space Decode” register 0x068 will allow the address decode registers to reside at different address locations for each of the GT-64120 devices once out of Multi-GT mode.

15. Memory Controller:

15.1 SDRAM Controller

Question:
Does Refresh have priority over other requests to the GT-64120 memory bus?

Answer (last updated February 2, 2000):
The GT-64120 internal arbitration to memory is round robin (CPU - PCI - UMA - DMA - Refresh - UMA). Please refer to the first page of the “Memory Controller chapter in the GT-64120 data sheet.

For example, to optimize performance, a PCI master may perform continuous burst accesses to the GT-64120 SDRAM. The GT-64120 PCI slave unit splits the long PCI burst to 32 or 64 bytes bursts (depending on the MaxBurst register setting). The GT-64120 memory unit round robin arbitration scheme relates to these “short” bursts.

The bus occupancy time from CPU, PCI, UMA, and DMA devices is not restricted to the refresh interval. If necessary the GT-64120 may de-assert TRdy* (or a STOP could be issued if a time-out is reached) to accomplish the refresh. In this scenario, refresh has “higher priority” than PCI because the refresh could occur before the end of the continuous burst to SDRAM.

With a greater GT-64120 memory clock (TClk) and smaller PCI clock (PCLK), Trdy* may not need to be de-asserted to accomplish the SDRAM refresh.

Question:
The GT-64120A can handle processor bus and SDRAM bus up to 100MHz. In the GT and SDRAM-DIMM interface, do we have to register the ADR/CNTL signals going to GT to SDRAM, if we have enough margin?

Answer (last updated August 22, 2000):
The SDRAM interface does not need to be registered if there is margin in the timing to allow for loading and signal fly time.

Question:
Is the GT-64120A SDRAM interface compatible with 32-bit, 128Mbit SDRAM? The documentation for the SDRAM Address Decode Register has a table for a 64-bit, 128Mbit configuration, and for a 32-bit, 64Mbit configuration, but not one for the 32-bit, 128Mbit configuration. Even though it's not listed in these tables, we were hoping that we could find a way to make it work. Is this possible, and if so, what needs to be done to make it work?

Answer (last updated January 3, 2001):

The reason it is not in the GT-64120A datasheet is because the 32-bit 128Mb configuration is not supported. The 64-bit 64Mb configuration is prefered over the 32-bit 128Mb configuration. The price is the same but the performance is better (with the 64-bit device).

15.2 Programmable SDRAM Parameters

Question:
It appears that GT-64120A BA0 & BA1 don't hang around long enough for the PRECHARGE command to take effect on the bank that was previously accessed. When we move the bank selects down into a region below 8 Mbyte, our stack and OS get corrupted. What are the required register settings for an SDRAM device?

Answer (last updated January 3, 2001):
The 256 Mbit density SDRAM's are usually 4 way bank interleaving, however you need to set this bit separately to indicate that there are 4 banks and thus 2 BA signals. Please check the value of bit 5 "bit 5 64bitInt" in the SDRAM Parameter Register of this SDRAM chip select (one of the registers at offset 0x44c - 0x454) to configure the 64-bit SDRAM Interleaving. Bit 5 specifies how many banks are supported for interleaving if the bank. The defualt is 0x0 for 2 way

bank interleaving. Set this to 0x1 for 4 way bank interleaving.

15.3 SDRAM Performance

Question:
Would enabling ECC have a performance impact? If yes, by how much?

Answer (last modified August 22, 2000):
ECC adds one clock of latency. For example, the CPU to SDRAM read performance for the GT-64120A with bypass not enabled is 10-1-1-1 instead of 9-1-1-1.

Question:
In our GT-64120 design one PCI bus writes to the SDRAM while the other PCI Bus reads from it. Now if we use the speculative method then when one PCI is reading from the SDRAM (and now one fifo is filled and the other fifo might get filled) and at this time if the second PCI Bus wants to write to the SDRAM how will the 64120 act as a target of this write? Meaning that will it insert waits or will it do a retry?

Would it be better to use the non-speculative method so that the throughput on one PCI Bus is not affected by that of the other? or is it going to be equally the same in the above situation (By the way we can guarantee that the addresses will not be the same for reads from one PCI Bus and writes by the other PCI Bus)

Answer (last updated December 18, 2000):
The GT-64120 has two seperate PCI slave units - one for each PCI bus. the two units share the same one interface to SDRAM unit. there is an arbiter between them that give them fair arbitration. One more thing to remember - the SDRAM unit is capable of handle two requests in parallel - it can accept a new request even if it is in the middle of a previous transaction, so it minimizes the penalty between last data of previous transaction and 1st data of new transaction.

PCI write transactions in the GT-64120 are posted - we first fill the FIFO and only when it's full, we request from the SDRAM unit to write the data to SDRAM (while doing this, we can still fill the 2nd FIFO with data written from PCI bus). We can't accept any more data only when both slave's FIFOs are full, so we deassert TRDY# till FIFO is empty again (it's data was moved to SDRAM), and capable of receiving new data. we will assert STOP# only if the slave reaches timeout (timeout value is programable so it's up to you if you want the transaction to be STOPed or not).

Each FIFO fill/empty corresponds to one SDRAM transaction. so if you use the speculative read method, each FIFO fill from SDRAM corresponds to one SDRAM read transaction. so if you have in one PCI long burst read, and in the other PCI long burst write, in both slaves there will be toggle between the FIFOs (two FIFOs in each slave), but since each FIFO fill/empty is translated to one SDRAM request, and since there is an arbiter that gives fair arbitration to the two slaves, you will get time sharing. so in your example, you will gain from the speculative read as long as it stands alone, if it is interfered by write from the other PCI, you will not gain, but it will not do any harm to the write in the other PCI.

Question:
If the access is non-aligned to an SDRAM burst and worst case if the burst is set to 8 and the first access is to an address which is the last one in this burst length, in that case I think the GT-64120 SDRAM Controller shall start the fetch from this address and then initiate a new access to the SDRAM again to fetch the remaining part of the burst (This is because the SDRAM bursts do not allow you to cross boundaries and they wrap around, so that say the access started at address 7 the burst goes 7,0,1,2 ...).

Could you elaborate on how the GT-64120 shall look at this situation? It probably will add wait states after the first 64 bit chunk is transfered on to the PCI bus. Do you have any ballpark figure as to how many waits it would add ?

Answer (last updated December 18, 2000):
Each slave FIFO fill/empty corresponds to SDRAM transaction. it is the GT-64120 slave responsability to do the split. if for example there is a PCI burst write to 64120 SDRAM with start address 0x3c, 1st data will be written to 1st FIFO, and the rest data will be written to 2nd FIFO (while 1st FIFO request a write of a single data to SDRAM). The same

goes for reads - PCI burst read from 64120 SDRAM with start address 0x3c will result in read of single data from SDRAM (data is written to 1st FIFO), and then burst read of 8 datums from SDRAM (data written to 2nd FIFO).

Question:
The GT-64120 data sheet says that PCI to SDRAM Read returns the first data after 13 clocks.Probably the read data that comes to the PCI first goes into the Fifo inside the 64120.

Answer (last updated December 18, 2000):
It's true that data from SDRAM first goes into the FIFO. However, we don't wait for the FIFO to get full. As soon as there is one valid data in the FIFO, we drive it on the PCI bus. Part of the 13 clocks goes for synchronization overhead (we get the data from SDRAM in sysclk domain, and drive it to PCI in pclk domain).

Question:
The GT-64120 read access time from pci to sdram is about 13-1-1-1 ... I have the following questions regarding the timing diagram:

1. The diagram shows the timing for a 32 X 4 bytes read, during which the sdram burst length is set to be 8. Apparently the GT-64120 wouldn't write to the receive fifo for the pci device until the first burst of 8 X 8 bytes is completed. Why? I read the spec for GT64010A and found that the same thing occurred in "PCI 32 Bytes Read Multiple from DRAM": no data is ready until the 8 x 4 bytes burst is completed. Is there a reason for that?

2. Assuming what mentioned in 1 is necessary, will the read access time improve to, say 10-1-1-1 ... if the sdram burst length is set to 4 or even 2? In that case the GT-64120 will start writing the receive FIFO for the pci device 2 or 4 sdram cycles after SCAS is asserted.

3. I also don't see why we need to wait that long for pre-charge between the 2 8 x 8 byte bursts. Precharge command should be asserted at 1 cycle before the last data out for a CAS Latency of 2. Together with multi-bank inter-leaving, a no-penalty situation should be achievable such that there will be no gap between data at all on the AD bus.

We are concerned about read access time from pci to sdram because we have pci bw bottleneck in our current architecture mainly caused by pci to dram accesses. We expect 10-1-1-1 or better because 10-1-1-1 is what we currently get from 64010A.

Answer (last updated December 18, 2000):
I compared pci read from dram performance between 64010A and GT-64120. I run it with the SAME conditions (same clock ratios, same sync mode, best dram parameters). In GT-64120 the 1st data is driven on pci bus ONE clock LATER than in 64010A. The reason is that Memory Interface Unit get's it's 1st data from SDRAM one clock later than it got in 64010A from DRAM. But 2nd,3rd,4th ... data comes FASTER than in the 64010A. one important thing - when comparing, don't count CLOCK CYCLES bus nS - we loose one clock but it is a 66MHz clock!

Question:
If the GT-64120’s TCLK and PCLK are asynchronous, a synchronization stage is always needed. This synchronization stage could be omitted if TCLK and PCLK are tightly coupled (this depends on the implementation your designers used).

See "PCI Read Performance from SDRAM" chapter in the GT-64120 data sheet. SYNC Mode 2 delivers most performance when using TCLK/PCLK ratio 66/33 (PCLK frequency is synchronized to TCLK). So what is the relationship between these two clocks?

Answer (last updated December 18, 2000):
T clock is the processor clock which controlls the CPU bus interface, the internal timing and the AD bus timing (a master clock if you like). The P clocks are the PCI clocks and are entirely independent from each other. We have internal FIFO's and arbitration mechanisms which decouple the internal functional units.

You can run all the clocks at the same frequency if you like.

15.4 Unified Memory Architecture (UMA) Support

Question:
Could a high priority UMA request cause the GT-64120 to issue a PCI stop during a long PCI burst access to SDRAM (GT-64120 as target)?

Answer (last updated February 2, 2000):
Yes, a high priority UMA request may cause the GT-64120 to issue a PCI stop during a long PCI burst access to SDRAM (the GT-64120 is a target device on PCI).

If a UMA device places a high priority request for an access to SDRAM, the GT-64120 has a maximum of 35 TClks before it will assert MGNT* (section 5.7.5 of the GT-64120 data sheet rev 1.3).

The GT-64120 PCI slave unit splits the long PCI burst into 32 or 64 bytes bursts (depending on the MaxBurst register setting). The GT-64120 memory unit round robin arbitration scheme (as shown on the first page of the "Memory Controller chapter in the GT-64120 data sheet) relates to these "short" bursts.

If there is a pending High priority UMA request, UMA will be served after the GT-64120 PCI slave unit accesses the SDRAM with a 32/64 bytes burst. The GT-64120 PCI slave unit will insert wait states on the PCI bus (de-assert TRDY*), and if a time-out is reached, will disconnect (assert STOP*).

Question:
I'm writing some new [PLD] code which uses the UMA-support in the GT64120A. I am going to use the treq signal to give the bus back to the GT64120, and I would just like to verify the signal really is asserted when the SDRAM needs refresh.
Answer (last updated August 22, 2000):
The TREQ* is asserted for any internal request of the bus ownership, including a refresh request.

Question:
The GT-64120A arbitration round robin for the memory controller shown on the first page of the Memory Controller section doesn't explain how long each controller gets to keep the bus. Here's what I've determined/guessed, and would like your input:

(V)UMA - no limit; up to our UMA controller.

PCI - longest single burst - 8 data transfers, plus PCI overhead

Refresh - as needed, with the rate controlled by RefIntCnt

DMA - 64B transfer (8 clocks with 64b busses)

CPU - ?

Answer (last updated January 3, 2001)
The GT-64120A VUMA has no limit as you say. Up to your design of the arbiter. Other then that each resource receives one access when requesting. Both PCI and DMA can requst up to 8 data transfer . The CPU maximum transfer is a cache line which is 4 data transfers and the refresh will generate a requst at a rate according to the refresh counter. Each of the accesses generated by either the CPU, DMA and PCI have an overhead of about 8 cycles - RAS to CAS, CAS to Data and internal protocol overhead.

This overhead is partially hidden by hidden arbitration and also if their is interleaving.

Question:
The GT-64120A data sheet mentions operation with two GT-64120s, at a reduced CPU bus rate. This is interesting for us, because we're concerned about overall memory performance using the shared UMA memory. How about a configuration with one 64120 sharing memory with our custom device, and a second 64120 with private, non-UMA memory? Seems like that would get us two disjoint memory spaces with two simultaneous open memory pages. What is the down side - aside from money? Is DMA supported between the two 64120s?

Answer (last updated January 3, 2001)
This can definitely boost your overall performance. The down side could be in reduced system speed if your AC timing analyses of the CPU bus shows that the extra laod will prevent working at 100 MHz.

If the two GT-j64120A devices reside on the same PCI bus then you can use the DMA to transfer information from one device to the other device. We do not keep any page open when accessing the SDRAM. The only cycle saving is when there is interleaving we can open a new page before we closed the first one when they are in different banks.

Question:
If the GT-64120A VUMA gets to keep the SDRAM access with no limit and that it is up to our design of the arbiter... Does this mean that we will never be pre-empted off of the SDRAM? Or if we can, what is the likelihood and how is this done? The manual says that access is granted via a 1 clock cycle low assertion of the MGNT pin. How would pre-emption, if it is done at all, be signalled.

Answer (last updated January 3, 2001)
I am not sure from the perspective of which side you are looking to be pre-empted. Assuming that the GT64120A is the M1 and your arbiter with the logic is M2 the following will occur -

1. After reset M1 is the master of the bus.

2. According to the arbitration scheme described in section 5 figure 10 the SDRAM ownership will be given to M2. A UMA high priority has two slots in the round robin arbitration while a low priority request is granted access when M1 is IDLE.

3. The bus will remain at the full ownership of M2 until M2 will relinquish the ownership by de-asserting MREQ*. M1 can signal to M2 system of a pending request using the TREQ* signal.

As long as MREQ* is asserted M1 will not take ownership of the bus. When MREQ* is de-asserted M2 should stop driving the bus on the same cycle and then their will be one dead cycle for turn over of the bus.

Question:
We are thinking of always asserting a high-priority VUMA request to the GT-64120A so we can bound the arbitration overhead for VUMA. As I understand this, this is done by sending a 0100000... to the MREQ pin. When is the earliest that MGNT will be low asserted? I have a state machine that needs to make sure to check for MGNT and I do not want to miss it. The absolute earliest possible if MREQ is sampled on the rising edge is when MREQ is temporarily set to 1 before it is set back to 0. But I suspect, because of the bus handoff protocol (driving certain SDRAM lines high 1 cycle before MGNT is asserted, etc.) That the earliest MGNT can be asserted is the second MREQ 0.

Answer (last updated January 3, 2001)
Assuming that the GT64120A is the M1 and your arbiter with the logic is M2 -

The earliest the MGNT* signal can be asserted is 3 cycles. First rising edge is to capture the MREQ*. Assuming M1 is IDLE, then second rising edge control signals will be driven to the inactive state. Third rising edge the signal MGNT* will be asserted low.

15.5 Device Controller

Question:
Does data remain on the GT-64120 device AD bus between two consecutive bursts?

Answer (last updated February 2, 2000)
Between two consecutive burst accesses the GT-64120 keeps driving data on the AD bus according to the TurnOff device parameter. On writes to device, the GT will keep driving valid data on the AD bus N cycles after WR* de-assertion, where N is the TurnOff parameter. The next cycle on the AD bus (device or DRAM transaction) will be no earlier than this TurnOff time.

The TurnOff parameter is defined for reads in section 5.7.1 of the GT-64120 datasheet rev 1.4.

Between two consecutive accesses within a single burst write to a device, the burst address changes right after Wr* (write enable) deassertion. The AD bus will not change until the rising clock edge just before the falling edge which drives Wr*.

15.6 Memory Controller Restrictions

Question:
Regarding the GT-64120, while a read/write cycle to a 32 bit device of an odd number of words from an aligned address, or read/write cycle to a 32 bit device to an unaligned address, an extra read/write will occur to the complimentary word of the double word address accessed with no byte enables active.

This may result in destructive reads and loss of performance while accessing the GT-64120 device.

Where the terminology used: "word" - 32 bit, "aligned" - aligned to a word.

Examples:

1. CPU writes to offset 0 of a 32 bit device 1 word. Result: write to offset 0 with BE asserted and a write to offset 4 with BE not asserted.

2. CPU writes to offset 4 of a 32 bit device 1 word. Result: write to offset 0 with BE not asserted and a write to offset 4 with BE asserted.

3. DMA writes 5 words to offset 0 of a 32 bit device. Result: Burst of 6 to the device, with the last BE not asserted.

Is there a workaround for this ?

Answer (last updated December 18, 2000):
Yes.

1. If the device is always accessed for single word transfers, connect the GT-64120 AD[4:2], latched, to the device's least significant address pins instead of the BAdr[2:0]. This will make sure there are no destructive reads.

2. Access the device always with an even number of words at double word aligned addresses (e.g. 0x0, 0x8, 0x10...). This means that the DMA must have the DatTransLimit field set to at least 8 bytes. This will make sure there are no destructive reads and no loss of performance.

Question:I am unclear about the first workaround for the following GT-64120A 32 bit device limitation: “During a read/write cycle of an odd number of words to a 32-bit device from an aligned address, or a read/write cycle to a 32-bit device to an unaligned address, an extra read/write occurs to the complementary word of the double-word address accessed with byte enables not active. This may result in destructive reads and loss of performance while accessing the device.”

Workaround (1 of 2): If the device is always accessed for single-word transfers, connect the AD[4:2], latched, to the device?s least signifi-cant address pins instead of the BAdr[2:0]. This will make sure there are no destructive reads.

Does this mean that when the CPU writes a single word to offset 4, the AD[4:2] are 4 and when the CPU writes a single word to offset 0, the AD[4:2] are 0. ?

Answer (last updated December 19, 2000):
Yes. This workaround fully tested with both CPU writes a single word to offset 0 and to offset 4?

16. Data Integrity:

16.1 SDRAM ECC

Question:

The GT64120A data sheet explains a case where the syndrome contains a single "1". It states that the GT64120A will not report nor correct this case. Is this true?

Answer (last updated January 12, 2001):
Yes. This is a single bit error in the ECC byte, the data is correct. The initiator (e.g. CPU) will receive correct data, and the mistaken ECC byte will be kept in the memory as is (we do not drive this ECC to the initiator; we drive correct parity based on the data).

17. PCI

17.1 PCI Master Configuration

Question:
The data sheet says:

The CPU interface unit cannot distinguish between an access to the GT-64120A PCI configuration space and an access to an external PCI device configuration space. Both are accessed using an access to the GT-64120A internal space (i.e. Configuration Data register). The software engineer must keep this in mind, especially when byte swapping is enabled on the PCI interface. In this case, internal configuration registers and configuration registers in external devices will appear to have a different endianess.

But right above it talks about the address stepping thing and that appears to be a way for us to address up to 21 PCI target devices on PCI 0. Can you help clarify?

Answer (last updated August 22, 2000):
The upper section shows how to address up to 21 target devices for configuration access from an address perspective. The section you quoted refers to the data phase. Typically the CPU system is in big endian notation. The internal registers are in little endian notation. The PCI is also typically in little endian notation.

When the CPU is the master of a transaction and the access is to internal registers the GT64120A will perform byte swapping on the data (if programmed to do so by bit 12 of register 0x000). An access to register 0xcfc is an access to internal register and thus in this case the data will be swapped when written into the register. When the configuration access is targeted to an external device on the PCI (according to the value in register oxcf8) the content of register 0xcfc will be driven in the data phase onto the PCI. If bit 0 of register 0xc00 is configured to perform byte swap when the GT64120A is the master of the PCI then the content of the 0xcfc will be byte swapped and thus we will receive in the data phase a big endian notation data will be driven on the little endian PCI bus.

CPU data [11-22-33-44] --> GT64120A internal reg [44-33-22-11] --> PCI bus [11-22-33-44]

To avoid this the CPU should do in software a byte swap when accessing the configuration space of devices on the PCI which are not the local system controller.

So in order to write 44-33-22-11 to the external device the CPU should :

CPU data [44-33-22-11] --> GT64120A internal reg [11-22-33-44] --> PCI bus [44-33-22-11]

Question:
Regarding parking the PCI bus on the GT-64120, is it meaningfull to park the bus on the 64120's PCI Interface? In other words if the bus is idle and the arbiter parks on the 64120 then would the 64120 start its next PCI cycle without asserting request?

Answer (last updated December 18, 2000):
The 64120 PCI master will always assert REQ. If it is parked on the bus it implies that the next cycle after REQ it asserts FRAME#. If it is not parked on the bus, it will assert REQ and wait for GNT. If theoretically, the arbiter can return GNT the same clock cycle it got the REQ, you don't gain by parking on 64120. However, i don't think the arbiter can do such a thing, so you still gain at least one clock cycle.

Question:
The GT-64120 spec says there is one 8x64-bit CPU->PCI FIFO in 64-bit mode and two 8x64-bit CPU->PCI FIFOs in dual 32-bit mode. This doesn't seem right. It seems like it should be two 8x32-bit FIFOs in 32-bit mode. Which one is right?

Answer (last updated December 18, 2000):
In the GT-64120 we have two PCI masters and each of them has 8X64-bit FIFO in double PCI we have 2 8X64-bit FIFO. In 64-bit mode only one of them is activated as a 64-bit PCI master so we have only one 8X64-bit FIFO.

Question:
When is the GT-64120 Target-Abort indicated ?

Answer (last updated December 18, 2000):
Target-Abort is indicated by the PCI master when it senses that the Target aborted the PCI cycle not in te normal manner. Unlike Retry or Disconnect, where the Target may stop the cycle by asserting STOP* together with DEVSEL asserted, Target-Abort condition is when the STOP* is asserted and DEVSEL is deasserted. In the case of Retry/Disconnect, the PCI Master will retry the transaction. In the case of Target-Abort, the PCI master will not retry the transaction.

17.2 PCI Target Interface

Question:
How excactly is this GT-64120 burst managed, in the sense that since the SDRAM is programmed for a burst of fixed value (say 8 to match the Fifo Depth) and also that PCI cannot give any advance information of the number of bursts at what point in the whole cycle (wrt PCI) will the SDRAM controller initiate a new burst to fetch the next burst incase the PCI Burst length is longer than 2 times the burst SDRAM Burst length (in our case the 64120 shall have 2 distinct PCI buses).Will the access latency between 2 bursts on the SDRAM memory bus get hidden ? ALso what happens when both the PCI Buses are accessing the SDRAM data. Does this access latency get affected?

Answer (last updated December 18, 2000):
Since the GT-64120 does not know in advance the read burst size, it always requests from SDRAM the maximum to fill one FIFO. If it requested more than what is really needed, it will flush the extra data from fifo at end of transaction.

We support two methods of filling the FIFOs. One is to first fill one FIFO, empty it to PCI, and only when FIFO is empty (all it's content was driven to PCI) and transaction is still alive, request for new burst from SDRAM and when data comes from SDRAM fill the 2nd FIFO.

In the second method, we don't wait for the first FIFO to being ALL read to PCI. After 1st data is read from 1st FIFO, we speculatively request new burst from SDRAM in order to fill 2nd FIFO. this way we have less dead cycles (TRDY# deasserted) between read from 1st FIFO to PCI and 2nd FIFO. Once again, at end of transaction, we flush extra data from both FIFOs.

Use max_burst register (bits[6:4]) for using this speculative read method. The access between the two bursts as you mentioned above can not be totaly hidden, but can be small if you use this speculative read method.

Access from both PCI buses to SDRAM might affect access latency. Since SDRAM interface is 64 bit and PCI is 32 bit, PCI bandwidth is at least two times slower than SDRAM bandwidth. In an ideal situation if you use speculative reads, while SDRAM fills one FIFO in PCI_0 slave, FIFO in PCI_1 slave should be read by PCI. However, in reality there are overheads.

Question:
If I am writing to PCI0, and both GT-64120 ping-pong FIFO's are full, is it possible that TRDY would be de-asserted ("1") for only one clock cycle? In other words, after only one clock cycle, one of the FIFO's becomes available. Is this possible ?

Clk	Trdy	Devsel	Frame
0	1	1	1
1	1	1	0
2	1	1	0
3	1	0	0
4	0	0	0

Or would it always be de-asserted for 2 clock cycles or more, even from the beginning. Like this...

Clk	Trdy	Devsel	Frame
0	1	1	1
1	1	1	0
2	1	1	0
3	1	0	0
4	1	0	0
5	0	0	0

Answer (last updated December 18, 2000):
If you are talking about a situation that previous PCI write transaction filled both GT-64120 Slave FIFOs, and now you start a NEW PCI write transaction targeted to 64120, TRDY# might be ASSERTED two cycles after FRAME# assertion if 1st FIFO was already flushed to DRAM.

If for example the previous PCI write transaction is a write of 9 words to address 0x3c and the new PCI write transaction starts right after (back to back), one word will fill 1st FIFO, the rest 8 words will fill the 2nd FIFO, and by the time the new transaction starts, 1st FIFO is already empty and you will get TRDY* asserted 2 clocks after FRAME* assertion:

Clk	Trdy	Devsel	Frame	Tx#
0	1	1	1	-
1	1	1	0	1
2	1	1	0	1
3	0	0	0	1
4	1	0	0	1
5	1	0	0	1
6	0	0	0	1
7	0	0	0	1
8	0	0	0	1
9	0	0	0	1
10	0	0	0	1
11	0	0	0	1
12	0	0	0	1
13	0	0	1	1
14	1	1	0	2
15	1	1	0	2

16	0	0	0	2
17	0	0	0	2

bottom line - when we say in the data sheet that there is 2 clocks "penalty" on toggle between the FIFOs, we mean on the same transaction. However, since we can never assert TRDY# of a NEW transaction before 2 clocks after FRAME# assertion, it implies that even if you run back to back write transactions, you will get at least 2 clocks TRDY#

deasserted between previous transaction TRDY# and new transaction TRDY#.

Question:
I suppose depending on the start address, the GT-64120 TRDY could be asserted for only one clock cycle while FIFO switch occurs. Like this.

Clk	Trdy	Devsel	Frame
0	1	1	1
1	1	1	0
2	1	1	0
3	1	0	0
4	1	0	0
5	0	0	0
6	1	0	0
7	1	0	0
8	0	0	0

Answer (last updated December 18, 2000):
Correct.

Question:
I'm concerned the PCI device on the GT-64120 PCI bus 1 may hog the bus because the SDRAM will continue to be granted to the device on PCI bus 1 if it assert frame. Is there a burst size limit control that will send a retry to the device on bus PCI 1 when a burst limit is reached so that devices on bus PCI 2 will have a chance to get on the bus when PCI is arbitrated?

Will the burst limit feature help in this situation? The only problem is if Frame* is still asserted on PCI_0, it will hold the bus indefinely, correct? (i.e. of course not considering the TImeout registers).

Answer (last updated December 18, 2000):
If you are worried about PCI_0 occupy SDRAM Interface unit because FRAME0# is asserted (long burst), there is no problem. Remember - PCI slave has the two FIFOs with the ping-pong mechanism. Every time we move from one FIFO to another, it is a new request from SDRAM interface unit and the arbitration cycle starts again. So if PCI_0 has a long burst (FRAME0# is "always" asserted), and in the middle PCI_1 has an access to SDRAM, the moment PCI_0 slave interface toggle from one FIFO to another, PCI_1 will get the access to SDRAM and PCI_0 will have to wait.

Question:
This question pertains to a PCI write to GT-64120 memory when both FIFO's are full, and TRDY is de-asserted from the beginning. What is the minimum number of clock cycles for which TRDY would be de-asserted after DEVSEL goes active?

Answer (last updated December 18, 2000):

In general, it will take a full Memory Interface Unit arbitration cycle in the worst case which means the max time it takes for Memory Interface Unit to serve CPU Interface Unit request (probably block read from the slowest device), PLUS the time it takes to Memory Interface Unit to serve DMA request (probably read of 8 words from the slowest device) PLUS Memory Interface Unit refresh cycle.

It also depends on Ready* pin (if they use it, it can take infinite time). It also depends if they use double PCI - if they do, there is an arbiter between PCI_0 and PCI_1, so if the last served request from PCI_0 and now you have requests from both PCI_0 and PCI_1, PCI_1 will be served first, and PCI_0 will have to wait for another whole Memory Interface Unit arbitration cycle.

Question:
I know that setting the GT-64120 max burst length can be done for 4 or 8. Is a 0 for 4 and 1 for 8? What does it do? Does it affect the prefetch on reads at all? Or only on writes?

Answer (last updated December 18, 2000):
The max burst length is a 0 for 4 and 1 for 8. It affects any PCI transaction to DRAM or Device. altough the 64120 inter-units bus is capable of running bursts of 8 * 64 bit, and Memory Interface Unit is capable of running the same bursts to 64 bit DRAM or Device, this register let the user limit this burst to 4 * 64, in order to let Memory Interface Unit serve other transactions.

The user MUST program this register to 0 if not working with 64 bit devices, in order to avoid a burst longer than 8 which is not supported by Memory Interface Unit.

For example, if working with 32 bit DRAM, and Max-burst is programed to 1, PCI slave will attempt to do burst of 8*64bit to Memory Interface Unit, which means Memory Interface Unit should run burst of 16*32 to DRAM but it CAN'T!!!

Question:
During a PCI write to 64120 if both FIFO's are full, can I expect TRDY to be inactive at the beginning of the transaction until either of the following occurs?

1. timeout0 expires, or

2. FIFO is emptied into SDRAM.

Is it possible to completely avoid TRDY from being deasserted by not crossing 64 byte boundaries. Is the above situation an exception to that statement?

Answer (last updated December 18, 2000):
If PCI writes to 64120 while both slave's FIFOs are full, TRDY# will be inactive untill timeout0 is expiered (in which 64120 slave will assert STOP#) or till one of the FIFOs is emptied to SDRAM - which ever comes first.

If TRDY# is asserted, it will not get deasserted till end of write transaction, if transaction does not cross 64 byte boundary.

Question:
If I issue two consecutive WRITES cycles of 12 words (32 bit = 1 word) as follows...

1st - 12 words starting at 00

2nd - 12 words starting at 00

Will I see GT-64120 TRDY deasserted during the second or not?

Answer (last updated December 18, 2000):
In general, on write transactions 64120 PCI slave will assert TRDY* two cycles after frame assertion, and will deassert TRDY* for 2 cycles every toggle between slave FIFOs. this is true only if slave's FIFOs are empty - they are being empty (data driven to DRAM) faster then being filled, so by the time the 2nd FIFO is full, 1st FIFO is already empty, and capable to be filled up again.

In your specific question, if PCI is 32 bit, 33MHz (which means DRAM interface runs much faster), and "max burst" register bits at offset 0xc40 are set to 1 (which means FIFO size is 16 * 32), TRDY* will not be deasserted for the whole transaction (each 12 words transaction will fill one FIFO).

Question:
What if the burst length does not match that of the target GT-64120 memory which in our case is 64 bits. We will have a DMA size of 48 bytes which is not an integer multiple of 8 bytes.

Answer (last updated December 18, 2000):
I am not sure what Burst length you refer to. any way, you can transfer any amount of bytes you want. The GT-64120 DMA burst length determines how to split the big data block to smaller bursts.

Question:
What is the earliest that 64120 would respond with TRDY inactive?

Can a PCI memory read / write command (CBE = 011x) be used for DMA operations of more than 8 words? (i.e. greater than a cache line). I will be doing anywhere from 1 to 32.

Answer (last updated December 18, 2000):
On PCI write to 64120, if FIFOs are empty you will get 1st TRDY* 2 cycles after FRAME* assertion, TRDY* will be active until FIFO is full, then 2 dead cycles, then it starts fill the other FIFO (if not full). On reads, 1st TRDY* depands on access time to DRAM/Device. You can read/write from PCI to DRAM/Device any burst length you want, but you will have dead cycles.

Question:
How are the GT-64120 "Ping pong" FIFOs for PCI DMA filled? If a burst does not completely fill the buffer, will the next burst start from the beginning of the next buffer, or will it finish filling the first buffer. Question relates to the frequency of TRDY inactive occurrances.

Answer (last updated December 18, 2000):
Next burst start from the begining of next buffer.

Question:
There is a two clock cycle delay when switching GT-64120 PCI Slave FIFOs, correct? I.e. TRdy is de-asserted for two cycles?

Answer (last updated December 18, 2000):
It depends. in the best case it is 2 clocks. however,in case of read transaction, TRDY# will be deasserted for many more clocks until data arrives from DRAM/Device. even in writes it might be more than 2 clocks after both FIFOs are full (if Memory Interface Unit is busy in some other activity, or if devices are slow, and by the time 2nd FIFO is full, 1st one is not empty yet).

Question:
What is the GT-64120 PCI performance on transactions, depending on the address. In other words, how does one FIFO fill up before ping-ponging to the other FIFO?

Now, on each FIFO interface, there are two 16 x 32 FIFOs in the 120.How does the 120 decide when to switch FIFOs? Is it different for reads or writes? What is the latency between switching from one FIFO to the other? I think a table in the data sheet of this would be helpful.

Answer (last updated December 18, 2000):

32bit PCI (16 x 32 FIFO) and max_burst bit is set:

STARTING ADDRESS   ENTRIES FILLED BEFORE SWITCHING FIFOS

00	16
04	15
08	14
0C	13
10	12
14	11
18	10
1C	9
20	8
24	7
28	6
2C	5
30	4
34	3
38	2
3C	1

32bit PCI (16 x 32 FIFO) and max_burst bit is cleared:

STARTING ADDRESS	ENTRIES FILLED BEFORE SWITCHING FIFOS
00	8
04	7
08	6
0C	5
10	4
14	3
18	2
1C	1

64bit PCI (8 x 64 FIFO) and max_burst bit is set:

STARTING ADDRESS	ENTRIES FILLED BEFORE SWITCHING FIFOS
00	8
08	7
10	6
18	5
20	4
28	3
30	2
38	1

64bit PCI (8 x 64 FIFO) and max_burst bit is cleared:

STARTING ADDRESS	ENTRIES FILLED BEFORE SWITCHING FIFOS
00	4
08	3
10	2
18	1

Question:
I am totally confused by the GT-64120 Max_Burst register description. The way I understand it, if the SDRAM is 64-bits then you must set Max_Burst to 16. If the SDRAM is 32-bits then Max_Burst = 8. If you do not set it this way (at least for 32-bit SDRAM), then you loose data.

This does just not seem like the right way to do things, shouldn't you DISCONNECT on the PCI bus. This is why DISCONNECT and RETRY exist. Maybe I miss understood.

Answer (last updated December 18, 2000):
If you use 64 bit SDRAM, you can program Max_Burst to 8 OR 16 (it gives you one more way to controll chip perfor-mence - if you limit PCI to DRAM bursts to 8, a pending CPU to DRAM transaction can be served faster). If you use 32 bit SDRAM, you MUST program Max_Burst to 8 (if you don't, you will probably lose data.

Now, for the technical reasons. the maximum burst Memory Interface Unit can drive to DRAM is 8 words regardless of DRAM width. so if you use a 32 bit SDRAM, the maximum burst per single DRAM transaction is 8 words of 32 bit. Since the internal busses in the GT-64120 are 64 bit wide, the maximum slave to Memory Interface Unit burst per single transaction in this case is 4 words of 64 bit.

It has nothing to do with PCI transaction burst size - it can be as long as you like. the way the slave has always worked, is that it splits the long burst that comes from PCI, to slave-Memory Interface Unit transactions with maximum burst equal to slave FIFO depth (each FIFO fill, is translated to single slave-Memory Interface Unit transaction). but since in the GT-64120 we doubled the FIFO depth, if you don't program Max-burst to 8, the slave can drive to Memory Interface Unit a maximum burst of 16 * 32 bit (actually 8 * 64 bit). programing Max-burst to 8 has the affect of minimizing FIFO depth back to 8 (as in 64010,64011).

There is no reason to disconnect the PCI transaction - the only thing need to be done is split the long PCI burst to appropriate slave-Memory Interface Unit bursts. This is exactly what we do - the only limit here is that the programmer need to program one more register.

Question:
Your timing diagrams for PCI accesses to GT-64120 SDRAM where for one interface only. We would like to know what the effects of simultaneous PCI requests to SDRAM would be. In our environment, we will have two 32-bit PCI buses. If both accesses are reads, will the SDRAM interface be able to pipeline the reads into SDRAM?

Answer (last updated December 18, 2000):
YES.

Question:
During simultaneous accesses, which GT-64120 PCI interface has priority? Does the CPU have priority over both PCIs into SDRAM?

Answer (last updated December 18, 2000):
If both PCI transactions come exactly at the same cycle, PCI0 gets the priority. the arbitration between PCI and CPU is round robin.

Question:

If simultaneous accesses occur from the GT-64120 PCI's are followed by a CPU access before the first PCI access is completed, will the CPU access occur before the other PCI bus is serviced?

Answer (last updated December 18, 2000):
YES, since our Memory Interface Unit does not distinguish between the two PCI buses.

Question:
If the GT-64120 PCI-1 interface is held off because the PCI-0 bus is accessing the SDRAM, can a CPU cycle to PCI-1 be completed?

Answer (last updated December 18, 2000):
If you are talking about a write transaction, the answer is YES. data from CPU will be posted to PCI1 master fifo (transaction is complete from CPU point of view), and PCI1 master will drive it on the bus as soon as the bus is free (as i understand your question, the bus is currently busy with the read).

Question:
We are experiencing prefetch from a 32 bit device (EXTERNAL FIFO)- even though the GT-64120 data sheet (rev 0.1) on page 19, section 3.2.9 Prefetching Data states clearly "Prefetching will not occur on read accesses from 32 or 64-bit devices." Could there be a bug in the 64120 here? What else could cause the 64120 to behave as described below?

Answer (last updated December 18, 2000):
The GT-64120 will not do prefetch from a 32 or 64 bit devices for CPU transaction. PCI reads from device or dram ALWAYS do prefertch. The only case PCI read will not result in prefetch is single data transactions, where the initiator master asserts FRAME* for ONE PClk only.

Question:
We are losing data from a FIFO device becuase of GT-64120 prefetch. When reading from a device to the PCI bus, the GT64120 is prefectching 8 words from the device before responding to the PCI request for data. More specifically, the master asserts FRAME and IRDY, the GT64120 then prefetches the 8 words from our device (the external FIFO), and then asserts TRDY and delivers the data to the PCI bus. The question is this: Why is the GT64120 prefetching data from a device, especially a 32 bit device, which the Rev 0.1 specification specifically says on page 19 cannot be done.

Answer (last updated December 18, 2000):
The GT-64120 (same as 64010 and 64011) is defined as prefetchable memory space. If the External master will force FRAME* to be active for a single clock cycle, the prefetch should not occur.

Question:
If prefetch is disabled for a certain type of access (i.e. an MRL) does this mean that for a 2 word PCI burst read, the GT-64120 will issue 2 seperate SDRAM RAS/CAS cycles?

Answer (last updated December 18, 2000):
Not always it depends on the address. The slaves FIFO is 8 words deep so the 64120 will read the data to fill the FIFO beginning from the address so if the address offset is 0x1c then their will be 2 separate cycles to SDRAM but if it is lower then 0x1c their will be only 1 cycle.

Question:
In our GT-64120 system, we will be using bridges with posted reads and writes. The bridges will disconnect from the secondary bus with retries while they forward the read request upstream. Multiple masters on secondary PCI bus can then also have the bridge attempt to give them data or issue writes while waiting for read data.

My understanding of the Galileo slave interface is that it will respond to a read request and then stay on the bus until it has responded to this request. i.e. We can't issue a read, have the Galelio disconnect, issue a write and then ask the Galileo for the read data thus doing having a PCI write hidden during the SDRAM access.

Answer (last updated December 18, 2000):
You are right. the Galileo slave interface does NOT support delayed reads - when it recieves a read request, it stays on the bus until it has the valid data from SDRAM to drive on the bus. If it reaches timeout before data from SDRAM is available, it asserts STOP#, and FORGET about the transaction - when the initiator master retry the read transaction, it starts all over again.

17.3 PCI Synchronization Barriers

Question:
What are the initial values of the GT-64120 0xc0 pci0 sync barrier virtual register and 0x0c8 pci1 sync barrier virtual register?

Answer (last updated December 18, 2000):
Sync barrier virtual registers are VIRTUAL which means these are not real registers - you can not read or write the register. if CPU read one of these registers, CPU Interface Unit will activate sync-barrier check (check that PCI slave FIFOs are empty), and when gets empty indication from PCI slave, drives ValidIn* with JUNK data. the CPU should not use this data - it is just an indication that sync-barrier check is complete.

Question:
100 clocks after the rising edge of reset, the GT-64120 simulation attempts to read PCI configuration register 0 from PCI 0. This results in the GT-64120 driving the bus with all 0's and asserting devsel0_. However, there is no trdy_ signal, which is what I would expect. Also, the data doesn't appear to be correct. Eventionally, the Galileo issues a stop0_, which I understand. This first read is shown in the attached PDF. A subsequent read of configuration register 4 results in an immediate stop0_ with no data. PCI 1 is disabled. PCLK0 and TCLK are running synchronously at 30 ns (~33 Mhz). The CPU is not connected. No initialization ROM is used. Any ideas...

Answer (last updated December 18, 2000):
Configuration read from PCI is one of our slave sync-barrier transactions. Slave will not finish the transaction properly until it gets "empty indication" from CPU Interface Unit. Is there a CPU in this environment? Are there any pending transaction in CPU Interface Unit's FIFO?

If our slave reaches timeout before it got the empty indication from CPU Interface Unit, it asserts stop#. A "normal PCI master" that gets this stop#, will retry the transaction. When there is empty indication, the configuration read will end properly.

In your case, the initiator master did not retry the transaction, but initiated a new configuration read (a new sync-barrier transaction), and since our slave can handle only a single sync-barrier transaction at a time, this transaction got stop# immediately.

17.4 Target Configuration and Plug and Play

Question:
The GT-64120A rev 1.0 data sheet says in Section 7.6.2 "PCI Autoload of Configuration Registers at RESET" that a PLD can be used to store the information loaded to configure the PCI registers. Then there is a note saying that"The PLD must be programmed to support burst read cycles". Does this mean that at boot time the access to the boot device (FLASH or EPROM or PLD)is a burst type read access? If so, then I have a problem with the eval board schematic. The FLASH used there (AMD29LV040)does not support BURST reads, and the addresses for this flash device come straight form the GT.How can it do a burst when on a device read access, the address is presented to the device only once at the beginning of the read cycle? What I am misinterpreting here?

Answer (last modified August 22, 2000):
The Autoload feature enables the user to perform reconfiguration of the internal registers after reset. This registers are 32 bit wide and thus the internal transaction is always 32 bit wide. If the external boot device is a 32 bit device then the GT-64120A will perform a single access each time to retrieve a single 32 bit data word per each register it is writing internally. If the external boot device is 16 or 8 bit device then it must support burst access since the GT64120A will perform a burst access to retrieve the full 32 bit of data it will later write to the internal register. On our schematics we

have boot device that is 8 bits wide and does not support burst access. Since the autoload feature is not implemented on this eval board - Dadr[0] is pulled high.

Question:
The data sheet says Section 7.6.2 "PCI Autoload of Configuration Registers at RESET" that a PLD can be used to store the information loaded to configure the PCI registers. Then there is a note saying that"The PLD must be programmed to support burst read cycles". Does this mean that at boot time the access to the boot device (FLASH or EPROM or PLD)is a burst type read access? If so, then I have a problem with the eval board schematic. The FLASH used there (AMD29LV040)does not support BURST reads, and the addresses for this flash device come straight form the GT.How can it do a burst when on a device read access, the address is presented to the device only once at the beginning of the read cycle? What I am misinterpreting here?

Answer (last modified August 22, 2000):
The Autoload feature enables the user to perform reconfiguration of the internal registers after reset. This registers are 32 bit wide and thus the internal transaction is always 32 bit wide. If the external boot device is a 32 bit device then the GT-64120A will perform a single access each time to retrieve a single 32 bit data word per each register it is writing internally. If the external boot device is 16 or 8 bit device then it must support burst access since the GT64120A will perform a burst access to retrieve the full 32 bit of data it will later write to the internal register. On our schematics we have boot device that is 8 bits wide and does not support burst access. Since the autoload feature is not implemented on this eval board - Dadr[0] is pulled high.

17.5 PCI Bus/Device Bus/CPU Clock Synchronization

Question:
In the 0.2 revision of the GT64120 data sheet there is no "Clock Section" which would specify the relationship between TCLK and PCLK especially for synchronous applications. Can you support us with additional informations concerning the relationship between these two clocks.

Answer (last updated December 18, 2000):
PCI spec let a system run with pclk frequency range of 0 up to 33MHz (or 66MHz). The 64120 supports the whole range of PCI frequency regardless of CPU clock (tclk). In order to achieve this, there a syncronization mechanism inside the chip (synchronize pclk domain signals to tclk domain and the other way around). however, this synchronization mechanism has it's clocks penalty.

In order to "save clocks" (get better performance), there are few sync modes supported by 64120, that affect synchronization mechanism work. If for example, you run with tclk at 50MHz and PCI at 33MHz, you can run in both sync modes 0 and 1, however, it's better to run in sync mode 1 (you will get higher performance). If you run with tclk at 66MHz, and pclk at 33MHz, and they are SYNCHRONIZERD (they are derived from the same source), you will get best performance with sync mode 2.

Bottom line - you can always run with sync mode 0 (this is the default). However, if tclk/pclk ratio in your system is such that you can use other sync mode, do it in order to get better performance.

17.6 Locked Cycles

Question:
We have two GT-64120s in our system, utilizing both 32 bit PCI buses on each controller. There are two PCI buses, both connected to each GT-64120, allowing devices on either PCI bus to access SDRAM connected to either controller. Assuming the two controllers are labeled A and B, is there any advantage of connecting PCI0-A to PCI1-B and PCI0-B to PCI1-A versus the opposite arrangement?

I know PCI1 on the GT-64120 does not have a LOCK pin, is there any mechanism for performing locked accesses from the CPU?

Answer (last updated December 18, 2000):

PCI_0 and PCI_1 sides of the 64120 are almost identical - you will get the same performance either way. The only difference is that PCI_1 does not support LOCK* and I2O.

About the lock question, CPU can not lock a cache line. Only PCI_0 can lock. However, if PCI_0 locks a cache line, this address is locked for accesses from CPU and from PCI_1.

Question:
On locked cycles, is the size of the line programmable in the GT-64120 PCI cache line size register, or is it fixed? If Fixed, what is the size?

Answer (last updated December 18, 2000):
On locked cycles the size of the line is fixed and it is 32 bytes.

18. Intelligent I/O (I20) Standard Support:

18.1 Overview

Question:
The GT64120 supports I2O on 32 bit PCI_0 bus or 64 bit PCI_0 bus. Right? I'd like to make sure we can use I2O on 64 bit PCI.

Answer (last updated December 18, 2000):
Yes. you can use 64120 I2O when it is configured to 64 bit PCI mode.

18.2 I2O Registers

Question:
If GT-64120 PCI_0 RAS[1:0] is remapped (Register C48), the I2O registers remain at PCI_0 RAS[1:0] space (Register 0x010), right ?

Answer (last updated December 18, 2000):
The remap register affects only the address PCI slave access the memory interface unit - it has no affect on the address decoding itself.

The GT-64120 PCI slave decodes the transaction address against RAS[1:0] BAR and size register, and only then if there is a "hit", it does the remap action.

If the transaction address is in the 1st 4K of RAS[1:0] (based on BAR and size reg), and I2O is enabled, the transaction will be directed to I2O registers and not to DRAM so remap registers are out of the game in this case.

If the transaction address is one of the two I2O ports (offsets 0x40,0x44), the transaction is directed to RAS[1:0] based on Queue Base Address Register and the relevant head/tail pointer. the remap register is out of the game in this case also.

Question:
What is the size of the GT-64120 I2O register space when enabled ? 4K, or 128 bytes.

Answer (last updated December 18, 2000):
When I2O is enabled, first 4K of DRAM0 space is used for I2O regs, which means that any access to that area will NOT cause access to DRAM. however only part of this 4K space is occupied by I2O regs. the rest are reserved addresses.

18.3 Circular Queues

Question:
There is an interrupt asserted when there is an overflow in the GT-64120 Outbound Free Queue, but there is not this same overflow interrupt asserted for any of the other three queues. What is the reason we did not include this interrupt for the other queues?

Answer (last updated December 15, 2000):
The GT-64120 I2O implementation is i960RP complient (we did exactly what Intel did in the i960RP). The inbound post queue will continue writing..even if it is full. It is responsibility of the i960 to ensure it manages overflows. The outbound free queue is a little different. It generates an NMI to the i960 to tell it the queue has overflowed. The outbound free contains the message frames that the i960 can use. It does not contain data for the i960 to process. The I2O spec clearly says how many outbound free message frames are to be created.

Question:
Maybe it's obvious how you enable the CPU interrupt for Inbound queue write. Which GT-64120 registers affect operation of this interrupt, and how should they be configured?

Answer (last updated December 15, 2000):
PCI write to GT-64120 Inbound Port Queue (offset 0x40 of RAS[1:0] BAR space when I2O is enabled) will cause bit [4] of Inbound Interrupt Cause register to be set, and if interrupt is not masked (by bit [4] of Inbound Interrupt Mask reg) a CPU interrupt is generated. interrupt is deasserted when CPU software clears the bit in the cause register (write 1 clears).

Question:
We will not use SDRAMs with the GT-64120. Then can we NOT use the circular queues. Even in this case we CAN use In/Outbound message register0/1, In/Outbound Doorbell registers and In/Outbound Interrupt Casue/Mask registers. Right?

Answer (last updated December 18, 2000):
Right. If you don't use SDRAM in you system, you can still use the above registers. you can't use the circular queues. Just make sure that RAS[10] BAR is enabled (although you don't use SDRAM, don't disable it through BAR Enable Register, since these registers are mapped in 1st 4K of RAS[10] BAR space).

Question:
What happens if the GT-64120 I2O outbound queue becomes full, and we write to it from the PCI side?

Answer (last updated December 18, 2000):
I assume you are talking about the Outbounf Free queue. if queue is full you can NOT write to it - you will get STOP* (retry).

19. DMA Controllers:

19.1 DMA Controllers

Question:
Can I do GT-64120 DMA from PCI_0 to PCI_1, or must the data first go into the SDRAM before being sent to the other PCI? What if DMA hits an unmapped address -- will it abort or just keep going (and log error in Bus Error register) with DMAOut bit in the interrupt Cause register set?

Answer (last updated December 18, 2000):
You can do direct DMAs from PCI_0 to PCI_1, without placing it in the SDRAM first. If the DMA hits unmapped address space it will keep going with wrong data.

19.2 DMA Channel Control Register

Question:
I am trying to generate a single 64 bit PCI access with the GT-64120 DMA controllers. The problem is that if I set up the DMA controller to transfer 8 bytes the result is two 32 bit accesses. I don't get 64 bit accesses until the data transfer limit is 12. Is this correct?

Answer (last updated December 18, 2000):
Your observation is correct: Not every DMA transfer size results in a 64 bit access. The rule is: If the Data Transfer Limit per each DMA access is more than 8 bytes, the access will be in 64 bits (in your case 12 > 8). Otherwise (Data Transfer Limit is less or equal to 8 bytes) the access will be in 32 bits. The Data Transfer Limit is programmed by setting the DatTransLim bits in the DMA Channel Control register. The reason that exactly 8 bytes are still performed via two 32 bit accesses is that you will not get a performance improvement if it was a single 64 bit access, due to the overhead involved.

19.3 DMA Design Information

Question:
When and in what sequence will the records of a Chained Mode DMA be fetched by the GT-64120?

I am designing an FPGA to be used on the device bus of the GT-64120. This FPGA will also control the DMAReq* line(s). The DMA descriptor will be read from the FPGA's 32 bit FIFO.

With a Demand Mode DMA and with the Fetch Next Record bit enabled, when will the first record of a Chained Mode DMA be fetched? In what sequence will the descriptor be fetched from the FPGA 32 bit FIFO?

Answer (last updated February 2, 2000):
The descriptors are fetched with no relation to DMAReq*. As soon as the DMA channel is enabled to chain mode with fetch next record set to '1' the channel will fetch the first descriptor. After it finishes the byte count of the descriptor it will fetch a new descriptor, again, with no relation to DMAReq* assertion.

The order of the fetch of the descriptors are: (1) Byte Count, (2) Source Address, (3) Destination Address and (4) Pointer to the next record.

Question:
We are interfacing the 64120 to EXTERNAL FIFOs on the device bus. If data is being transfered with PCI, wont this use only ONE of the INTERNAL DMA 64-byte FIFOs? We need to ensure the SECOND INTERNAL FIFO is available to support a critical path from an EXTERNAL FIFO DEVICE to DRAM. Uncontrolled delay on the PCI bus could clog the INTERNAL FIFO(s) and cause the EXTERNAL FIFO to overflow and lose data.

Answer (last updated December 18, 2000):
When a channel is programmed to do DMA from some source destination (PCI in your case), it will always use a specific fifo (channels 0,1 share one fifo, channels 2,3 share the other fifo), so the other fifo is free to serve other DMA transfers (device to dram in your case).

Question:
From the GT-64120 data sheet, "Fly-by: The SDRAM adress must be written to the source address register of the DMA regardless of direction". Does this mean that the Device address must be written to the distination address register regardless of deirection?

Answer (last updated December 18, 2000):
No need to write destination address. on fly by, Memory Interface Unit does not controls devices address, only device control signals.

Question:
How may we execute dynamic DMA chaining with the GT-64120?

Answer (last updated December 18, 2000):
Dynamic chaining is when you want to add DMA records to a DMA chain on which the GT-64010 DMA engine is working on. The problem is to synchronize between when the GT-64010 reads the last chain record (the NULL pointer) to the time the CPU changes the current last DMA record. Following is an algorithm I think provides this synchronization mechanism.

```
  1. Prepare the new record.
  2. Change the last record's NRP to point to the new record.
  3. read the DMA control register.
     If the DMAActSt bit is 0 (not active)
     {
         Update the NRP register in the GT-64010 and
         Write the Fetch Next Record
     }
  4. else
     {
   read the NRP register from the GT-64010.
        if it's equal to NULL (0)
{
Mark (by using a flag or something) that in the next DMA
chain complete interrupt you'll need  to -
[[[[{
         Update the NRP register in the GT-64010 and
         Write the Fetch Next Record
}]]]]
     }
  }
```

20. Interrupt Controller:

20.1 Interrupt Cause Register

Question:
Since there is no PCI 1 Int1* interupt, how does the GT-64120 report certain error conditions on PCI bus 1? What is your recommendation for having an Interrupt on PCI_1?

Answer (last updated December 18, 2000):
Int1* on PCI1 and JTRST* were replaced by Vref0 and Vref1 (for the PCI 5/3.3 V signaling). Use Int_0 for both PCI busses.

21. Reset Configuration:

21.1 Reset Configuration

Question:
What effect does the GT-64120's 66 Mhz config bit (DADR9 at reset) have?

Answer (last updated December 18, 2000):
It only determines the reset value of 66MHz capable bit of status & command configuration register.

Question:
The Req64* is being pulled down for GT-64120 reset configuration. On the PCI motherboard, this signal is supposed to be pulled-up for correct operation. Having a permanent pull-down on the PCI expansion board is therefore not possible. How are you supposed to manage this conflicting requirement? Especially as the config inpiut is supposed to be stable 3 clocks after reset deassertation.

Answer (last updated December 18, 2000):
Unlike other configuration inputs, Frame1*/Req64* configuration input requires zero hold time with respect to RESET rising edge. This means that in order to resolve the conflict you can use a tri-state driver. Connect its input to GND. The output is connected to Frame1*/Req64* and to the pullup resistor (required by PCI). Then connect the output enable of the tri-state driver to Rst* such that when Rst*=0, the driver drives (LOW since input is grounded) and tri-states when Rst*=1.

Question:
When does the 64120 reset configuration latch the REQ64* input for determining the PCI bus width?

With a 50MHz TClk. 3 TClk cycles is 60ns and is past the hold time of REQ64* (0-50ns) at reset de-assertion per PCI specification.

Doesn't the 64120 latch the REQ64* line at reset de-assertion and the 3 TClks are for synchronization. Note that the data sheet states that the reset configuration is conintuously sampled untle 3TClks after Rst* is deasserted.

We are using the 64120 as a 64bit PCI controller for an I/O card within a system. The Galileo is one of 4 PCI controllers within a system.

On page 91 of the 64120 0.2 spec, it states that the "configuration pins are continiously sampled from Rst* assertion until 3 TClk cycles after Rst* is deasserted". I have a question reguarding the sampling of the Frame1*/Req64* line and compliance with the 2.1 PCI spec. The Req64* line is sampled by the 64120 to determine the width of the PCI bus...0=64 bit, 1=32bit. On Page 134 & 140 of the PCI 2.1 spec, it states that the central resource (motherboard, not 64120) must drive the REQ64* line low during active RST*, releasing it within 50ns of de-assertion of RST#. The devices (64120) use the reset-time assertion of REQ64* to determine if a 64-bit data path exists. If the devices sample the REQ64* line low, a 64 bit data path exists.

My question is, when does the Galileo latch the REQ64* input for determining the PCI bus width? We are using a 50MHz TClk. 3 TClk cycles is 60ns. This is past the hold time of REQ64* (0-50ns) at reset de-assertion.

Ideally, the Galileo would latch the REQ64* line at reset de-assertion. This is stated at the bottom of page 140 within the 2.1 PCI spec (in footnote 36)..."This allows REQ64# to be sampled on the deassertion edge of RST#"

If the 64120 latches the state of the REQ64* line at 3 TClks, we will have to add external circuitry to our I/O card to drive the state of the REQ64* line seen by Galileo past the time defined by the PCI spec. This would force us to violate the PCI spec by adding loads/circuitry to the PCI signal lines.

It seems that Galileo tried to use this feature of the PCI spec to automatically determine the width of the PCI bus, but it fails to conform to the spec from what I can tell. I see the evaluation board also relies on the central resource to assert the REQ64* pin for cinfiguration purposes...that's what I would like to do.

Answer (last updated December 18, 2000):
The GT-64120 Req64# sample was designed so that there is no need for hold time on this signal after reset rise (not like the rest of the 64120 configuration pins that require extra tclk cycles after reset rise).

Question:
Shouldn't each GT-64120 PCI interface have a seperate reset pin?

Answer (last updated December 18, 2000):
In 64120 spec, rst* pin is mentioned as part of PCI_0 bus. this is not correct. rst* is a global reset to 64120 for all interfaces (PCI_0,PCI_1,CPU). It is not absolutly PCI complient, since PCI requires rst* pin per PCI bus, but during the

design we came to the conclusion that we can't support seperate rst* pins - so when you work with 64120 you need to reset ALL your system with the same single rst* signal.

Question:
For the GT-64120 is there a pin-strapping option which will specific the class code and subclass code of the GT-64120? The lastest spec. says the default is host bridge, but this may cause some problems with PC applications where BIOSs do not allocate resources for the class code.

Answer (last updated December 18, 2000):
Yes, there is such a pin-strapping option. the pin is Dadr[11] - if sampled 0 our class code is 0580 (Memory controller), if sampled 1 our class code is 0600 (Host bridge).

Question:
Ready* is sampled for a multi=GT address ID. What if Multi-GT mode is disabled by DAdr[10]. Will Ready* still be sampled by the GT-64120?

Answer (last updated December 18, 2000):
No.

Question:
GT-64120 Ready* and CStiming are tied HIGH for 11. But, Ready* tied HIGH will give a major problem as the devices will never be ready. If Multi-GT mode is ENABLED, and Ready* is tied HIGH, and Ready* is not implemented by some EPLD or anything, then the system will never boot! ??? Is this correct? How can we drive Ready* LOW after the RESET?

Answer (last updated December 18, 2000):
One way to do it is to connect the reset pin, inverted, to the Ready*.

Question:
The Int* pin of the CPU interface is supposed to be pulled-down in our application. Is that output open-drain from the 64120, or is it in fact a totempole output (after config)?

Answer (last updated December 18, 2000):
The Int* signal of the CPU Interface, serves as configuration input during RESET and turned to a pure output (not open-drain) after RESET deassertion, which would typicically drive HIGH (inactive). This is true both for the GT-64120 and the GT-64130.

22. Connecting the Memory Controller to SDRAM and Devices:

22.1 Devices

Question:
Regarding the device interface of the 64120-A. The tables in the “Connecting Memory Controller” section of the data sheet imply to me that for a multiple byte wide device, 32 bit (quadword) for example, the demultiplexed device address on A[29:0] is NOT a byte address but an address of the unit of the bus width of data. For example:

Little Endian System (CPU and devices)

32 bit wide device:

CPU BYTE write to CPU byte address offset 00000007h .

Demuxed Galileo device address [29:0] = 00000001h .

Wr[0:2]* inactive

Wr[3]*   active

Is this correct?

Answer (last updated December 19, 2000):
Section 13.2 describes the connection of the device bus to all different bus width devices. The address bus in each table is a concatenation of a portion of AD[31:3] and BAdr[2:0] where the AD portion is the most significant address bits and BAdr is the least significant address bits.

If to summarize in a table -

Table 1:

	64 Bit	32 Bit (odd)	32 Bit (even)	16 Bit (odd)	16 Bit (even)	8 Bit (odd)	8 Bit (even)
AD	31:6	31:5	31:5	31:4	31:4	31:3	31:3
BAdr	2:0	2:0	2:0	2:0	2:0	2:0	2:0
WR#	7:0	7:4	3:0	5:4	1:0	4	0

The usage of even and odd banks is an option for load balancing on the AD bus.

Per your example a write to byte at address 0x0000.0007 will result in

AD[31:5] == 0x0

BAdr[2:0] == 0x1

WR#[3:0] == 0xE

The memory map of the devices is as follows -

Word Address Byte 3 in a word Byte 2 in a word Byte 12 in a word Byte 0 in a word

0x0 Byte 3 Byte 2 Byte 1 Byte 0

0x1 Byte 7 Byte 6 Byte 5 Byte 4

0x2 Byte 11 Byte 10 Byte 9 Byte 8

0x3 Byte 15 Byte 14 Byte 13 Byte 12

23. JTAG Application Notes:

23.1 JTAG Application Notes

Question:
Which JTAG opcodes are supported by the GT-64120?

Answer (last updated December 18, 2000):
The GT-64120 supports the JTAG opcodes defined by the IEEE Std. 1149.1 (1994):

highz (0011)

idcode (0010)

extest (0000)

sample (0001)

bypass (1111)

Question:
We are chaining the JTAG ports of 64120B-B-4 and a few FPGA's on our board. We are to use this JTAG bus to program the FPGA. We put the GT-64120 in reset and expect our JDI/JDO to simply pass data through but we are not seeing data coming through. If 64120 is put into reset, are all its pins tri-stated, including JDO, so this would not work? Is there a way to make this work?

Answer (last updated January 12, 2001):
The problem seems to be putting the GT into reset. All this time the reset of the device should be high.

Put the device jtag state machine to bypass mode, using TMS, TDI and TCK and then program your FPGA using the one cycle delay path between TDI to TDO.

24. Big and Little Endian:

24.1 Background

Question:
With the GT-64120 in 64-bit PCI mode, if ByteSwap is set, does byte swapping occur across the 8 byte lanes?

Answer (last updated December 18, 2000):
Yes.

24.2 Configuring a System for Big and Little Endian

Question:
Regarding GT-64120 byte swapping:

If the PCI is 64 bit, and is set for big endian, how is the data byte swapped? I.e., if a 64-bit double word is presented on the PCI bus as: 0x12 34 56 78 9A BC DE F0 and it is a memory transaction, will is be written byte swapped as:

0xF0 DE BC 9A 78 56 34 12 or 0x78 56 34 12 F0 DE BC 9A?

In other words, is it byte swapped on 64-bit alignment or 32-bit? Possibly its obvious, but I don't know...

Answer (last updated December 18, 2000):
Byte swap is 64-bit alignment.

25. Using Different PCI Configurations:

25.1 Using the GT-64120 in Different PCI Configurations

Question:
Please explain how to configure the 64120 PCI bus Vref0 and Vref1 pins for 64-bit mode. Also, please explain how to configure the 64120 PCI bus pins in the different PCI modes.

Answer (last updated December 18, 2000):
When running in 64 bit PCI configuration you need to connect VREF1 to the same voltage you connect VREF0. The reason is part of PCI1 bus pins are multiplexed and are used in PCI64 configuration.

The Reset Configuration chapter of the data sheet shows how to configure to different PCI modes.

26. Register Tables:

26.1 General

Question:
What is the maximum access time for the CPU to read internal registers? Is there a deterministic upper bound to the access time to the INTERNAL REGISTERS of the GT-64120 when doing a read from the CPU interface bus?

Answer (last updated February 2, 2000):
There is no special arbiter for internal registers.

The maximum time to read an internal register varies according to the register that is being read. A register from the CPU-unit may be read faster than a register from the Memory-unit.

It is difficult to determine the maximum access time because it depends on the state machines of the internal units of the GT-64120. For example, state machines in the Memory Controller may be stalled if Ready* is de-asserted.

GT-64130 FAQs

27. CPU Interface Description:

27.1 Multiple GT-64130 Support

Question:
We have added our own ASIC (slave only) on the CPU bus of the GT-64130. What are the issues when connecting an ASIC along side the 64130 that is in “MultiGT"? What happens when AACK and TA are asserted on the same cycle? Why can there be a situation that the 64130 is generating an AACK even though there was an AACK already generated, and moreover, the address is not a hit in the GT-64130? Why is the GT-64130 responding to an address that is not his?

Answer (last updated December 19, 2000):
What happens when AACK and TA are asserted on the same cycle? The GT-64130 can't accept a situation that TA is asserted before AACK. But what happens when they are asserted at the same time? This is a marginal case, and it is hard to predict. It depends on paths inside the GT-64130. For this reason, we should regard this as a situation that the GT-64130 can NOT handle. This situation should not be generated in a system with the GT-64130. The basic reason is that the GT-64130 starts "hunting" for a TA only after it sees an AACK. It should be reemphasized that the GT-64130 MultiGT state is regarded by the GT-64130 as a few GT-64130s connected together. Under this assumption, the GT-64130 state machines were built. Because GT-64130 has a multiplexed bus working with Quick Switches, there is no situation that the GT-64130 will generate a TA before, or even with an AACK.

Why can there be a situation that the GT-64130 is generating an AACK even thought there was an AACK already generated, and moreover, the address is not a hit in the GT-64130? The assumption while building the GT-64130 state machines was that MultiGT regards a situation of multiple GT-64130s connected together. The state we are seeing is: the system is working in Pipeline TSs, meaning a TA is generated even before the previous cycle has finished. This is fine, but, the following cycle is generated toward the EPLD and not to the GT-64130. An AACK is asserted by the ASIC, even though the previous TA was not yet accepted. This could not have happened if there were two GT-64130s because the quick switch would have still been generating the data of the previous cycle, and not showing the address of the current TS. Consequently, the GT-64130 is not listening to the AACK until TA of the previous cycle.

Why is the GT-64130 responding to an address that is not his? As a result of the AACK being already generated, the address bus is no longer valid. There might have been a transition state that caused the GT-64130 to regard an invalid address as a valid address that caused a hit. The system designers should make sure this situation does not occur, i.e. the EPLD should be listening to the TA of the previous cycle. Only after it has been asserted, may the EPLD respond with an AACK of its own.

27.2 PowerQUICC II Support

Question:
This is strange, as all the timing diagrams in the PQII data sheet have (for single beat reads and bursts), DVAL asserted with TA. I'm looking at a timing diagram with a four beat burst (64 bit wide) and it asserts DVAL.

This would be good news if we can pullup DVAL, but I'm skeptical, as all the correspondence with Motorola made me believe this is the case. Has anyone connected a 603/750 to the PQII 60x bus and seen that this works?

Answer (last updated January 3, 2001):
This is a question for Motorola, not for Galileo. There is a working example posted on the web at http://www.mot.com/SPS/RISC/netcomm/tools/ Look for MPC8260 then look for scout board.

PSDVAL* signal is a signal needed if one needs dynamic bus sizing. If one can live with transfer of 64bit only for its transfer one need not to assert PSDVAL*. For example if your external master is another PQ2 then you would like to connect the PSDVAL* but if the external master is an 603e since this device does not support dynamic bus sizing you need not to assert PSDVAL*.

Just in case: A dynamic bus sizing is an ability for the bus controller to support multiple of bus size with out software restriction. This for example allow 64Bit data to be transferred to a 32bit device in two bus cycle automatically with out a need for software routine.

The reason you see PSDVAL* asserted every cycle is because the bus is transferring at normal bus cycle. If it was transferring for example 4 x 64 bit burst to a 32 bit bus you will not see this. (again 603 nor 7xx do not support this type of transfer).

27.3 CPU Interface Restrictions

Question:
Although we have had G4 7400 CPU's working on our boards with the GT-64130 for a while, one of our contractors stumbled onto an anomaly. With the 60x/7x0 CPU's data transfer's which are not aligned to a 32 bit boundary are broken into two cycles by the CPU. On the 7400 the CPU seems to do this only for transfer's which are misaligned on a 64 bit boundary. The example our contractor provided was trying to write a word (4bytes) to memory address 100002. This caused corruption of the data words at 100000 and 100006. I ran the simulation and this does in fact seem to be the case. Is there something I am missing here? Can the GT-64130 be setup to support misaligned transfers within 64 bit values?

Answer (last updated December 19, 2000):
The GT-64130 does not support access of 4 bytes that cross a 32-bit boundary. Those accesses will be translated as 8 byte writes. There is no problem with the MPC750, since it brakes such an access into to two transactions on the bus. There is no work around at the chip level. There might be at the software level. Discovery supports this however.

28. Memory Controller:

28.1 SDRAM Controller

Question:
Typically SDRAMs do not respond to memory accesses for about 70ns after the start of a refresh cycle. Does the GT-64130 / GT-64131 automatically wait for a certain period of time after it executes a refresh cycle to the SDRAM? What is the time value ? Is this period of time programmable ? How long does the SDRAM controller on the GT-64130/GT-64131 hold off memory transactions to the SDRAM when performing a refresh cycle?

Answer (last updated January 3, 2001):

Yes. the GT-64130 / GT-64131 automatically waits for a certain period of time after it executes a refresh cycle to the SDRAM. This time is 10 Tclk cycles. The SDRAM controller on the GT-64130/GT-64131 holds off memory transactions to the SDRAM when performing a refresh cycle. The transactions are held of 5 clock cycles before and 10 clock cycles after the refresh. These hold off times are not programmable.

28.2 SDRAM Performance

Question:
Would enabling ECC have a performance impact? If yes, by how much?

Answer (last modified August 22, 2000):
ECC adds one clock of latency. For example, the CPU to SDRAM read performance for the GT-64130 with bypass not enabled is 11-1-1-1 instead of 10-1-1-1.

28.3 Unified Memory Architecture (UMA) Support

Question:
The GT-64130 data sheet mentions a “Total Request” signal, than can be used in a VUMA architecture. Could I get a little bit more info on it? How is this to be used? Should this signal be used in conjunction with the MGNT? Does it replace MGNT completely?

Answer (last updated March 6, 2000):
In UMA configuration the partner UMA device can hold the bus indefinitely after it is granted ownership of the bus. On the other hand the GT-64130 after granting the bus using the MGNT* signal will immediately request the bus, by de-asserting MGNT*, regardless of a need for the bus. That is the GT-64130 can be totally idle and still after granting the bus will request ownership of the bus. In order to allow for external hardware to implement a better and more fair arbitration scheme the GT-64130 can be configured to assert the TREQ* signal which is a qualifier for the MGNT* request. The TREQ* is asserted only when there is a pending request from one of the GT-64130 interfaces (i.e. - CPU, PCI or DMA) to the memory controller. The TREQ* is asserted from rising edge of TClk and should be sampled by rising edge of TClk. The timing in cycles will differ according to originating entity that has issued the request for the memory controller.

28.4 Error Checking and Correcting (ECC)

Question:
When the 64130 determines a memory error and the interrupt controller interrupts the CPU. When the CPU reads registers 0x480 - 0x490 for status are they cleared? If no, how do they get cleared?

Answer (last updated January 12, 2001):
The GT-64130 registers are not cleared automatically. When they are read, following an interrupt they get "unlocked" and they can be overwritten either by writing to them (normally no need for that) or by a new interrupt, should this occur, whereas their contents are updated with the new data.

29. PCI Interfaces:

29.1 PCI Interfaces

Question:
Are you familiar with apps using the GT64130 to bridge between the two PCI busses?

Answer (last updated December 19, 2000):
There are two options:

1) Use one of the GT-64130 DMAs to master xfer from PCI_0 to PCI_1

2) Have the external PCI master move data into SDRAM from PCI_0, then use a DMA to push it to PCI_1 (or have ext master pull it).

29.2 PCI Master Operation

Question:
A DMA descriptor is initiated to transfer 5 32bit words. If I initiate the DMA control register to do a DMA write (from local memory to PCI) of 5 32bit words, 5 32bit words are written on the PCI bus. If I initiate a DMA control register to do a DMA read (from PCI to local memory) of 5 32bit words, 6 32bit words are read on the PCI bus. The same thing happens with 7 words, the DMA read does 8. Is this observation correct?

Answer (last updated January 3, 2001):

This is explained by the GT-64130 design: The internal data path between the DMA engines and the PCI is 64 bits. Consider write from a DMA source to 32 bit PCI of an odd number of 32 bit words. The DMA will pass to the PCI unit data in 64 bit chunks with the last chunk marked with Byte Enabled active on just 32 bits. The PCI unit knows how to translate only that data with Byte Enable active to actual 32 bit PCI writes. This is why in your DMA write (from local memory to PCI) of 5 32bit words, 5 32bit words are written on the PCI bus. Consider read from a DMA source from the 32 bit PCI of an odd number of 32 bit words. The DMA will request data from the PCI unit data in 64 bit chunks without any Byte Enables. This means that the PCI unit will always translate these requests to an even number of 32 bit read accesses on the 32 bit PCI. This is why in your DMA read (from PCI to local memory) of 5 32bit words, 6 32bit words are read on the PCI bus.

29.3 PCI Master Configuration

Question:
I am having problems making config 1 cycles on PCI. Config 0 cycles works fine, but when I set the Bus number to something different from zero, the config enable bit (bit 31 in the config address register (offset 0xcf8)) seems to stay on when I look at the access on PCI: For example I would like to do a config 1 read access from address 0x20001 (where the 1 indicates config type 1). I write 0x80020000 to offset 0xcf8, and tries to read the data from offset 0xcfc. When I check what happened on the PCI bus, I see a config 1 access to address 0x80020001, which is not what I expected. Do you know what I do wrong?

Answer (last updated January 3, 2001):

If bus number programmed in 0xcf8 is not 0 (in this example it is 0x2), the GT master initiates a type1 config transaction (which means bit[0] of the transaction address is set to 1). The PCI spec defines the address of config type 1 transaction as follows:

bits[1:0] - 01

bits[7:2] - reg #

bits[10:8] - function #

bits[15:11] - device #

bits[23:16] - bus #

bits[31:24] - reserved

It's probably true that the GT master drives bit[31] of the address to 1, but since it is part of the reserved field it is allowed. The PCI spec states explicitly that "targets must ignore reserved fields", so there is nothing wrong with the GT master behaviour (section 3.2.2.3.1 of PCI spec rev 2.2).

29.4 Target Configuration and Plug and Play

Question:
Regarding the PCI_0 Base address registers enable, at offset 0xc3c in the GT-64130. Why can't I disable the IntMeMEn bit (bit 4) for memory mapped access to internal registers? I would like to disable all base address registers but the SCS[1:0]En, so I write 0xff to the register, but when I read it back, I find that bit 4 is still zero, i.e. I read 0xfe, and I still have access to the internal registers from PCI.

Answer (last updated January 3, 2001):

This feature was designed in order to prevent from PCI to disable both memory mapped and i/o mapped internal registers BARs. The reason is that once you disable them both, you can not access this 0xc3c register anymore from PCI, so it is not reversable.

29.5 PCI Interface Restrictions

Question:
Given a GT-64130 configured as having (2) 32-bit PCI interfaces, Can (and how?) a device on one PCI port (say PCI_0) access devices connected to PCI_1? I think I can see how one would run config cycles across the 64130, but how to set up the registers to map from one port to the other?

Answer (last updated January 12, 2001):
There are two ways to move data from the GT-64130 PCI_0 to PCI_1:

1) Program a GT-64130 DMA to move the data, or

2) Move the data from PCI_0 to SDRAM, then from SDRAM move it to PCI_1.

There is no way for an agent on PCI_0 to do a PCI config cycle on PCI_1. Only the CPU can issue PCI config cycles on PCI_0/1.

You can always add a P2P bridge (such as an Intel 21152) to sit between PCI_0 and PCI_1. Also, our new GT-64260 has a built-in P2P bridge to do this.

30. Interrupt Controller:

30.1 Interrupt Cause Registers

Question:
When I read in the Interrupt Cause register (offset 0xc18) that a master abort has occured, how do I know if it was one of the DMA controllers, or an access from the processor that caused the master abort? Is there any other status register which gives this information? Is there any internal information that can be extracted following a Master-Abort, besides the fact that such Master-Abort occured ? Can you suggest any hints on how to get such information ?

Answer (last updated January 3, 2001):

Master-Abort event is triggered when devsel is not asserted as a response to the PCI Master cycle. The information of which PCI cycle, was aborted by the 64130 PCI Master cannot be extracted directly.

The only hint that can be used is the fact that the GT-64130 master returns in case of read data of 0xffffffff. So if it is a CPU read that got the master abort, it will read data of 0xffffffff. If it is a DMA from PCI to DRAM, the DRAM will be written with 0xffffffff.

We don't have any more debug information for master abort. Galileo's next generation system controllers (Discovery) may provide additional debug information.

GT-9601x FAQs

31. Ethernet Controller:

31.1 Ethernet Configuration Register (ECR)

Question:

During our hardware tests we have observed the RAC accessing the Ethernet Hash Table, even when the Ethernet Controller is in promiscuous mode. I would expect the RAC to completely ignore the Hash Tables while in promiscuous mode. Have I missed something?

Answer (last updated January 9, 2001):
Yes, RAC is accessing Hash Table in promiscuous mode. It than marks frame with the appropriate M and HE bits although do not discard frames.

The CPU can use this info to distinguish between frames that were received due to promiscuous mode, to frames that were directed to the CPU.

GT-961xx FAQs

32. Communication Interface Unit (CIU):

32.1 CIU Arbiter Configuration Register

Question:
We are having trouble getting the GT-96100A IDMA starting. Any ideas?

Answer (last updated January 9, 2001):
With the GT-96100A IDMA, you should be aware of a common programming error. There is a register called CIU Arbiter Configuration Register, Offset: 0x101AC0. The default value for the "IPL" IDMA Priority Level field is b'00' which masks IDMA requests to the arbiter and prevents the IDMA from ever starting. This field MUST be changed during initialization to have the DMA engines work.

33. Serial DMA (SDMA):

33.1 SDMA Descriptors

Question:
I have a question about the GT-96100A descriptor endianess. On our hw, CPU is BIG endian and your GT is LITTLE,

Assume I set the bit 12 to 0 (BIG) of the CPU interface ..

1, Is my memory layout (all my Mips CPU read and write) is still BIG?

2. If I set the CIU descriptor endianess to BIG then .. What is the proper descriptor structure look like?

The doc says:

1st WORD Command/Status

2nd WORD Byte Count / Reserved

3rd WORD Buffer Pointer

4th WORD Next Descriptor Pointer

But in real life, we need to declare the structure as..

1st WORD Byte Count / Reserved

2nd WORD Command/Status

3rd WORD Next Descriptor Pointer

4th WORD Buffer Pointer

Is it true? Can you explain that?

Answer (last updated January 12, 2001):
1) GT does not byte-swap for SDRAM accesses. The cpu reads data in whatever endianess it has written it into the memory. In your case, it is big-endian.

2) It should still be:

1st WORD Command/Status

2nd WORD Byte Count / Reserved

3rd WORD Buffer Pointer

4th WORD Next Descriptor Pointer

Remember, in going from little endianess to big endianess, we only changed the positions of the words, but not which we call most/least significant. Lets say we have a 64-bit word, the byte orders are as follow:

Little endian: -7-6-5-4-3-2-1-0-

Big endian equivalent: -0-1-2-3-4-5-6-7-

The zeroth word in the little endian format is -3-2-1-0-. The zeroth word in big endian is -0-1-2-3-.

33.2 Transmit SDMA

Question:
In Tx SDMA description, in one of the paragraph reads "when the tx SDMA controller encounters a Not-Owned Desc with it's First bit SET, after the last desc of the frame, tx IDLES.'

In the next paragraph it reads "when the tx SDMA controller encounters a Not-Owned Desc with it's First bit RESET, after the last desc of the frame, tx ABORTS.' It seems to me that the first case is ABORT case and the second one is a NORMAL case. After tx of a frame, why would the next desc need to have it's First bit set? Can you clarify this.

Answer (last updated January 12, 2001):
The description is correct.

There are two ways to organize your descriptor list. One is linearly. You set up the first descriptor, set the F bit and in the NDP specify where the next descriptor is. In the second descriptor, you will both F and L bit reset and the NDP point to still another descriptor. This goes on until you hit the last descriptor in which you set the L bit and the NDP to null. So when 96100A finishes the last descriptor and it will not try to fetch the next one and stop.

Another way of doing it is to have a circular list. You do the same things as in the linear case except in the last descriptor, you have the NDP points to the first descriptor in the list. In this case, when 96100A fetches the next descriptor after finishing with the last frame, it expects to see the F bit set in the new descriptor. If it is not set, the circular list is assumed broken and a resource error interrupt is raised. If the F bit is set but the cpu hasn't released the descriptor by not setting the owning bit, the DMA will halt.

34. FlexTDM Units (FTDM):

34.1 FlexTDM Synchronization

Question:
I am trying to get a BRI wic to work with FTDM on Galileo. We have two BRI wics and both interface with GT using IOM-2 standard. I am able to get one of the WICs working, but I have encountered problem with the other wic.

I have taken some Logic Analyzer samples from the working wic (BRI-S/T uses Siemens 2186 chip) and from the non working wic (BRI-U uses Siemens 2091 chip). One FTDMxRINT/FTDMxTINT is generated each time an entry in the Rx/Tx dual port is executed, if the FTINT bit (bit 26) of that entry is set. Once I enable FTDM, I am getting one FTDMxRINT and FTDMxTINT, and after that I get lots of FTDMxRSL and FTDMxTSL. I have done H/W probing and I do see the clock & sync going to the FTDM properly. I have different combination of DE, SE & RSD, but nothing helped.

According to the manual, FTDMxRINT/FTDMxTINT is generated if FTINT bit is set in the current entry AND FTINT bit is not set in the previous entry that it executed. That still doesn't explain why I am getting an FTINT, as all my entries in the Dual Port have an FTINT bit set. So, I should not be getting FTDMxRINT/FTDMxTINT

Since the WIC is giving both the CLOCK and the SYNC, so wic cannot run faster/slower than GT.. We have done some Logic Analyzer probing, and saw that we do get a SYNC at the expected times and the width of the sync conforms to the FTDM timing diagrams.. SO, not sure why I am getting these FTDMxRSL/FTDMxTSL...

Question is, when and why are FTDMxRINT and FTDMxTINT generated and why am I getting FTDMxRSL and FTDMxTSL.

Another difference between the working wic & non-working wic is the length of the SYNC pulse, which is a whole lot wider on the working wic. According to the 96100A databook, the TRSYNC's rise & fall time needs to be a maximum of 15ns, It looks to me it's a little more than that.. Would that be a problem.

Answer (last updated January 12, 2001):
FTDMxRSL/FTDMxTSL is generated when GT is not sensing SYNC at the clock edge that it expected. I think the non-working wic is probably running faster than GT expected, given the SYNC pulse is much narrower.

You should still get one interrupt (from the first entry) even when you have FTINT of all entries set. So the FTDMxRINT/FTDMxTINT you see is probably from the first entry.

If I read your waveforms correctly, it seems like SYNC has a rise time of slightly less than 28ns. It is way outside the 15ns we spec'ed. Is there a way to speed it up? It could be the reason why you're seeing loss of synchronization.

35. Physical Signal Routing:

35.1 Signal Routing

Question:
How is the actual physical routing of the MPSC and TDM signals defined in the GT-96100?

Answer (last updated January 9, 2001):
Table 2, “GT-96100 Port Configurations”, in the "Overview" chapter of the GT-96100 data sheet describes the optional configuration between the physical pins (i.e. port A, port B, ..) to the internal hardware. For example port C can be connected to either a GPP port or connected to FlexTDM 0 or MPSC 2. This is the physical connection from the system level to the available circuitry in the GT-96100. The MRR reflects the configurations available for the internal MPSC circuitry. Each of these MPSCs be connected either to one of the physical ports (i.e. port A through port F and MII0) or to any one of the FlexTDM's.

The MRR configuration in conjunction to the General Purpose Configuration register will determine if a physical port is connected to the MPSC or to a FlexTDM. For example to configure MPSC 2 to work with FlexTDM 2 you need to configure MRR<8:6> to 0x3 and configure port E to be connected to the functional pins by setting GPC1<22:16> to 0x7f. When configuring the GPP functionality to be connected to a peripheral port you should also configure the respective GPIO to reflect the correct behavior of the different pins of the physical port when connected to this peripheral. For port E to be connected to FlexTDM 2 the GPIO1<22:16> should be configured to 0x5 (PortE[0] - TTXD2, PortE[1] - TRXD2, PortE[2] - TDSTRB2, PortE[3] - TTSYNC2 PortE[4] - TRCLK2, PortE[5] - TRSYNC2, PortE[6] - TTCLK2).

Once port E is configured as peripheral function it can be connected either to MPSC4 or to FlexTDM2. It is up to the system designers to configure other MRR bits to reflect the fact that MPSC 4 is not connected to the physical port E.

The GT96100 counters can not be cascaded to one another.