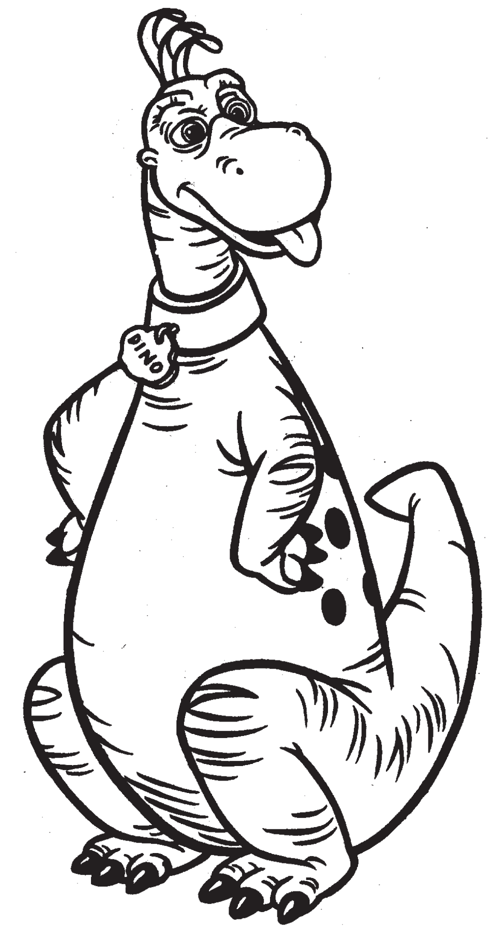
For:billk
Printed on:Thu, Jun 3, 1999 14:33:55
From book:dino_ers
Document:rev
Last saved on:Thu, Jun 3, 1999 14:14:00
Document:cover_page
Last saved on:Thu, Jun 3, 1999 14:31:06
Document:Contents
Last saved on:Thu, Jun 3, 1999 14:30:30
Document:Figures
Last saved on:Thu, Jun 3, 1999 14:29:58
Document:Tables
Last saved on:Thu, Jun 3, 1999 14:29:28
Document:intro
Last saved on:Thu, Jun 3, 1999 14:28:58
Document:overview
Last saved on:Thu, Jun 3, 1999 14:28:25
Document:registers
Last saved on:Thu, Jun 3, 1999 14:27:53
Document:bridge_regs
Last saved on:Thu, Jun 3, 1999 14:26:54
Document:operation_
Last saved on:Thu, Jun 3, 1999 14:25:34
Document:GSC
Last saved on:Thu, Jun 3, 1999 14:25:03
(...)

REVISION HISTORY PAGE

REV.	REVISION DESCRIPTION	DATE	INITIALS
0.001	Pre–released	6–13–94	
0.1	Pre–released	7–5–94	
1.0	Pre–released	12–9–94	
1.1	Pre–released	2–16–95	
2.0	Released	11–7–95	
2.1	Released	6–12–96	
3.0	Released	2–7–97	
3.1	Released	9–19–97	



DINO ERS

A GSC–to–PCI Bridge

Systems Technology Division

Fort Collins Systems Lab

Revision 3.1
September 19, 1997

Notice

The information contained in this document is subject to change without notice.

HEWLETT–PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE MATERIAL. INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Hewlett–Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with furnishing, performance, or use of this material.

Hewlett–Packard sassumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett–Packard.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Objectives

The primary objective for the Dino chip is to provide a bridge between GSC and PCI. Dino's throughput is high enough to support state of the art graphics and I/O functions on the PCI bus segment. Also, PCI and GSC do not need to be synchronized, making the system clock design simpler and more flexible, while maximizing the performance of both buses.

The primary design center for the Dino chip are the B132 , B180, C180, C200, and C240 desktop workstations. Dino provides access to industry standard PCI technology, giving customers the widest range of possible I/O options.

Additional functionality has been added to Dino for potential use in a future system. This functionality includes an RS–232 port, and PS2 support.

The primary objective for this document is to describe the software interface to the Dino chip with enough detail to allow driver development. This document (along with referenced documents) contains enough detail to ascertain the intended functionality of each block on the Dino chip down to the register level.

This version of the document represents the state of the chip as of the 3.1 tape release, which corresponds to the part number 1FC3–0004.

1.2 Feature Summary

The following is a list of the functionality provided by the Dino chip:

- Supports high speed graphics
- Implements GSC+ features
- Relocatable HPA
- Mapping register with **8MB** resolution
- Integrated PCI Arbitration
- Integrated Interrupt Register
- Coalesced DIO writes
- Implements GSC type "F" transaction
- Package: 208–PQFP
- >40MHz GSC operation (in limited configuration)

- >33MHz PCI operation (in limited configuration)
- PS2 interface (not used in Raven or Merlin)
- RS–232 port
- 3.3 V / 5.0 V PCI

2 OVERVIEW

2.1 Typical System Example

Dino is designed to work with a HP's newest family of CPUs and IOAs as a GSC guest, providing connectivity to PCI and some other miscellaneous functions.

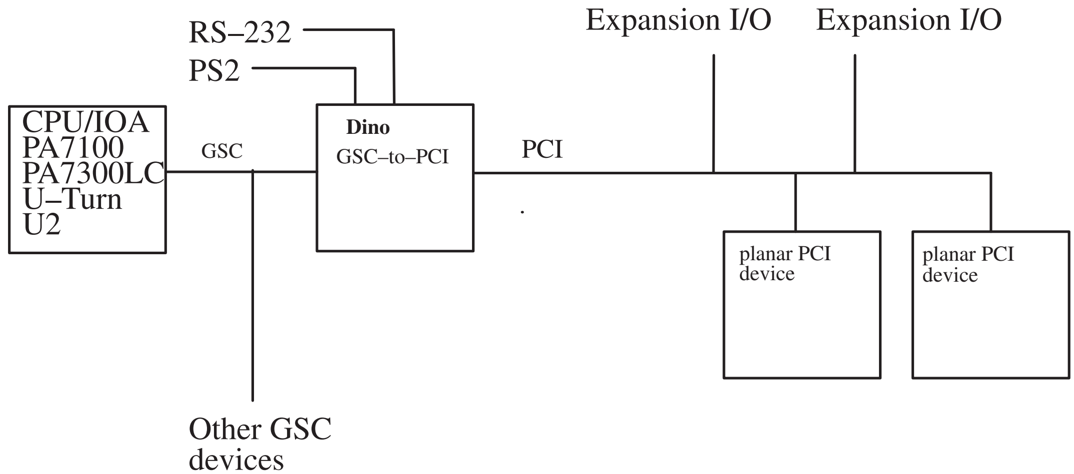


Figure 1. Example I/O Subsystem Block Diagram

2.2 Dino Chip Block Diagram

Dino's internal data path consists of one downstream address and data FIFO, one downstream data FIFO, one upstream address and data FIFO, one upstream data FIFO, and one upstream control FIFO. The downstream FIFOs hold addresses and data going from GSC to PCI, and the upstream FIFO moves addresses and data from PCI to GSC. The upstream control FIFO transfers transaction codes from the PCI to the GSC interface. These FIFO's bridge between the PCI and GSC clock domains allowing these busses to run at different frequencies. When enough data from PCI appears in the upstream FIFO a transaction is started on GSC. The same is true for transactions going in the other direction.

Arbitration controllers for GSC and PCI are independent, meaning that a transaction can be started on either bus without disturbing the other. The PCI and GSC interfaces arbitrate with their respective busses when data needs to be transferred. These interfaces communicate with each other by constantly watching FIFO status signals through synchronizers.

The miscellaneous register block contains bridge configuration and control bits and can only be accessed by word transactions from GSC. The interrupt block provides a convenient way of signalling GSC and PCI interrupts to the host CPU/IOA by way of GSC. Interrupt registers are part of Dino's HPA space.

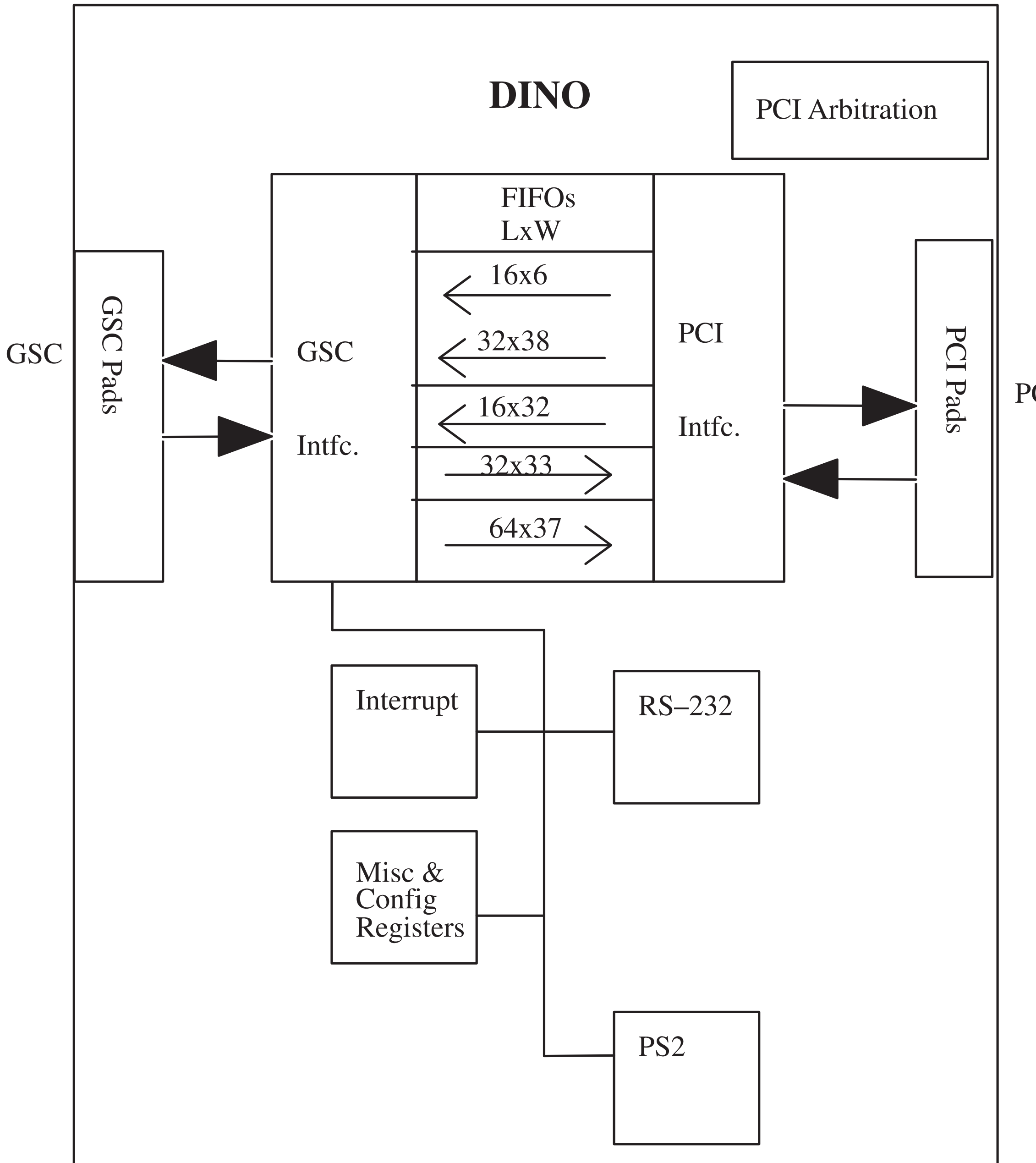


Figure 2. Dino Internal Block Diagram

3 DINO REGISTER OVERVIEW

3.1 Overview

Dino requires I/O address space for control and status registers for the GSC to PCI bridge, PS2 controller and RS–232 interface. Dino uses I/O device control (IODC) submodules to implement all of the above functionality under one IODC header. (See the *Precision I/O Architecture Reference Specification* for more details on IODC.) Bridge registers live in IODC submodule 0, PS2 registers live in IODC submodule 1, and RS–232 registers live in IODC submodule 3.

For PCI related registers, Dino will use the HVERSION Dependent Register Set registers (HVRSes) of its hard physical address (HPA) space.

Note that the GSC bus and the Dino hardware model use little–endian bit ordering, which is the opposite of the PA I/O architecture descriptions that use big–endian bit ordering. This document will provide both bit orderings for all registers, which will hopefully help clarify the situation!

3.2 Dino Broadcast Physical Address (BPA) Registers

Dino implements two registers within the BPA space. One register (IO_FLEX) provides a flexible bus identification and address space configuration mechanism as well as a universal DMA enable/disable function. The second register (IO_COMMAND) provides for a global reset capability. The Dino BPA register addresses are given in Table 1. The following subsections describe each of the BPA registers.

Register	Symbol	I/O Address	R/W	Description
IO Flex (Bus ID)	IO_FLEX	FFFC0020	W	The IO_FLEX register provides a flexible relocation mechanism for Dino's GSC address space, and controls the enabling and disable of Dino's GSC bus mastership.
IO Command	IO_COMMAND	FFFE0030	W	The IO_COMMAND register allows Dino to respond to global resets on GSC.

Table 1. BPA Registers

NOTE: Do not assert gscready_L on broadcast transactions.

IO_FLEX (IOFlex (BusID) Register)

	0 – 3	4 – 13	14 – 30	31
0xFFFC0020	1111	**BUS_ID**	00 0000 0000 0000 000	**EN**
	31 – 28	27 – 18	17 – 1	0

- BUS_ID: At power on the EN bit will be set to 0 and the bus host will broadcast IO_FLEX. Neither BUS_ID nor EN are affected by command reset.
- EN: The EN bit can be used to enable or disable Dino from arbitrating for bus mastership. To enable Dino bus mastership of the GSC bus, the EN bit must be asserted and Dino's mask bit in the GSC Arbitration Mask register must be cleared.

Until the the host broadcasts IO_FLEX , Dino's HPA page is not reachable, and no transactions will be forwarded to PCI.

The BUS_ID field is combined with Dino's GSC OFFSET[0:3] and a sub–module ID (always 00 for Dino's control HPA) to determine the address of Dino's 4kB control and status register page (the HPA registers). Dino determines the address of its HPA registers based on the following address structure:

	31 – 28	27 – 18	17 – 14	13 – 12	11 – 0
HPA GSC Addr:	1111	**BUS_ID**	**OFFSET[0:3]**	submod	HPA Reg. Offset

Note: OFFSET[0:3] is the same as the slot address for the GSC device (gscsl[0:3]). BUS_ID is sometimes called "FLEX".

IO_COMMAND (IO Command Register)

	0 – 15	16 – 23	24	25 – 31
0xFFFE0030	**Reserved**	**CMD_DEP**	0	**CMD**
	31 – 15	15 – 8	7	6 – 0

- CMD: Dino recognizes two commands: CMD_RESET (CMD=5) and CMD_CLEAR (CMD=3). All other fields are ignored. Dino will set its IO_CONTROL register to OFF mode upon receipt of a CMD_RESET. CMD_RESET will reset the state of the bridge's GSC interface and be forwarded to the PCI bus. Dino transactions within 16 states of CMD_RESET are not allowed. CMD_CLEAR clears all the se and estat bits of the IO_STATUS register.

3.3 Registers

The following Table is a list of Dino's registers, this includes registers in the RS232 and PS2 sub modules.

Sub Module	Offset	Read/ Write	Name	Description
0	0x004	R/W	IAR0	Interrupt Address Register 0
0	0x008	R/W	IODC	IODC Address and Data
0	0x00c	R	IRR0	Interrupt Request Register 0

Sub Module	Offset	Read/ Write	Name	Description
0	0x010	R/W	IAR1	Interrupt Address Register 1
0	0x014	R	IRR1	Interrupt Request Register 1
0	0x018	R/W	IMR	Interrupt Mask Register
0	0x01c	R/W	IPR	Interrupt Pending Register
0	0x020	R/W	TOC_ADDR	Transfer Of Control Address Register, not used.
0	0x024	R/W	ICR	Interrupt Control Register
0	0x028	R	ILR	Interrupt Level Register
0	0x030	W	IO_COMMAND	Command Register
0	0x034	R	IO_STATUS	Status Register
0	0x038	R/W	IO_CONTROL	Control Register
0	0x040	R	IO_GSC_ERR_RESP	GSC Error Address
0	0x044	R	IO_ERR_INFO	Error Information
0	0x048	R	IO_PCI_ERR_RESP	PCI Error Address
0	0x05c	R/W	IO_FBB_EN	Enables Fast back to back for GSC
0	0x060	R/W	IO_ADDR_EN	Address Enable Register
0	0x064	R/W	PCI_CONFIG_ADDR	PCI Configuration Register
0	0x068	R/W	PCI_CONFIG_DATA	PCI Configuration Data Port
0	0x06c	R/W	PCI_IO_DATA	PCI IO DATA Port
0	0x070	R/W	PCI_MEM_DATA	PCI Memory DATA Port
0	0x7B4	R	GSC2X_CONFIG	GSC2X Configuration Register
0	0x800	R/W	GMASK	GSC Arbitration Mask
0	0x804	R/W	PAMR	PCI Arbitration Mask
0	0x808	R/W	PAPR	PCI Arbitration Priority
0	0x80c	R/W	DAMODE	PCI Arbitration Mode
0	0x810	R/W	PCICMD	PCI Command Register
0	0x814	R/WC	PCISTS	PCI Status Register
0	0x81c	R/W	MLTIM	Master Latency Timer
0	0x820	R/W	BRDG_FEAT	Bridge Feature Enable
0	0x824	R/W	PCIROR	PCI Read Optimization Register
0	0x828	R/W	PCIWOR	PCI Write Optimization Register
0	0x830	R/W	TLTIM	PCI Target Latency Timer
1	0x008	R/W	IODC	PS2 IODC Address and Data
1	0x800	R	ID	Keyboard ID

Sub Module	Offset	Read/ Write	Name	Description
1	0x800	W	RESET	Keyboard Reset
1	0x804	R	RCVDATA	Keyboard Received data
1	0x804	W	XMTDATA	Keyboard Transmit data
1	0x808	R/W	CONTROL	Keyboard Control Register
1	0x80c	R	STATUS	Keyboard Status Register
1	0x900	R	ID	Mouse ID
1	0x900	W	RESET	Mouse Reset
1	0x904	R	RCVDATA	Mouse Received Data
1	0x904	W	XMTDATA	Mouse Transmit Data
1	0x908	R/W	CONTROL	Mouse Control Register
1	0x90c	R	STATUS	Mouse Status Register
3	0x000	R/W	RESET	RS232 Reset. Resets on write OR read.
3	0x004	W	TEST	RS232 Test Register
3	0x008	R/W	IODC	RS232 IODC Address and Data
3	0x060	R/W	DITHER	Controls RS232 clock frequency generator
3	0x800	R	RBR	Receiver Buffer Register (DLAB=0)
3	0x800	W	THR	Transmitter Holding Register (DLAB=0)
3	0x800	R/W	DLL	Divisor Latch Register LSB (DLAB=1)
3	0x801	R/W	IER	Interrupt Enable Register (DLAB=0)
3	0x801	R/W	DML	Divisor Latch Register MSB (DLAB=1)
3	0x802	R	IIR	Interrupt Ident Register
3	0x802	W	FCR	Fifo Control Register
3	0x803	R/W	LCR	Line Control Register
3	0x804	R/W	MCR	Modem Control Register
3	0x805	R	LSR	Line Status Register
3	0x806	R/W	MSR	Modem Status Register
3	0x807	R/W	SCR	Scratch Register

Table 2. Dino’s Register Listing

4 BRIDGE REGISTERS

4.1 Bridge Hard Physical Address (HPA) Registers

The HPA for a bus converter is split into a Supervisory Register Set (SRS), an Auxiliary Register Set (ARS), a Bus Specific Register Set (BSRS), and an HVERSION–Dependent Register Set (HVRS). Please see the *Precision I/O Architecture Reference Specification* for more details.

Dino implements the required SRS and ARS registers. Dino also implements several optional SRS registers (for the Interrupt Controller), a couple optional ARS registers (for address mapping), several BSRS registers (for EGSC and GSC+ control), and several HVRS registers (control and status specific to the Dino chip). The subsections below detail each of the SRS, ARS, BSRS, and HVRS register set implementations for Dino.

4.1.1 HPA Supervisory Register Set (SRS Registers)

Dino's SRS register addresses are given in Table NO TAG. The following subsections describe each of the SRS registers.

Register	Symbol	HPA Register Offset	R/W	Description
Interrupt Address Reg. 0	IAR0	0x004	R/W	Interrupt address to be used when a device mapped to int0 by the ICR issues an interrupt.
IODC Address Register	IODC_ADDR	0x008	W	Allows for selection between multiple IODC DATA words.
IODC Data Register 0	IODC_DATA_0	0x008	R	The first IODC_DATA word.
IODC Data Register 1	IODC_DATA_1	0x008	R	The second IODC_DATA word
Interrupt Request Reg. 0	IRR0	0x00C	R	The IRRO contains the status of all requesting interrupts that are mapped to int0.
Interrupt Address Reg. 1	IAR1	0x010	R/W	Interrupt address to be used when a device mapped to int0 by the ICR issues an interrupt.
Interrupt Request Reg. 1	IRR1	0x014	R	The IRR1 contains the status of all requesting interrupts that are mapped to int1.

Register	Symbol	HPA Register Offset	R/W	Description
Interrupt Mask Register	IMR	0x018	R/W	The IMR is used to mask pending interrupts.
Interrupt Pending Reg.	IPR	0x01C	R/W	The IPR is used to latch incoming interrupts and indicate them as pending.
TOC Address Register	TOC_ADDR	0x020	R/W	The address used to issue a CMD_RESET when TOC is asserted. Not used.
Interrupt Control Reg.	ICR	0x024	R/W	The ICR is used to control whether an interrupt source is mapped to int0 or int1.
Interrupt Level Register	ILR	0x028	R	The ILR register indicates the state of the interrupt lines.
Command Register	IO_COMMAND	0x030	W	The IO_COMMAND register allows Dino to respond to directed resets on GSC.
Status Register	IO_STATUS	0x034	R	Dino reports its overall status to system software via the IO_STATUS register.
Control Register	IO_CONTROL	0x038	R/W	The forwarding of transactions by a the Dino bridge is regulated by writing to the IO_CONTROL register.

Table 3. HPA SRS Registers

IODC_ADDR (IODC Address Register)

This is a one bit register that points to either IODC_DATA_0 or IODC_DATA_1. If all zeros are written to IODC_ADDR then IODC_DATA_0 will be read. If 32'h0000_0004 is written to IODC_ADDR then IODC_DATA_1 will be read.

IODC_DATA_0 (IODC Data Register 0)

	0 … 15	16 … 23	24 … 31
0x008	**IODC_HVERSION**	**IODC_SPA**	**IODC_TYPE**
	31 … 16	15 … 8	7 … 0

For Dino 2.0 (1FC3–0001) IODC_DATA_0 will read 0x6800_004d

For Dino 2.1 (1FC3–0002) IODC_DATA_0 will read 0x6801_004d

For Dino 3.0 (1FC3–0003) Bridge Mode IODC_DATA_0 will read 0x6802_004d

For Dino 3.0 (1FC3–0003) Card Mode IODC_DATA_0 will read 0x0040_0044

For Dino 3.1 (1FC3–0004) Bridge Mode IODC_DATA_0 will read 0x6803_004d

For Dino 3.1 (1FC3–0004) Card Mode IODC_DATA_0 will read 0x0040_0044

- IODC_HVERSION: Hardware version number. The HVERSION field is 16 bits in length,IODC_DATA_0[31:16]. The HVERSION has two internal fields, model and revision. The model field, IODC_DATA_0[31:20] will be set to 0x680 for Dino. This signifies that Dino is a GSC++ bus bridge. **The the revision field,IODC_DATA_0[19:16], is set to 0x0 for Dino 2.0, set to 0x1 for Dino 2.1, set to 0x2 for Dino 3.0, and set to 0x3 for Dino 3.1.**

- IODC_SPA: Soft physical address capabilities. IODC_SPA,IODC_DATA_0[15:8] is set to 0x00 for Dino, since Dino does not require SPA space.
- IODC_TYPE: Identify module type. IODC_TYPE is made up of four internal fields,*mr* bit,*wd* bit, R bit, and type field. The *mr* "more" bit, IODC_DATA_0[7], will be set to 0 for Dino. Dino will have no more than 8 bytes of IODC. The *wd* "word" bit,IODC_DATA_0[6], will be set to 1 for Dino. Dino will provide a full word of data for IODC. The R bit, IODC_DATA_0[5], is reserved and will be set to zero in Dino. The type field,IODC_DATA_0[4:0], will be set to 0xD. This indicates that Dino is a bus bridge.

IODC_DATA_1 (IODC Data Register 1)

	0 ... 31
0x008	**IODC_SVERSION**
	31 ... 0

For Dino in Bridge Mode IODC_DATA_1 will read 0x0000_0a00

For Dino in Card Mode IODC_DATA_1 will read 0x0000_9d80

NOTE: IODC_SVERION is incorrect for DINO 1.0 parts. The correct value for IODC_SVERION is 0x0000_0A00. This will be corrected in the second tape release.

- IODC_SVERSION: SVERSION is made up of three internal fields, revision field,model field and opt field. The opt field,IODC_DATA_1[7:0], will have a value of 0x00,software coherence, for Dino. The model field,IODC_DATA_1[27:8], will have a value of 0x0000a. This indicates that the bus bridge is to PCI. The revision field,IODC_DATA_1[31:28], will have an initial value of 0x0 and will be incremented if there are additional revisions of the Dino chip that effect software.

IAR0 (Interrupt Address Register 0)

This register contains the address of the IO_EIR and the group code that will be used when a device mapped to int0 by the ICR issues an interrupt. Please see the Interrupts chapter of this ERS for more details.

IRR0 (Interrupt Request Register 0)

The IRR0 contains the status of all requesting interrupts that are mapped to int0. Please see the Interrupts chapter of this ERS for more details.

IAR1 (Interrupt Address Register 1)

This register contains the address of the IO_EIR and the group code that will be used when a device mapped to int1 by the ICR issues an interrupt. Please see the Interrupts chapter of this ERS for more details.

IRR1 (Interrupt Request Register 1)

The IRR0 contains the status of all requesting interrupts that are mapped to int1. Please see the Interrupts chapter of this ERS for more details.

IMR (Interrupt Mask Register)

The IMR is used to mask pending interrupts. A 1 in an IMR bit enables the corresponding pending interrupt to create an interrupt request. The IMR is cleared at reset. Please see the Interrupts chapter of this ERS for more details.

IPR (Interrupt Pending Register)

The IPR is used to latch incoming interrupts and indicate them as pending. The assertion of an internal interrupt signal causes the corresponding IPR bit to be set to 1. Please see the Interrupts chapter of this ERS for more details.

TOC_ADDR (TOC Address Register)

The TOC_ADDR register contains the address of the processor's IO_COMMAND register. A CMD_RESET will be sent to to this address when TOC is asserted. Note that since there is no TOC pin this register does not do anything.

	0 3	4 13	14	15 18	19	20 25	26 29	30 31
0x020	1111	flex = 1111111110	**1**	client_id	0	regset = 000000	reg=1100	00
	31 28	27 18	17	16 13	12	11 6	5 2	1 0

- client id: This four bit value corresponds to the fixed field of the monarch processor and is concatenated between the hardwired flex field and register information to form the address of the monarch processor's IO_COMMAND register. On power on the client id will be set to zero, causing a register value of 0xFFFA0030. This register is not effected by CMD_RESET or CMD_CLEAR.

ICR (Interrupt Control Register)

The ICR is used to control whether an interrupt source is mapped to int0 or int1. Please see the Interrupts chapter of this ERS for more details.

ILR (Interrupt Level Register)

The ILR register indicates the state of all interrupt lines. Please see the Interrupts chapter of this ERS for more details.

IO_COMMAND (IO Command Register)

	0 15	16 23	24	25 31
0x030	Undefined	**CMD_DEP**	0	**CMD**
	31 16	15 8	7	6 0

- CMD: Dino recognizes two commands: CMD_RESET (CMD=5) and CMD_CLEAR (CMD=3). All other fields are ignored. Dino will set its IO_CONTROL register to OFF mode upon receipt of a CMD_RESET. CMD_RESET will reset the state of the bridge's GSC interface and be forwarded to the PCI bus. Dino transactions within 16 states of CMD_RESET are not allowed. CMD_CLEAR is used to clear the se and estat bits of the IO_STATUS register.
- For more details about CMD_RESET forwarding to PCI see the "PCI Command Register" description.

IO_STATUS (Status Register)

	0–15	16–31
0x34	Undefined	**IO_STATUS**
	31–16	15–0

PA Bit	Symbol	IO_STATUS Description	GSC Bit
0–15	Undefined		31–16
16–21	estat	Contains an error status code corresponding to the most severe error currently logged. Dino will only have 3 possible vectors stored in this field: 1) 6’b000000 –clear, 2) 6’b000001 – less than fatal, 3) 6’b000011 –fatal. See the “Dino Errors and Abnormal Conditions” section.	15–10
22	se	This bit is always zero unless Dino is in “Less Than Fatal Mode”.	9
23	he	Hardwired to 0 because Dino does not distinguish this type of ERROR. See the “Dino Errors and Abnormal Conditions” section.	8
24	fe	When set, indicates that a fatal error was detected by Dino.	7
25	ry	When set, indicates that Dino is ready to accept commands. Probably will be hardwired to 1.	6
26–27	Reserved		5–4
28	lp	Set if this status register applies to lower bridge port. This bit is hardwired to 0 in Dino (this is an upper bridge port status register).	3
29–31	pwrstat	Remote bus power status. Hardwired to 000 in Dino.	2–0

Table 4. Status Register Bit Definition

Please see the “Dino Errors and Abnormal Conditions” section in this document for more details on the error related bits.

IO_CONTROL (Control Register)

	0–15	16–21	22	23–24	25–31
0x038	Undefined	Reserved	Und	**mode**	Reserved
	31–16	15–10	9	8–7	6–0

- mode: Dino only implements two control modes. Mode 01 (INCLUDE) indicates that all addresses enabled in the register IO_ADDR_EN should be forwarded. Any other value of Mode will not enable Dino to forward the transaction to PCI.

4.1.2 HPA Auxiliary Register Set (ARS Registers)

Dino’s ARS register addresses are given in Table NO TAG. The following subsections describe each of the SRS registers.

Register	Symbol	HPA Register Offset	R/W	Description
IO Error Responder	IO_GSC_ERR_RESP	0x040	R	Logs the address of the GSC responder in an erroneous GSC operation.
IO Error Info	IO_ERR_INFO	0x044	R	Provides extended error logging information about an erroneous GSC or PCI operation.
IO Error Requestor	IO_PCI_ERR_RESP	0x048	R	Logs the requestor (could be on GSC or originate from PCI) of the erroneous GSC operation.
IO Fast Back–to–Back Enable	I0_FBB_EN	0x05c	R/W	The IO_FBB_EN register enables fast back–to–back programmed IO writes on the GSC bus.
IO Address Enable	IO_ADDR_EN	0x060	R/W	The IO_ADDR_EN register defines which 8MB windows to decode within the 256MB of PA IO space.
PCI Configuration Address Port	PCI_CONFIG_ADDR	0x064	R/W	The address port for PCI I/O and Configuration space.
PCI Configuration Data Port	PCI_CONFIG_DATA	0x068	R/W	The data port for PCI Configuration space.
PCI IO Data Port	PCI_IO_DATA	0x06c	R/W	The data port for PCI I/O space.
PCI Memory Data Port	PCI_MEM_DATA	0x070	R/W	The data port for PCI Memory space.

Table 5. HPA ARS Registers

IO_GSC_ERR_RESP (GSC Error Responder Address)

	0–31
0x040	**IO_GSC_ERR_RESP**
	31–0

- IO_GSC_ERR_RESP: This register logs the 32–bit address sent from Dino to a GSC target when an error is signalled. **This value is only valid when the vag bit in the IO_ERR_INFO register is set.**

IO_ERR_INFO (Error Logging Information)

	0	28	29	30	31
0x044	Undefined	**epci**	**egsc**	**vap**	**vag**
	31	3	2	1	0

- IO_ERR_INFO: When *vag* is 1, the contents of the IO_GSC_ERR_RESP register is valid. When *vap* is 1, the contents of the IO_PCI_ERR_RESP register is valid. When *egsc* is 1, an error happened on GSC while Dino was **not** the master. When *epci* is 1, an error happened on PCI when Dino was **not** the master.

IO_PCI_ERR_RESP (PCI Error Responder Address)

Offset	0–31
0x048	**IO_PCI_ERR_RESP**
	31–0

- IO_PCI_ERR_RESP: This register logs the 32–bit address sent from Dino to a PCI target when an error is signalled. This **value is only valid when the vap bit in the IO_ERR_INFO register is set.**

IO_FBB_EN (IO Fast back to back Enable)

Offset	0–30	31
0x05C	**0**	**IO_FBB_EN**
	31–1	0

- IO_FBB_EN: Writing 32'h0000_0001 to this register will enable fast back to back transactions on the GSC bus. IO_FBB_EN is disabled at power on.

IO_ADDR_EN (IO Address Enable)

Offset	0	1–30	31
0x060	0	**IO_ADDR_EN**	0
	31	30–1	0

- IO_ADDR_EN: The IO Address Enable register is used to enable address chunks to be forwarded from the GSC bus to the PCI bus. There are 32 x 8MB chunks of address space in PA I/O space. Each bit in IO_ADDR_EN can be set to enable one 8MB chunk of PA I/O space. The MSB is hardwired to zero because the highest 8MB chunk is used by the CPU and cannot be forwarded to PCI. The LSB is hardwired to zero because the lowest 8MB chunk of PA I/O space is used by the boot ROM. See the PCI Chapter of this ERS for more details.

PCI_CONFIG_ADDR (PCI Configuration Address Port)

Offset	0–31
0x064	**PCI_CONFIG_ADDR**
	31–0

- PCI_CONFIG_ADDR: PCI_CONFIG_ADDR is the address port for PCI I/O and Configuration space. Please see the *PCI* chapter of this ERS for more details.

PCI_CONFIG_DATA (PCI Configuration Data Port)

Offset	0–31
0x068	**PCI_CONFIG_DATA**
	31–0

- PCI_CONFIG_DATA: PCI_CONFIG_DATA is the data port for PCI Configuration space. Please see the *PCI* chapter of this ERS for more details.

PCI_IO_DATA (PCI IO Data Port)

Offset	0–31
0x06c	**PCI_IO_DATA**
	31–0

- PCI_IO_DATA: PCI_IO_DATA is the data port for PCI I/O space. Please see the *PCI* chapter of this ERS for more details.

PCI_MEM_DATA (PCI IO Data Port)

	0 … 31
0x070	**PCI_MEM_DATA**
	31 … 0

- PCI_MEM_DATA: PCI_MEM_DATA is the data port for PCI memory space. Please see the *PCI* chapter of this ERS for more details.

Please see the PCI chapter of this ERS for more details on mapping address space between GSC and PCI.

4.1.3 HPA Bus(GSC) Specific–Dependent Reg. Set (BSRS Registers)

Register	Symbol	HPA Register Offset	R/W	Description
GSC2X Configuration Register	GSC2X_CONFIG	0x07B4	R	Used to tell which GSC2X features are enabled.

Reading this register on Dino always returns 0x0000_0001 to indicate that the type “F” transaction is implemented. No other GSC2SX features are implemented.

4.1.4 HPA HVERSION(Dino)–Dependent Reg. Set (HVRS Registers)

Dino’s HVRS register addresses are given in Table NO TAG. These registers are specific to Dino, and include the registers required for control/status of Dino’s downstream (PCI) port. The following subsections describe each of the HVRS registers.

Register	Symbol	HPA Register Offset	R/W	Description
GSC Arbitration Mask	GMASK	0x800	R/W	GSC Arbitration Mask register.
PCI Arbitration Mask	PAMR	0x804	R/W	PCI Arbitration Mask register.
PCI Arbitration Priority	PAPR	0x808	R/W	Controls PCI arbiter priority scheme.
Dino Arbitration Mode	DAMODE	0x80C	R/W	Controls the mode of the PCI arbitration lines.
PCI Command Register	PCICMD	0x810	R/W	Provides coarse control of Dino’s PCI interface
PCI Status Register	PCISTS	0x814	R/WC	Records status info for PCI bus related events. Writes to this register clears it.
Undefined		0x818		
PCI Mast. Latency Timer	MLTIM	0x81C	R/W	Specifies, in units of PCI clocks, the values of the latency timer for the Dino PCI master.

Register	Symbol	HPA Register Offset	R/W	Description
Bridge Feature Enable	BRDG_FEAT	0x820	R/W	Enable/disable bridge features.
PCI Read Opt. Regist.	PCIROR	0x824	R/W	Defines the Dino read–hints.
PCI Write Opt. Regist.	PCIWOR	0x828	R/W	Defines the Dino write–hints.
PCI Target Latency Timer	TLTIM	0x830	R/W	Number of cycles for target subsequent latency timer.

Table 6. HPA HVRS Registers

GMASK (GSC Request Mask)

	0 …	31
0x800	Undefined	**GMASK**
	31 …	0

- GMASK: This register provides a means of masking the request line of the GSC block. Please see the Arbitration chapter of this ERS for more details.

PAMR (PCI Arbitration Mask)

	0 … 25	26 … 31
0x804	Undefined	**PAMR**
	31 … 7	6 … 0

- PAMR: This register provides a means of masking the request line of PCI devices using the Dino PCI arbiter. Please see the Arbitration chapter of this ERS for more details.

PAPR (PCI Arbitration Priority)

	0 … 25	26 … 31
0x808	Undefined	**PAPR**
	31 … 7	6 … 0

- PAPR: This register provides a means of controlling the Dino PCI arbiter priority scheme. Please see the Arbitration chapter of this ERS for more details.

DAMODE (Dino Arbitration Mode)

	0 … 24	25 … 31
0x80C	Undefined	**DAMODE**
	31 … 8	7 … 0

- DAMODE: DAMODE is a two bit register used to set or clear Performance mode arbitration for the PCI arbiter device PCID. Please see the Arbitration chapter of this ERS for more details.

PCICMD (PCI Command Register)

	0 – 24	25 – 31
0x810	Undefined	**PCICMD**
	31 – 7	6 – 0

PA Bit	Symbol	PCICMD Description	GSC Bit
0–24	Undefined		31–7
25	SEC_RESET	This bit is cleared at power on and after a CMD_RESET, causing the PCI RST# signal to be asserted (low). Writing a 1 to this bit deasserts RST# on PCI, bringing the PCI bus and the bridge's PCI interface out of reset.	6
26	FBBE	Fast back–to–back Enable. Controls ability of Dino to generate fast back to back transactions to different devices on PCI.	5
27	MWI	Memory write and invalidate enable. Dino will not generate MWI transactions, so this bit will be hardwired to 0. (Dino does respond to MWI commands as a slave.)	4
28	SERR_EN	Controls enable of Dino's SERR# driver.	3
29	PER	Parity error response. When PER is set, Dino will check for address and data parity errors. When PER is clear, Dino will not check for address or data parity errors. The PER bit does not affect how Dino reacts to PERR# being asserted.	2
30	LOW_DEC	This bit must be set to a 1 to enable upstream transactions in address range $0000_0000 – $EFFF_FFFF to pass from PCI to GSC.	1
31	NEG_DEC	The NEG_DEC bit must be set to a 1 to enable Dino to perform negative–decode of upstream PCI memory transactions. Negative decode is used in the address range from $F000_0000 – $FFFF_FFFF. Please see the PCI chapter of this ERS for more details.	0

Table 7. PCI Command Register Bit Definition

- PCICMD: Note, some of the bits may not be implemented.

Note: IF the NEG_DEC bit is set, caution should be taken when addressing PA Space above F000_0000. If a transaction starts in one 8Mb chunk and finishes in the next 8Mb chunk it is assumed that the second chunk is *not* enabled in the IO_ADDR_EN register.

PCISTS (PCI Status Register)

	0 … 22	23 … 31
0x814	Undefined	**PCISTS**
	31 … 9	8 … 0

PA Bit	Symbol	PCISTS Description	GSC Bit
0–22	Undefined		31–9
23	FBBC	Fast back–to–back capable. Returns a 1 to indicate Dino supports back–to–back accesses.	8
24	DPD	Data parity detected. Set when these three conditions are all true: Dino is the bus master, Dino signaled or received PERR#, and the PER bit in PCICMD is set.	7
25–26	DEVSEL	DEVSEL timing. Returns 2’b01 for Dino.	6–5
27	STA	Signaled target abort. Set whenever Dino target target–aborts.	4
28	RTA	Received target abort. Set whenever Dino master receives target–abort.	3
29	RMA	Received master abort. Set whenever Dino master master–aborts.	2
30	SSE	Signaled system error. Set whenever Dino asserts SERR#.	1
31	DPE	Detected parity error. Set whenever Dino detects a PCI parity error and is the data sink, regardless of the PER bit in the PCICMD register.	0

Table 8. PCI Status Register Bit Definition

- PCISTS: Several of these bits may not be needed for Dino.

NOTE: All bits that are not hardwired will be cleared by a CMD_CLEAR in the IO_COMMAND register.

The RMA bit (Received Master Abort) is handled differently for configuration space reads and writes.

MLTIM (Master Latency Timer)

	0 … 23	24 … 31
0x81C	Undefined	**MLTIM**
	31 … 8	7 … 0

- MLTIM: This register contains the programmable value of the Master Latency Timer for use when Dino is a master on the PCI Bus. The granularity of the timer is 8 PCI clocks. Thus, the 3 LSBs are hardwired to zero.

BRDG_FEAT (Bridge Feature Enable)

	0 ... 1	2 ... 31
0x820		**BRDG_FEAT**
	31 ... 30	29 ... 0

- BRDG_FEAT: This register is used to enable/disable bridge features. Please see the “Bridge Operation and Configuration” section of this document for more information.

PCIROR (PCI Read Optimization Register)

	0 ... 7	8 ... 31
0x824		**PCIROR**
	31 ... 24	23 ... 0

- PCIROR: This register is used for PCI read performance optimization. Please see the “Bridge Operation and Configuration” section of this document for more information.

PCIWOR (PCI Write Optimization Register)

	0 ... 7	8 ... 31
0x828		**PCIWOR**
	31 ... 24	23 ... 0

- PCIWOR: This register is used for PCI write performance optimization. Please see the “Bridge Operation and Configuration” section of this document for more information.

TLTIM (PCI Target Latency Timer)

	0 ... 23	24	25 ... 31
0x830		**EN**	**TLTIM**
	31 ... 8	7	6 ... 0

- EN: If the EN bit is set the PCI block uses the value of TLTIM as the count value for target subsequent latency. If the EN is NOT set the PCI block uses the value 8 as the count for target subsequent latency.
- TLTIM: This is the seven bit count value for the target subsequent latency. The lower two bits of this field are hardwire to zero.

5 BRIDGE OPERATION AND CONFIGURATION

5.1 Introduction

This section discusses how HPA registers are accessed, how DMA and DIO transactions work, and the effect of configuration register settings on bridge operation.

5.2 Overview

Before Dino will do anything its flex register needs to be programmed. Once the flex register is initialized Dino will respond to programmed I/O transactions to its GSC resident HPA registers. Dino's HPA registers completely define GSC to PCI bridge behavior.

Dino maps a fixed transaction length bus (GSC) to a variable transaction length bus (PCI). In order to do this efficiently transactions bound for GSC need to be cast into the optimum size. Each device Dino arbitrates for has a unique set of hint vectors, so to maximize DMA performance, the system designer needs to understand the type of bus traffic typically generated by these devices. Once the nature and arbitration position of these devices is understood a good set of hint vectors can be derived. Hint vectors are part of Dino's HPA register set. DIO transactions don't require hint vectors because they come from GSC and start out as fixed length transactions.

5.3 What is flex ?

One natural and often asked question is: What is flex? Flex is essentially a 10–bit bus identification number that is broadcast to all devices on a given bus segment. Upon receiving the broadcast flex transaction all the devices (on the segment receiving the broadcast) are programmed with the same bus identification number and enabled to respond to DIO (direct I/O transaction.) The 10–bit bus identification number, along with a 4–bit slot address, and a 2–bit submodule identification fully decodes 4K bytes of HPA space for an HPPA device. Dino compares all GSC addresses it receives to see if they match to the fully decoded HPA address as defined above: if so, then the register is accessed.

It is useful to remember that bus–ids are only used for accessing Dino's HPA registers and do not determine which addresses are forwarded to PCI.

5.4 A simplified Dino start–up example

In order to better explain how to use Dino's HPA registers a basic start–up example, based upon a real Verilog simulation test case, follows:

```
m01 ghost_em write1 0xfffc0020 0xff000001      /* Set Flex                  */
m02 ghost_em write1 0xff000038 0x00000080      /* Set IO_CONTROL            */
m03 ghost_em write1 0xff000804 0x00000000      /* Set PAMR                  */
m04 ghost_em write1 0xff000808 0x00000000      /* Set PAPR                  */
m05 ghost_em write1 0xff00005c 0x00000001      /* Set IO_FBB_EN             */
m06 ghost_em write1 0xff000060 0x0000fffe      /* Set IO_ADDR_EN            */
m07 ghost_em write1 0xff00080c 0x00000000      /* Set DAMODE                */
m08 ghost_em write1 0xff000824 0x00000000      /* Set PCIROR read hint=1    */
m09 ghost_em write1 0xff000828 0x00000000      /* Set PCIWOR write hint=1   */
m10 ghost_em write1 0xff000810 0x0000006f      /* Set PCICMD reset PCI      */
```

5.5 GSC Interface Optimization Registers

In order to improve bridge performance Dino provides several registers to fine–tune transaction mapping between PCI and GSC. These registers also give means of removing complicated operating modes, thus helping system turn–on efforts. Also, the feature enable register provides a means of enabling and disabling some of the more general behaviors of the bridge.

Dino's miscellaneous register page contains two registers that provide hints to the bridge on the behalf of the bus masters it arbitrates for. Specifically Dino arbitrates for 6 general PCI devices, all of which can have unique hint vectors.

Register	Symbol	Address Offset	R/W	Description
Read Optimization Register	ROR	0x824	R/W	The contents of ROR determine the read prefetch length under a variety of conditions. This register provides the flexibility to individually optimize read prefetch length PCI bus masters.
Write Optimization register	WOR	0x828	R/W	The contents of the WOR determine the number of words PCI write transactions are broken into for GSC. This register provides the flexibility to individually optimize writes onto GSC for each bus master.
Feature enable register	BRDG_FEAT	0x820	R/W	This register provides bits that enable various GSC/PCI bridge performance enhancement features.

Table 9. Registers that control GSC to PCI behavior

5.5.1 Read Optimization Register

	31 24	23 0
0x824		PCIROR

Figure 3. Read Optimization Register

Bit	Symbol	Value after Reset	Description
3:0	RORA	4’b0000	Encoded prefetch algorithm for PCI master reads from PCI device A.
7:4	RORB	4’b0000	Encoded prefetch algorithm for PCI master reads from PCI device B.
11:8	RORC	4’b0000	Encoded prefetch algorithm for PCI master reads from PCI device C.
15:12	RORD	4’b0000	Encoded prefetch algorithm for PCI master reads from PCI device D.
19:16	RORE	4’b0000	Encoded prefetch algorithm for PCI master reads from PCI device E.
23:20	RORF	4’b0000	Encoded prefetch algorithm for PCI master reads from PCI device F.
31:24	SPARE	8’b0000_0000	Unused bits.

Note: All devices can be configured to connect to an external arbiter.

Table 10. Read Optimization Register Definition

Prefetch Algorithm Encodings for RORA, RORB, RORC, RORD, RORE, and RORF		
Encoded Field	**Prefetch Algorithm**	**XQL asserted**
4’b0000	1 word prefetch	No
4’b0001	2 word prefetch	No
4’b0010	4 word prefetch	No
4’b0011	8 word prefetch	No
4’b0100	16 word prefetch	Yes, if enabled
4’b0101	24 word prefetch	Yes, if enabled
4’b1011	Keep FIFO 1/2 full	Yes, if enabled
4’b1111	Keep FIFO full	Yes, if enabled

Table 11. Read Optimization Field Encodings

When a PCI master starts a read transaction directed to GSC, the length of the transaction is unknown to the bridge. The GSC side of the bridge needs to pick a transaction size, so it must anticipate the actual size of the variable length PCI transaction. If it picks a prefetch size that is too small, too many small GSC transactions will be used. If it picks an unnecessarily large prefetch size, it will generate a large GSC transaction that transfers more data than necessary. It is best to understand the needs of the mastering device and choose the prefetch constant accordingly. To add flexibility, prefetch algorithms are also selectable.

Note: The PCI memory read line transaction , if properly enabled, forces the ROR field for the transaction to effectively be mapped to 4'b0101, meaning that 24 words will be prefetched regardless of the actual value in the ROR for that device.

Note: The PCI memory read multiple transaction , if properly enabled, forces the ROR field for the transaction to effectively be mapped to 4'b1111, meaning that bridge will try to keep the read prefetch fifo as full as possible regardless of the actual value in the ROR for that device.

5.5.2 Write Optimization Register

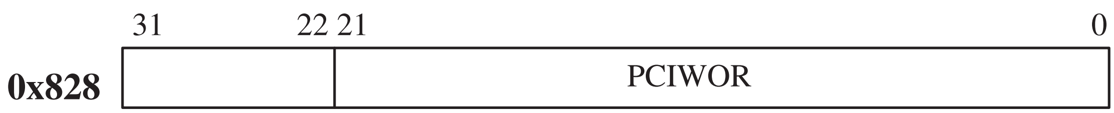


Figure 4. Write Optimization Register

Bit	Symbol	Value after Reset	Description
1:0	WPCIA	2'b00	Encoded length constant for PCI memory write from Dino's PCI device A.
3:2	SPARE	2'b00	Unused bits always read as zero
5:4	WPCIB	2'b00	Encoded length constant for PCI memory write from Dino's PCI device B.
7:6	SPARE	2'b00	Unused bits always read as zero
9:8	WPCIC	2'b00	Encoded length constant for PCI memory write from Dino's PCI device C.
11:10	SPARE	2'b00	Unused bits always read zero
13:12	WPCID	2'b00	Encoded length constant for PCI memory write from Dino's PCI device D. **This is the device that can be configured to connect to an external arbiter.**
15:14	SPARE	2'b00	Unused bits always read zero
17:16	WPCIE	2'b00	Encoded length constant for PCI memory write from Dino's PCI device E.
19:18	SPARE	2'b00	Unused bits always read zero

Bit	Symbol	Value after Reset	Description
21:20	WPCIF	2’b00	Encoded length constant for PCI memory write from Dino’s PCI device F.
31:22	SPARE	10’b00_0000_0000	Unused bits always read zero

Table 12. Write Optimization Register Definition

Encoded field	Behavior
2’b00	The GSC interface uses only single word writes when transforming a variable length PCI transaction.
2’b01	The GSC interface uses double and single word transactions to transform a variable length PCI transaction.
2’b10	The GSC interface uses only quad word and single word writes to transform a variable length PCI transaction.
2’b11	The GSC interface uses only eight word and single word writes to transform a variable length PCI transaction.

Table 13. Write Optimization Length Encoding

A variable length write from a PCI master needs to be properly decomposed into fixed length GSC write transactions. Bridge efficiency can be improved by tuning the variables in the write optimization register for each potential master device.

Note: If properly enabled, the PCI memory write and invalidate transaction type will map to a GSC write 8 regardless of the WOR settings. See the “Feature Enable Register” description.

5.5.3 Feature Enable Register

Dino contains a number of performance enhancing features that can be enabled and disabled by doing direct I/O transactions. These features are disabled at power up and must be enabled via DIO writes to this register.

	31 30	29 … 0
0x820		BRDG_FEAT

Bit	Symbol	Value after Reset	Description
29	PUSPLIT	1’b0	Prefetch Under Split: When PUSPLIT = 1, Dino will allow up to a 24 word DMA read prefetch under a SPLIT DIO transaction. When PUSPLIT = 0, Dino will only allow up to an 8 word DMA read prefetch under a SPLIT DIO transaction. If DMA reads are being starved by lots of DIO write transactions, setting PUSPLIT = 1 may help balance performance.
28	PARB_REL_GNT_MD	1’b0	PCI ARB Mode: When PARB_REL_GNT_MD = 0, Dino will keep the current PCI BUS Master GNT line asserted if that Device deasserts it REQ and their are no other Device REQ’s asserted. If PARB_REL_GNT_MD = 1, Dino will deassert a Device GNT line if that Device deasserts it’s REQ.
27	DPCIBACK-OFF	1’b0	Disable PCI Backoff: When DPCIBACKOFF = 1, Dino will not back DMA traffic off of the PCI bus during GSC splits or GSC pended DIO read transactions. Doing this can cause long splits and timeouts on pended DIO reads. DPCIBACKOFF should always be 0.
26	DPCIHIT	1’b0	Disable PCI Hit: When DPCIHIT = 1, Dino will ignore GSC transactions to PA I/O addresses corresponding to address ranges enabled in the IO_ADDR_EN register. DPCIHIT should only be set when Dino is used on GSC add–in–cards. See the Dino–on–a–Card Mode Chapter for more information.
25	DABORT	1’b0	Discard Aborts: When DABORT = 1, PCI master and target aborts will cause writes to be discarded and reads to return with data of –1. When DABORT = 0, PCI master and target aborts will cause the data to be trapped in Dino and retried indefinitely. **This feature is not officially supported. It may or may not work as expected.**
24	reserved		
23	DPERR_CHK	1’b0	When set this disables data parity checking on the GSC bus.
22–16	WATCH_DOG	7’b0	This is the timer referred to in the GOOD_DOG and AUTO_DOG descriptions below. The three lowest bits of the field are always zero.
15	reserved		
14	GOOD_DOG	1’b0	Set this bit when connecting Dino to a system that does not take away GRANT. When another device wants the bus. Dino will get off the bus after GRANTL goes away **OR** the counter expires.

Bit	Symbol	Value after Reset	Description
13	AUTO_DOG	1’b0	Set the GOOD_DOG and AUTO_DOG when Dino is connected to a system that gives a pulsed GRANTL. Dino will get off the bus only when the counter expires. This mode will not turn on unless the GOO_DOG bit is turned on also.
12	DCOMP	1’b0	Delayed Completion Enable. DCOMP = 1 will enable the PCI side to perform delayed completion for upstream reads whose initial latency is more than 2*TLTIM PCI clocks. (Where TLTIM is the value in the TLTIM register.) Setting this bit to 1 is necessary for the bridge to be fully PCI Rev 2.1 compliant. Performance for DMA reads will be lower with DCOMP = 1. **This feature is not officially supported. It may or may not work as expected.**
11	PMWI	1’b1	PCI Memory Write and Invalidate Command Enable. If cleared to 0, the bridge as a PCI target will treat PCI command 4’b1111 as exactly equivalent to normal Memory Writes—no special write optimizations.
10	PMRM	1’b1	PCI Memory Read Multiple Enable. If cleared to 0, the bridge as a PCI target will treat PCI command 4’b1100 as exactly equivalent to normal Memory Reads—no special read optimizations.
9	PMRL	1’b1	PCI Memory Read Line Enable. If cleared to 0, the bridge as a PCI target will treat PCI command 4’b1110 as exactly equivalent to normal Memory Reads—no special read optimizations.
8	COAL_ON	1’b0	Enable coalescing of DIO writes.
5–7	reserved		
4	LTFM	1’b0	Enable “Less Than Fatal Mode”. This may help some systems go down more gracefully, but increases the chances of propagating bad data.
3	reserved		
2	UXQL	1’b0	Use XQL. When set, this bit will map memory PCI read line and memory read multiple commands to GSC transactions that assert XQL. It may (haven’t decided yet) also turn on XQL for some read prefetch algorithms.

Bit	Symbol	Value after Reset	Description
1	ESGSC+	1’b1	Enable Slave GSC+ features. If this bit is set the bridge will respond to slave transactions using the GSC+ features if the gsc_L pin = 1’b1.
0	EMGSC+	1’b1	Enable Master GSC+ features. If this bit is set the bridge will master transactions using the GSC+ features if the gsc_L pin = 1’b1.

Table 14. Bridge Feature Enable Register Description

6 GSC INTERFACE

6.1 Introduction

Dino connects to the system CPU using HP's proprietary GSC (General System Connection). System firmware and software will not be directly affected by the operation of this interface. This section describes the capabilities and physical characteristics Dino's connection to GSC.

6.2 Overview

The GSC interface supports both slave and master transactions. These transactions appear on or are mastered from Dino's external PCI bus. When a device on the PCI bus acquires PCI and masters a transaction, it provides an address and transfers a variable number of data words; these arbitrary length transactions are converted to a number of properly sized fixed length GSC transactions. CPU mastered transactions are of fixed length and are presented to PCI as transactions of the same length, except when coalesced. The GSC and PCI devices are assumed to operate from different asynchronous clocks.

6.3 Features

- Converts variable length PCI transactions into fixed length GSC transactions
- Supports the new GSC type ”1111” transaction
- Configurable conversion thresholds
- Coalesces DIO writes
- Supports fast back to back single word writes (under some conditions)
- Operates between asynchronous clock domains
- Supports simultaneous bus mastership on PCI and GSC
- GSC+ compatible
- High speed GSC 66MHz operation

6.4 Dino's GSC transaction type support

Type[0:3]		Transaction Type	Functional-ity level	Dino Supported
binary	hex			
0000	0x0	Single/partial–word read	core	yes
0001	0x1	Double–word read	core	yes
0010	0x2	Four–word read	core	yes
0011	0x3	Eight –word read	core	yes
0100	0x4	Single/partial–word write	core	yes
0101	0x5	Double–word write	core	yes
0110	0x6	Four–word write	core	yes
0111	0x7	Eight–word write	core	yes
1000	0x8	Single word DIO read return	GSC+	yes
1001	0x9	Double–word DIO read return	GSC+	yes
1010	0xA	Four–word DIO read return	GSC+	yes
1011	0xB	Eight–word DIO read return	GSC+	yes
1100	0xC	Reserved	undefined	n/a
1101	0xD	Error event indicator	GSC+	no
1110	0xE	Clear (four word read with clear)	GSC+	no
1111	0xF	Write Variable (one to eight words)	GSC–1.5X	yes

Note: Dino also supports the pended DMA read return transaction described in the GSC+ specification.

Table 15. GSC Transaction Support

6.5 > 40 MHz GSC operation

In order to further improve system performance, Dino takes advantage of a point–to–point GSC bus topology that supports GCLK frequencies higher than 40MHz. Higher GSC frequencies will break some critical timing paths in Dino's GSC interface, making the use of an external delay line necessary. Also, **at these higher frequencies fast back–to–back writes will not work and therefore can not be enabled**.

6.6 Fast back to back single word writes

Fast back–to–back writes can be enabled by writing a one to the LSB in the IO_FBB_EN register. Fast back–to–back writes are disabled at power on.

6.7 Power–on and GSC

At power on (while PON = 1'b0) the Dino's GSC pads provide a weak pull–up to logic1 for all GSC control and data lines. It is assumed that the CPU/IOA contains bus holders that hold bus state for proper GSC operation. A command reset will not reactivate these weak pull–ups.

6.8 Some GSC+ & GSC2X details

Dino implements a full set of GSC+ features and also the Type "F" transaction from GSC2X, so we call Dino a GSC1.5X+ device. As a GSC1.5X+ guest Dino receives the gsc_L signal to indicate whether to use extended GSC transactions. On Dino HPA BSRS 30, register 13, bit 31 is hardwired set to indicate to the host that the Type "F" transaction is allowed. The Type "F" transaction is the only GSC core feature enhancement in GSC1.5.

Once a pended read, DIO/DMA, is outstanding dino will retry any downstream transaction, thus removing the possibility of a deadlock condition. In light of U–turn not allowing type "F" transaction to be retried, **type "F" transactions and pended reads must be mutually exclusive**. Type "F" transaction can be turned off in U–turn or pended reads can be turned off in Dino.

7

7 PCI

7.1 Introduction

Dino's PCI interface is the PCI end of a bridge between HP's proprietary GSC (General System Connect) local bus and the industry standard PCI (Peripheral Component Interconnect) local bus. Numerous computer systems today support this high speed local bus, which offers access to standard industry I/O hardware components.

7.2 Features

- Clock domain of PCI is independent of GSC
- Graphics support
- Arbitration for 6 PCI devices (see "Arbitration" section).
- Maximum latency of 30us
- Average latency of 3us
- HPPA configurable PCI address space
- Bridge FIFO behavior is configurable for performance optimization
- PCI and GSC can have independent bus ownership
- Generation of configuration space, I/O space, and memory space cycles.
- Recognition and use of Memory Read Multiple, Memory Read Line, and Memory Write and Invalidate PCI commands.

7.3 Unsupported Functionality

- Dual Address Cycles (**C/BE[3:0]#**==1101) cannot be generated by Dino. Dual Address Cycles generated by another master will be ignored by Dino. Absolutely no logging will be performed within Dino.

PCI Command	Command Encoding	Dino Generates	Dino Responds
Interrupt Acknowledge	0000	No	No
Special Cycle	0001	Yes	No
I/O Read	0010	Yes	No
I/O Write	0011	Yes	No
Reserved4	0100	No	No
Reserved5	0101	No	No
Memory Read	0110	Yes	Yes

PCI Command	Command Encoding	Dino Generates	Dino Responds
Memory Write	0111	Yes	Yes
Reserved8	1000	No	No
Reserved9	1001	No	No
Configuration Read	1010	Yes	No
Configuration Write	1011	Yes	No
Memory Read Multiple	1100	No	Yes
Dual Address Cycle	1101	No	No
Memory Read Line	1110	Yes	Yes
Memory Write and Invalidate	1111	No	Yes

Table 16. PCI Transaction Support

7.4 PCI Assumptions

It is assumed that Dino is only a host bridge. Dino does not support the registers and other functions needed to be a non–host device on PCI. This means that Dino can not, for example, be used to build a device that lives on a PCI bus and provides access to GSC resources.

Also, it is assumed that any PCI transaction that is not directed to another PCI device (i.e. must go through the bridge to GSC) is a transaction directed to memory. This means that read prefetching for a PCI mastered transaction can be done without regard to side effects. In addition, upstream (DMA) reads on PCI without all byte enables asserted will turn into reads on GSC with all byte enables asserted.

7.5 Using PCI at frequencies higher than 33 MHz

Dino's PCI bus can run at frequencies higher than 33MHz when used with PCI Rev. 2.1 66MHz devices and a point to point bus topology. Dino does not support any architected PCI status registers so there is no 66Mhz Capable bit. However, the clock generation circuitry needs to look at the M66EN signal to generate the correct clock frequency for the higher than 33MHz frequency bus segment. Use of PCI higher than 33MHz is intended to be for very restricted configurations. See the "PCI Local Bus Specification" Rev. 2.1, Chapter 7 for more details.

7.6 PCI Signals

As a PCI Bridge Dino makes use of all the required and optional PCI signals listed in the "PCI Local Bus Specification Revision 2.0, **except:**

- **AD[63:32], C/BE[7:4]#,PAR64,REQ64#,ACK64#** —> Dino is a 32–bit PCI bridge.

- **LOCK#** —> The industry is moving away from using lock.
- **SBO#, SDONE** —> Cache support is not needed.

7.7 Accessing PCI Configuration Space thru PA I/O Space

PCI Configuration space will be accessed through two 32–bit HPA registers: PCI_CONFIG_ADDR (HPA offset 0x064) and PCI_CONFIG_DATA (HPA offset 0x068). Accessing the PCI Configuration Space is a two step process:

- Write the bus number, physical device number, function number and register number to the PCI_CONFIG_ADDR register.
- Perform a read or write to the PCI_CONFIG_DATA register.

Note: The contents of the PCI_CONFIG_DATA register will be byte swapped between GSC and PCI.

DINO's PCI bus number is 8'h00.

PCI_CONFIG_ADDR (PCI Configuration Address)

	31 24	23 16	15 11	10 8	7 0
0x064	**Reserved**	**BUS_NUM**	**DEV_NUM**	**Funct.**	**REG_NUM**

GSC Bit	Symbol	PCI_CONFIG_ADDR Description
31–24	Reserved	
23–16	BUS_NUM	Identifies the PCI bus number the configuration access is intended for. This field is only needed if Type=01.
15–11	DEV_NUM	Identifies the physical PCI device number the configuration access is intended for.
10–8	Function	Identifies the function number within a physical PCI device.
7–0	REG_NUM	Identifies the **word** address of the target function's configuration register. The two LSB will always read back 2'b0

When the BUS_NUM field is equal to DINO's Bus number, 8'h00, a configuration cycle will be started on Dino's bus with the following address, this is called a type 0 transaction.

31 11	10 8	7 2	1 0
Only one "1"	**Funct.**	**REG_NUM**	**0 0**

The one address bit that is set in the range [31:11] is determined by the DEV_NUM field. A DEV_NUM value of 0 to 15 maps to bit 16 to bit 31 respectively and a value of 16 to 20 maps to bit 11 to 15 respectively. No bit is set for a DEV_NUM value in the range 21 to 31.

If the BUS_NUM field is not equal to DINO's bus number, 8'h00, the contents of the PCI_CONFIG_ADDR[31:2] register is copied on to PCI_AD[31:2] lines with PCI_AD[0:1] set to 2'b01. This is a type 1 transaction. Type 1 transactions are ignored by all targets expect PCI to PCI bridges.

The following table is the PCI Configuration Space Header, Figure6–1 of the PCI 2.1 spec. This table has PCI byte ordering.

Offset	31			0
00h	Device ID		Vendor ID	
04h	Status		Command	
08h	Class Code			Revision ID
0Ch	BIST	Header Type	Latency Timer	Cache Line Size
10h	Base Address Register 0			
14h	Base Address Register 1			
18h	Base Address Register 2			
1Ch	Base Address Register 3			
20h	Base Address Register 4			
24h	Base Address Register 5			
28h	Cardbus CIS Pointer			
2Ch	Subsystem ID		Subsystem Vendor ID	
30h	Expansion ROM Base Address			
34h	Reserved			
38h	Reserved			
3Ch	Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line

Table 17. PCI Configuration Space Header

7.7.1 Generating PCI Special Cycles thru PA I/O Space

When the PCI_CONFIG_ADDR registers BUS_NUM is the equal to the DINO's bus number, 8'h00, DEV_NUM and Function fields are all ones, and the REG_NUM field is all zeros the next write to PCI_CONFIG_DATA register will generate a special cycle on DINO's PCI bus. If the BUS_NUM field does not equal DINO bus number then a type 1 transaction will be forwarded to PCI as described above.

Note: Dino is using a legal PCI configuration address to generate a PCI special cycle. System firmware and software should not attempt to read or write to this configuration address when walking the PCI bus through configuration address space.

7.7.2 Bus Walking

When performing bus walking, the PCI specification requires that firmware test for the presence of a device at each of the 32 possible bus numbers. If a device does not respond to a configuration cycle, Dino will terminate the cycle on PCI with a Master–Abort. If the cycle without a response was a configuration read, Dino will return a data value of 0xFFFF_FFFF. ***However, Dino will not go into Fatal Mode as a result of any Master–Abort during a configuration cycle.***

7.7.3 Configuration Access Endianness

The PCI configuration registers are little–endian, and thus the contents of the PCI_CONFIG_DATA register will be swapped. The byte–enables will also be swapped between GSC and PCI. Note that the contents of the PCI_CONFIG_ADDR register will NOT be swapped, Dino will generate the appropriate PCI Configuration address based on the PCI_CONFIG_ADDR register format defined above.

7.7.4 Example of a Configuration Read and Configuration Write

The following examples show how a Configuration Read and Write can be executed on a typical system.

A) Configuration Read of Configuration Register at offset 0x10(Base Address Register #0) for Device #4 (bottom slot of a Raven).

The following assumptions have been made for this example:

There is a PCI card in Slot 4

System = Raven U

Dino HPA = 0xF160_0000 (This value can be found with the following BCH command: *"db sim"*).

Firmware = Rev. 2.7 (with debug commands enabled)

PCI SLOT #4

PCI Function Number = 0

PCI Configuration Register = 0x10

Procedure:

1) Boot system to BCH

2) Change int the Manufacturing Menu by typing *"mfg"*.

3) Calculate the correct value for the PCI_CONFIG_ADDR register:

Device # = 0x4 = 00100b

Function # = 0x0 = 000b

Register # = 0x10 = 00010000b

PCI_CONFIG_ADDR = 00100 000 00010000b = 0x2010

4) Write the PCI Configuration Address to Dino's PCI_CONFIG_ADDR register (HPA + 0x64) by typing "*mw 0xf1600064 0x2010"*.

5) Now read one word (32 bits) from offset 0x10 of device #4 PCI Configuration Space Header via the PCI_CONFIG_DATA register (HPA + 0x68) by typing

"*mr 0xf1600068 1"*.

B) Configuration Write of Configuration Register at offset 0x4 (Command and Status Register) for Device #2 (bottom slot of a Merlin).

The following assumptions have been made for this example:

There is a PCI card in Slot 2

System = Merlin L2

Dino HPA = 0xFFF8_0000 (This value can be found with the following BCH command: *"db sim"*).

Firmware = Rev. 2.7 (with debug commands enabled)

PCI SLOT #2

PCI Function Number = 0

PCI Configuration Register = 0x04

Data to be written to PCI Register = 0x12345678

Procedure:

1) Boot system to BCH

2) Change int the Manufacturing Menu by typing *"mfg"*.

3) Calculate the correct value for the PCI_CONFIG_ADDR register:

Device # = 0x4 = 00010b

Function # = 0x0 = 000b

Register # = 0x10 = 00000100b

PCI_CONFIG_ADDR = 00010 000 00000100b = 0x1004

4) Write the PCI Configuration Address to Dino's PCI_CONFIG_ADDR register (HPA + 0x64) by typing "*mw 0xfff80064 0x1004*".

5) Convert the write data to Big Endian format from Little Endian:

Big = 0x78563412

Little = 0x12345678

6) Now write one Big Endian word (32 bits) to offset 0x04 of device #2 PCI Configura tion Space Header via the PCI_CONFIG_DATA register (HPA + 0x68) by typing

"*mw 0xfff80068 0x78563412*".

7.8 Accessing PCI I/O Space thru PA I/O Space

PCI I/O space will also be accessed through two 32–bit HPA registers: PCI_CONFIG_ADDR (HPA offset 0x064) and PCI_IO_DATA (HPA offset 0x06c). Accessing the PCI I/O Space is a two step process:

- Write of the target I/O byte address to the PCI_CONFIG_ADDR register.
- Perform a read or write to the PCI_IO_DATA register.

The lower half word written to this register,PCI_CONFIG_ADDR[15:0] will be the I/O address used for the PCI I/O transaction (no byte swapping will occur on the PCI_CONFIG_ADDR register). Note that this mechanism only allows single–word and sub–word accesses, and only to the first 64kB of PCI I/O space. Note, as is the case with PCI Configuration space accesses, the contents of the PCI_IO_DATA register will be swapped between GSC and PCI.

PCI I/O cycles provide full byte address, AD[1:0] contain the byte address. Use the following table when generating sub word transactions.

PCI_CONFIG_ADDR[0:1]	Byte Enable on Read/Write to PCI_IO_DATA register
32 Bit Word——2'b00	4'b0000
Upper 16 Bit Word — 2'10	4'b1100
Lower 16 Bit Word — 2'00	4'b0011
Byte Zero — 2'b00	4'b0111
Byte One — 2'b01	4'b1011
Byte Two — 2'b10	4'b1101
Byte Three — 2'b11	4'b1110

7.9 Accessing PCI Memory Space thru PA I/O Space

Dino provides a way to access dispersed ranges of PCI memory space from the fragmented available PA I/O space. There are 32–8MB chunks in PA I/O Space, and all but the first (used by boot ROM) and the last (used by CPU) can potentially be used for PCI.

Dino's IO_ADDR_EN register, located in Dino's HPA, is used to enable 8MB chunks of GSC I/O space to be used for accesses to PCI Memory Space. Any transaction whose address falls within an enabled chunk will be directly forwarded to the PCI bus. For example, assuming the 8MB chunk beginning at 0xF100_0000 has been enabled, a GSC read to address 0xF101_2340 will result in a PCI read to PCI memory space address 0xF101_2340.

IO_ADDR_EN (IO Address Enable)

	31	30 … 1	0
0x060	0	**IO_ADDR_EN**	0

GSC Bit	GSC Address Range Enabled
31	FF80_0000 thru FFFF_FFFF. This address space is reserved for the CPU and thus this bit is hardwired to zero.
30	FF00_0000 thru FF7F_FFFF
29	FE80_0000 thru FEFF_FFFF
28	FE00_0000 thru FE7F_FFFF
27	FD80_0000 thru FDFF_FFFF
26	FD00_0000 thru FD7F_FFFF

25	FC80_0000	thru	FCFF_FFFF
24	FC00_0000	thru	FC7F_FFFF
23	FB80_0000	thru	FBFF_FFFF
22	FB00_0000	thru	FB7F_FFFF
21	FA80_0000	thru	FAFF_FFFF
20	FA00_0000	thru	FA7F_FFFF
19	F980_0000	thru	F9FF_FFFF
18	F900_0000	thru	F97F_FFFF
17	F880_0000	thru	F8FF_FFFF
16	F800_0000	thru	F87F_FFFF
15	F780_0000	thru	F7FF_FFFF
14	F700_0000	thru	F77F_FFFF
13	F680_0000	thru	F6FF_FFFF
12	F600_0000	thru	F67F_FFFF
11	F580_0000	thru	F5FF_FFFF
10	F500_0000	thru	F57F_FFFF
9	F480_0000	thru	F4FF_FFFF
8	F400_0000	thru	F47F_FFFF
7	F380_0000	thru	F3FF_FFFF
6	F300_0000	thru	F37F_FFFF
5	F280_0000	thru	F2FF_FFFF
4	F200_0000	thru	F27F_FFFF
3	F180_0000	thru	F1FF_FFFF
2	F100_0000	thru	F17F_FFFF
1	F080_0000	thru	F0FF_FFFF
0	F000_0000	thru	F07F_FFFF.

7.10 Accessing PA I/O Space thru PCI Memory Space

Dino implements negative decoding for accesses to PA Space above F000_0000. Any PCI Memory transaction directed to a memory location *not* enabled by the IO_ADDR_EN registers, will be directly forwarded upstream to PA Space. The NEG_DEC bit in Dino's PCICMD (HPA offset 0x810) register must be set to enable Dino to perform negative decoding. In this address range, Dino will be a MEDIUM speed device as defined in the PCI spec. **Caution should be taken when addressing PA Space above F000_0000. If a transaction starts in one 8Mb chunk and finishes in the next 8Mb chunk it is assumed that the second chunk is *not* enabled in the IO_ADDR_EN register.**

7.11 Accessing PA Memory Space thru PCI Memory Space

Dino assumes that all addresses from 0x0000_0000 to 0xEFFF_FFFF are DMA transactions to PA memory space. All PCI memory transactions to these addresses are forwarded to the same address on the GSC bus provided the NEG_DEC bit in Dino's PCICMD (HPA offset 0x810) register is set to 1. Dino will be a FAST device for writes and a MEDIUM device for reads, where FAST and MEDIUM are defined in the PCI specification. Because of the special nature of address 0x0000_0000 to 0xEFFF_FFFF in HPPA, PCI devices should not be mapped into this range.

7.12 Accessing PA Space thru PCI I/O Space

Dino will not allow any PA Space to map to PCI I/O Space. Dino will simply not respond to any PCI transaction directed to PCI I/O Space. More specifically, Dino will not assert or drive **DEVSEL#**, **STOP#**, or **TRDY#**, if the PCI command is an "I/O Read" or "I/O Write"

7.13 Accessing Dino PCI Configuration Space From PCI

Dino's Configuration and Status registers will be accessed through Dino's HPA space, which can only be accessed through Dino's GSC interface. Dino's Configuration and Status registers can not be accessed through PCI Configuration transactions (Dino does not have a IDSEL input).

7.14 Responding to PCI Special Cycles

Though Dino has a mechanism for generating PCI Special Cycles, Dino will not respond to any PCI Special Cycles.

8 PERFORMANCE

8.1 Performance Summary

Dino is designed to achieve the highest level of performance that makes sense for a GSC to PCI bridge. Performance can, however, be degraded by programming the bridge incorrectly or using non–optimum PCI transactions. For example, multiword transactions on PCI that do not have all byte lanes enabled on each word will not map well between PCI and GSC thus generating numerous word/sub–word transactions, resulting in lost performance. Bridge hardware properly handles all legal PCI transactions, but using transactions that can be converted efficiently will result in better performance.

NOTE: U–Turn is the I/O adapter used with the PA7200, PA8000, and PA8200 processors.

Operation	PA7100	PA7300L	U–turn	Assumptions
DMA Write	128MB/s	128MB/s	100MB/s	40MHz GCLK Using 8–Word write transactions packed by the bridge. No memory contention.
DMA Read	85MB/s	85MB/s	 24MB/s 40MB/s	40MHz GCLK Assumes XQL is not used. Assumes XQL is used.
Direct I/O Write	70MB/s	100MB/s	100MB/s	40 MHz GCLK PA7100: 1 double word GSC write every 4 states PA7300LC: 1 8 word type “F” transaction every 10 states.
Direct I/O Read	14MB/s	14MB/s	10MB/s	40 MHz GCLK One single word read every 8 GSC states.

Table 18. Performance summary for Dino’s PCI to GSC bridge.

8.2 Performance Features

Dino’s performance enhancing features are described and analyzed in this section.

8.2.1 GSC Fast back to back DIO writes

In order to improve DIO write performance on GSC Dino implements fast back to back READYL response. This gives a peak DIO write bandwidth of (N/N+1)*160MB/Sec where N is the transaction size in words. It is important to note that single word write performance is maximized when the same Dino is accessed consecutively.

8.2.2 Coalescing DIO writes

The efficiency of DIO writes on PCI is maximized by coalescing GSC writes to consecutive addresses into long multiword writes on PCI. If GSC is running at 40MHz and PCI is running at 33MH, the host needs to use back to back 4 word writes in order use all the available PCI throughput, with coalescing turned on. If coalescing is disabled PCI bandwidth will always limit throughput at 40 and 33 MHz.

8.2.3 Variable length DIO write transactions

A new feature of GSC1.5 is the availability of a variable length write transaction. This transaction lets the host generate transactions with lengths between 1 and 8 words, providing a means of using the GSC bus more efficiently. The peak bandwidth of writes using this transaction is still (N/N+1)*160MB/Sec where N is the transaction size in words, however, using this transaction reduces the number of GSC transactions needed.

8.2.4 DMA read prefetching

In order to improve DMA read performance Dino prefetches data. The ROR register provides a means of selecting between four different read algorithms. These algorithms are implemented to provide different levels of speculative prefetching. Choosing the correct algorithm can have a very significant effect on performance. These algorithms also enable the use of the GSC XQL signal which provides a prefetch hint to GSC IOAs like U2.

8.2.5 Pended DIO and DMA read transactions

GSC+ pended transactions help improve bus efficiency in heavily loaded systems. These transactions release the bus while waiting for read data to return. It is useful to note that pended transactions hurt performance if the bus is not busy.

8.2.6 PCI Fast Back to Back Writes

Fast back to back transactions are implemented on PCI. Fast back–to–back writes can only be enabled if all of the devices on PCI are capable of supporting fast back–to–back writes.

8.3 Bridge Latency

Because Dino allows the GSC and the PCI clocks run totally independently, operations crossing the bridge incur a significant amount of latency. Because the implementation of Dino isn't complete, its not possible to specify exact numbers for these latencies that won't change. For example, there may be some opportunity to improve latency or certain design realities may cause latency to increase. With these caveats in mind, simulations have shown that average DIO write latency is currently in the neighborhood of 175 ns, measured from ADDVL assertion on GSC to **FRAME#** assertion on PCI. Average DIO read latency is currently in the neighborhood of 415 ns, measured from ADDVL assertion on GSC to READYL assertion on GSC.

The above times assume that GSC is running at 40 MHz and PCI is running at 33 MHz. The read time assumes the PCI target is as fast as possible (fast or medium decode speed and **TRDY#** low 2 clocks after **FRAME#**). Please note because of synchronization windows, the write latency for a given cycle could be about 16 ns more or less than the average, and the read latency for a given cycle could be 29 ns more or less than the average. The read number could be worse if GSC uses pended reads or if the read is a multi–word read. In the former case, arbitrating for the bus and starting a new transaction adds overhead. In the later case, all words of a multi–word read must be complete on PCI before the first word can be returned on GSC.

9 ENDIAN ISSUES

9.1 Problem

The byte–wise endian problem emanates from the need to access data information as 8–bit bytes some of the time and as 16–bit, 32–bit, or larger words some of the time. This problem wouldn't exist at the hardware level if a system was exclusively byte oriented or if a system was exclusively word oriented. Of course, because performance demands wider than 8–bit data operations and because byte oriented data types remain important, our systems must deal with both 8–bit and larger than 8–bit quantities.

There are two commonly used ways of packing sequential bytes into a larger data quantities: little endian byte ordering and big endian byte ordering. These are illustrated in Figure 5.

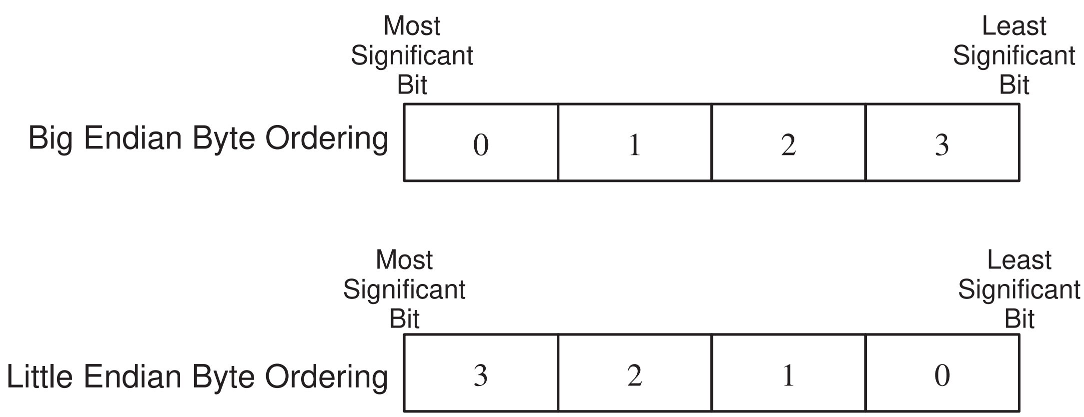


Figure 5. Byte Packing

Devices on the GSC bus conform to the big endian byte ordering convention, and devices on the PCI bus conform to the little endian byte ordering convention. The problem is how to move data between the GSC bus and the PCI bus. One option is to map the most significant bit on GSC to the most significant bit on PCI and the least significant bit on GSC to the least significant bit on PCI. This works well for 32–bit word oriented data but changes the way 8–bit and 16–bit data is accessed within a word. A second option is to map byte 0 on GSC to byte 0 on PCI and byte 3 on GSC to byte 3 on PCI. This works well for byte oriented accesses, but changes the way 16–bit and 32–bit data is viewed.

9.2 Dino Implementation

The second option of mapping byte 0 to byte 0 is the approach taken on Dino. This means that data on the most significant byte lane on GSC will be routed to the least significant byte lane on PCI. It has the benefit of working transparently for I/O drivers running with the PSW E–bit set.

To understand endianness fully, it is necessary to understand how the CPU behaves when it is in little endian or big endian mode. The *PA–RISC1.1 Instruction Manual, Third Edition* can be consulted for more information. Table 19 is intended to be a summary.

Item	CPU's PSW E–bit	
	Big Endian Mode	**Little Endian Mode**
Data in system memory	Stored as big endian.	Stored as big endian
Instructions in system memory	Stored as big endian.	Stored as little endian.
Instruction Fetches	No swap.	Processor swaps as they are brought in.
Loads/stores of halfwords, words, and double words	No swap.	Swap takes place as the data is read from or written to system memory or I/O.
Loads/stores of bytes.	No swap.	No swap.
	Note: Since data in a register is always flush right, and data in memory or on the GSC bus is always big endian, there is no difference in byte operations between the two modes.	
STBYS instruction (STORE BYTES SHORT)	See the *PA–RISC 1.1 Instruction Manual*. The details of this instruction are not important to this discussion.	
Data on GSC	0 | 1 | 2 | 3 Most Significant … Least Significant	
Data on PCI	3 | 2 | 1 | 0 Most Significant … Least Significant	
DMA Address Pointers for PCI devices as viewed in a CPU register	A[7:0] | A[15:8] | A[23:16] | A[31:24] Most Significant … Least Significant	A[31:24] | A[23:16] | A[15:8] | A[7:0] Most Significant … Least Significant
32–bit DMA Commands for PCI devices as viewed in a CPU register	C[7:0] | C[15:8] | C[23:16] | C[31:24] Most Significant … Least Significant	C[31:24] | C[23:16] | C[15:8] | C[7:0] Most Significant … Least Significant
PA Code generating program (e.g. compiler)	Compiles instructions as words or double words with no swapping.	Compiles instructions as words or double words with no swapping.

Table 19. Endian Comparison

Note that instructions are inherently word oriented, so in order to have the *words* appear the same in both big endian and little endian mode, the bytes of an instruction are swapped in little endian mode but not in big endian mode. As long as the process generating code and the process executing code run under the same E–bit mode this presents a consistent model. Only if a program is compiled under one endian mode and executed under the other does software byte swapping need to occur.

Figures 6 and 7 show the major data flows in the system in both processor modes. Word (32–bit) oriented data is shown for this illustration, but the data flows would be the same for halfword and doubleword data as well.

9.3 System Examples

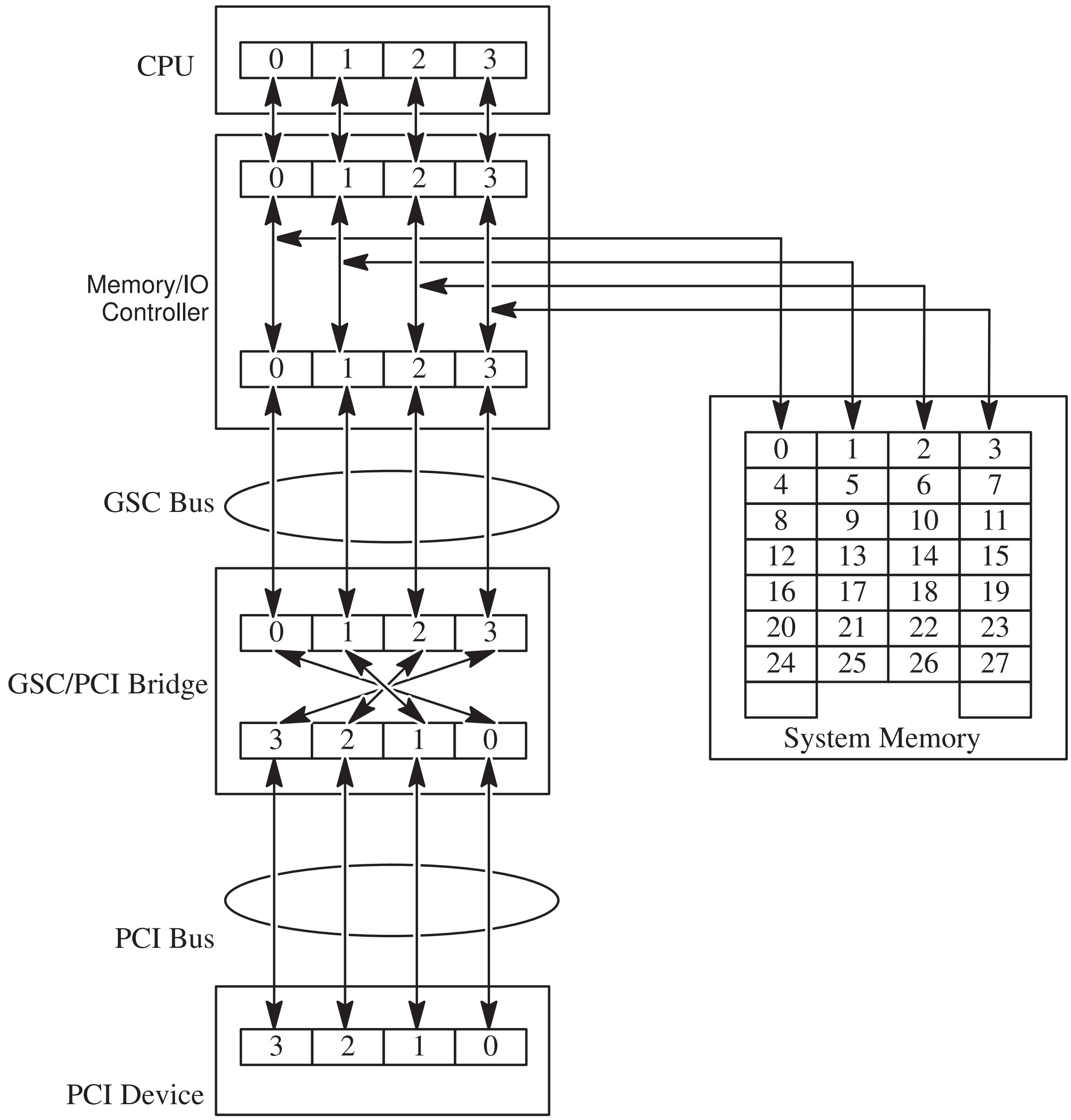


Figure 6. Big Endian Mode

With the processor's E–bit cleared (i.e. Big Endian Mode), byte wide accesses from the host to PCI devices will access the correct numbered byte on the PCI device.

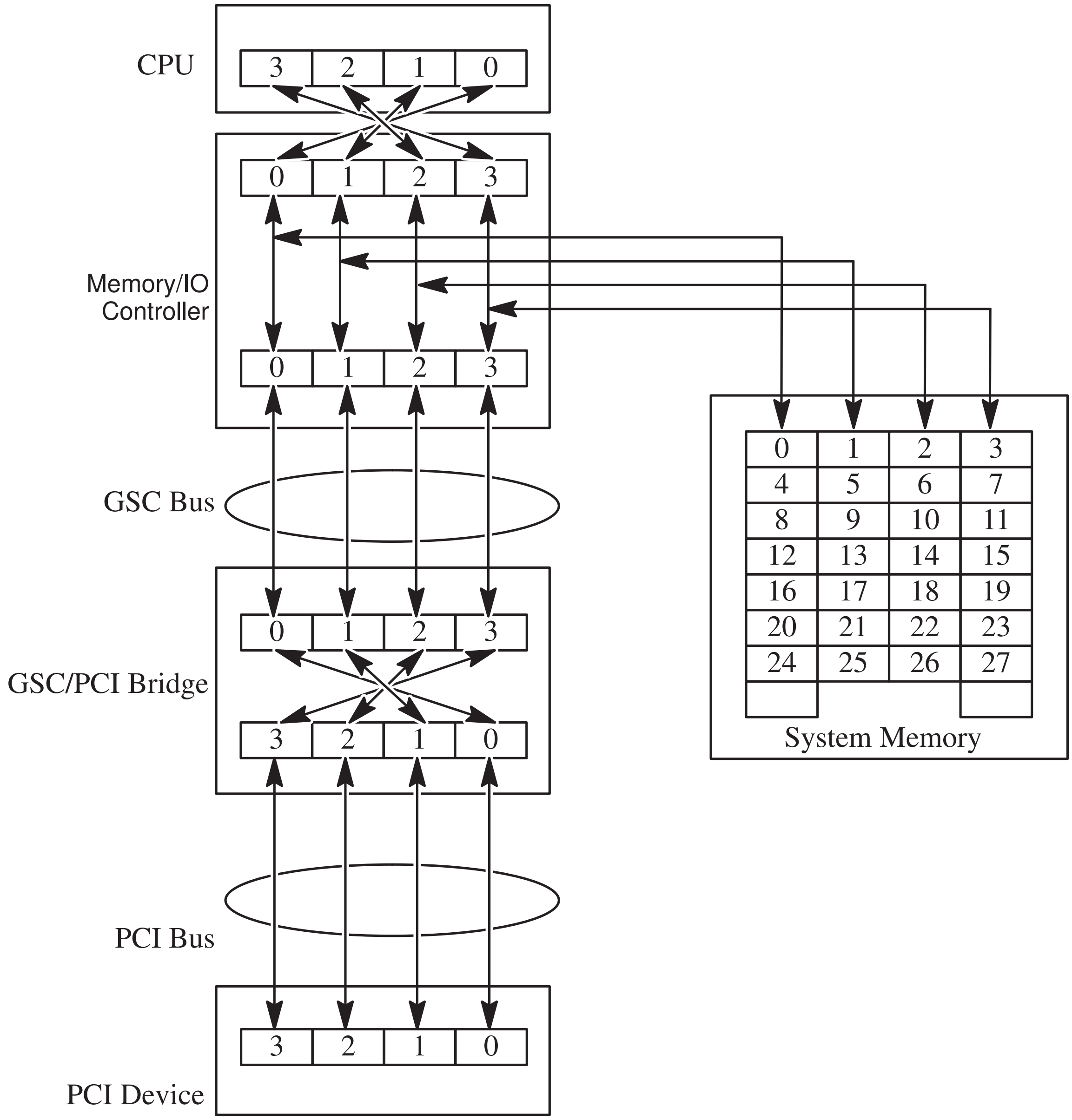


Figure 7. Little Endian Mode

With the processor's E–bit set (i.e. Little Endian Mode), byte wide accesses from the host to PCI devices will access the correct numbered byte on the PCI device, and 16–bit and 32–bit quantities will look the same to both the software running on the CPU and the PCI device.

Note that memory is consistently big endian for data regardless of whether the E–bit is set or cleared. This means that a data buffer with byte oriented data can be passed between a process running under little endian mode and a process running under big endian mode.

However, byte locations do get swapped for instruction storage depending on the state of the E–bit.

9.4 Programming PCI Devices

The programming model for PCI devices is different depending on whether the CPU is in big endian or little endian mode. The exceptions are generating PCI configuration and IO cycles. For information on these cycles, see the section defining the registers for configuration cycles.

When writing PCI drivers, using the CPU in Little Endian Mode is strongly recommended. This provides a consistent interface to software and obviates the need for the driver to do byte swapping.

9.4.1 Little Endian Mode

In little endian mode, the endian model for programming PCI I/O devices is the same as the model on an Intel X86 machine. Words, halfwords, and bytes can be constructed and sent to or received from the device as though the entire machine were little endian. If the documentation for a PCI peripheral lists a word (32–bit) register, it can be written to/read from as a word without byte swapping. If the documentation for a PCI peripheral lists a byte register, no modification to the address offset is needed to read/write it as a byte. Likewise, when using DMA, data structures can be constructed or referenced in memory just as though the whole system was little endian. DMA data structures should appear exactly like the PCI device documentation specifies.

9.4.2 Big Endian Mode

In big endian mode, if a software driver isn't restricted to using only byte accesses to a PCI device, the driver must be written with the intricacies of byte swapping in mind. The byte swapping model should be the same as for EISA devices. This means that for byte oriented data no swapping is needed, but for 32–bit or 16–bit oriented data (e.g. command words) each byte needs to be swapped. This is true for both DMA data structures in memory and direct I/O operations from the CPU.

10

10 DINO ERRORS AND ABNORMAL CONDITIONS

10.1 Dino Error Handling Overview

As a bridge between two busses, Dino must properly accommodate a number of abnormal conditions like data parity errors, time–outs, and incorrectly sized transactions. Dino also posts transactions, improving performance but making it difficult to contain errors. Non–contained errors on Dino happen when a posted transaction eventually completes in some sort of error on its target bus and the transaction on the initiating bus has long since completed without an error. In general, all non–contained errors on Dino are fatal and will put the bridge into *Fatal Mode*. Similarly, asserting ERRORL on GSC or asserting **SERR#** or **PERR#** on PCI will put Dino into Fatal Mode. In all error cases the correct GSC and PCI bus protocol will be followed.

10.2 Fatal Mode

Dino enters *Fatal Mode* in response to a number of different error conditions that require processor intervention. When in *Fatal Mode* Dino will not communicate in any way with PCI devices and only some HPA registers will be accessible. The processor will eventually attempt to access a disabled Dino resource, an action that will result in a bus error: in this way Dino passively signals a bridge error condition. *Fatal Mode* also keeps potentially bad data from propagating further.

In *Fatal Mode* DIO writes to disabled addresses on PCI or registers inside Dino will complete but the data will not transfer. DIO Reads of disabled registers or PCI will result in a GSC timeout. DMA from PCI is disabled. Interrupt transactions are also disabled.

The following registers are the only ones that can be accessed in the normal way during *Fatal Mode*:

- IO_COMMAND
- IO_STATUS
- IO_GSC_ERR_RESP
- IO_PCI_ERR_RESP
- IO_ERR_INFO

Fatal Mode modifies as little of Dino's state as possible, making it easier to analyze the error condition. The state of the following registers will not be modified by going into fatal mode:

- SRS Registers except IO_CONTROL, IO_STATUS
- ARS Registers except IO_GSC_ERR_RESP, IO_PCI_ERR_RESP, IO_ERR_INFO
- BSRS Registers
- HVRS Registers except PCISTS, PCICMD

As far as error logging is concerned, Fatal Mode logs the errant address and sets the proper bits in the IO_ERR_INFO and IO_STATUS registers. **To get Dino out of *Fatal Mode* a command reset needs to be issued on GSC.**

10.3 Less Than Fatal Mode

Less Than Fatal Mode is designed to help systems that do not wish to shut down with something as drastic as Fatal Mode. Less Than Fatal Mode generates an interrupt after the error occurs, keeps the errant address and logs the error as a soft error in the IO_STATUS HPA register. Less Than Fatal Mode also does not block access to any HPA registers and does not force parity errors on outgoing data. **Less than Fatal Mode allows bad data to flow through Dino.**

10.4 Preventing the propagation of bad data

In order to maximize bus efficiency and improve performance Dino posts transactions. Posting of transactions means that data parity errors on the requesting bus are no longer tied to the transaction—they are uncontained. Once the error is detected Dino enters Fatal Mode (as discussed above). In Fatal Mode the Dino forces parity errors on all data leaving the bridge on both the PCI and GSC sides, except the data coming from the HPA registers. In this way both the GSC based CPU gets and the PCI connected peripherals should detect parity errors, signalling a need to shut down their data paths or risk further propagation of bad data.

If the LTFM (Less Than Fatal Mode) bit is turned on, Dino simply generates an interrupt but does not force parity errors. In this way a system that is not too worried about bad data can perhaps recover more gracefully.

10.5 Dino error reporting and logging

Dino reports errors to help contain bad data and logs errors to help system diagnostics and error recovery. If the LTFM bit = 1’b0 bridge errors are reported by forcing parity errors and blocking transactions. If the LTFM bit = 1’b1 bridge errors are reported by generating an interrupt. Errant addresses from PCI and GSC, when Dino is a master, are logged in separate registers. When Dino is a target on either bus two separate bits log GSC and PCI error status.

10.5.1 Errors and the IO_Status register

The IO_Status register contains general error status. The LTFM (Less Than Fatal Mode) bit in the BRDG_FEAT register changes the way bits are set in the IO_Status register. If the LTFM= 1’b0 any kind of error will put Dino into Fatal Mode, setting the *fe* bit and putting the proper vector into the estat register. While the same errors with LTFM= 1’b1 will set the *se* bit and log a different code into the *estat* fields. CMD_CLEAR clears the *se* and *estat* fields.

IO_STATUS (Status Register)

	0–15 (bits 31–16)	16–31 (bits 15–0)
0x034	Undefined	**IO_STATUS**

PA Bit	Symbol	IO_STATUS Description	GSC Bit
0–15	Undefined		31–16
16–21	estat	Contains an error status code corresponding to the most severe error currently logged. Dino will only have 3 possible vectors stored in this field: 1) 5'b00000 –clear, 2) 5'b00001 – less than fatal, 3) 5'b00011 –fatal. See the "Dino Errors and Abnormal Conditions" section.	15–10
22	se	This bit is always zero unless Dino is in "Less Than Fatal Mode".	9
23	he	Hardwired to 0 because Dino does not distinguish this type of ERROR. See the "Dino Errors and Abnormal Conditions" section.	8
24	fe	When set, indicates that a fatal error was detected by Dino.	7
25	ry	When set, indicates that Dino is ready to accept commands. Probably will be hardwired to 1.	6
26–27	Reserved		5–4
28	lp	Set if this status register applies to lower bridge port. This bit is hardwired to 0 in Dino (this is an upper bridge port status register).	3
29–31	pwrstat	Remote bus power status. Hardwired to 000 in Dino.	2–0

Table 20. Status Register Bit Definition

10.5.2 Error address information

Dino posts transactions to and from two busses, meaning that it is possible for two addresses to simultaneously have an associated error. For this reason Dino keeps a separate PCI error responder and GSC error responder register in the HPA SRS (Supervisory Register Set). Here is a summary of how the HPA registers that log errant transaction information work.

IO_GSC_ERR_RESP (GSC Error Responder Address)

	0–31 (bits 31–0)
0x040	**IO_GSC_ERR_RESP**

- IO_GSC_ERR_RESP: This register logs the 32–bit address sent from Dino to a GSC target when an error is signalled. **This value is only valid when the vag bit in the IO_ERR_INFO register is set.**

If data parity error occurs on a DIO read return the value of IO_GSC_ERR_RESP register will be 32'h0000_0000.

IO_PCI_ERR_RESP (PCI Error Responder Address)

	0 … 31
0x048	**IO_PCI_ERR_RESP**
	31 … 0

- IO_PCI_ERR_RESP: This register logs the 32–bit address sent from Dino to a PCI target when an error is signalled. **This value is only valid when the vap bit in the IO_ERR_INFO register is set.**

IO_ERR_INFO (Error Logging Information)

	0 …	28	29	30	31
0x044	Undefined	**epci**	**egsc**	**vap**	**vag**
	31 …	3	2	1	0

- IO_ERR_INFO: When *vag* is 1, the contents of the IO_GSC_ERR_RESP register are valid. When *vap* is 1, the contents of the IO_PCI_ERR_RESP register is valid. When *egsc* is 1, an error happened on GSC while Dino was **not** the master. When *epci* is 1, an error happened on PCI when Dino was **not** the master.

10.6 Parity Errors

Parity is generated and checked on both PCI and GSC. On PCI data parity errors are signaled with **PERR#** and address parity errors are signaled with **SERR#**. Asserting either of these signals on the PCI side of the bridge will put Dino into *Fatal Mode*. On GSC both address parity errors and data parity errors result in the assertion of ERRORL. Dino will enter *Fatal Mode* if it sees ERRORL asserted.

10.7 Bus Walking

The existence or non–existence of I/O devices is often determined by testing their expected I/O address locations. If the device is not present a time–out or bus error will happen, thus signalling a missing device. With PCI, however, only configuration space is enabled for PCI devices after a PCI reset. Furthermore, non–responding addresses in configuration space do not cause GSC time–outs. For this reason, the presence of a time–out cannot be used to indicate a non–existent device. Instead, the read data needs to be compared to a data value of all 1's (0xFFFF_FFFF for a 32–bit read). See the PCI section of this ERS for more details.

10.8 Non–Responding Addresses

Any time–out on GSC with DINO as the target will put DINO into fatal mode. The system time–out value needs to be set so that data is returned before a time–out occurs.

At a given time, a read of a given PCI address may not generate a response from a PCI target. This may happen if there is no device at that address, if that particular address within the devices address map is not implemented, or if the address range is not currently enabled (PCI devices power up with all memory and IO space addresses disabled). A read directed to a non–responding addresses when not in PCI Configuration space will result in a time–out on GSC and the ERRORL signal will be asserted. This will put Dino into Fatal Mode, meaning that the CPU needs to do a directed reset to restart DINO.

It is important to note that a directed reset to Dino alters the Dino's state as little as possible. More specifically, directed resets to Dino will **not** alter the following registers:

- ☐ SRS Registers except IO_CONTROL, IO_STATUS
- ☐ ARS Registers except IO_ERR_RESP, IO_ERR_INFO, IO_ERR_REQ
- ☐ BSRS Registers
- ☐ HVRS Registers except PCISTS, PCICMD

10.9 Improper DIO transactions

Improper DIO transactions happen when the processor tries to do a legal DIO transaction to a device that doesn't support this type of transaction. This usually happens because of a programming error and may cause an HPMC (High Priority Machine Check).

If a PCI device asserts DEVSEL#, then it owns the starting address of a DIO transaction initiated by the CPU. The PCI device may terminate the transaction if it is not entirely supported by the device. Terminating the transaction on PCI will eventually result in an HPMC, unless Dino is in less–than–fatal" mode. In the DIO read case an HPMC will happen almost immediately. DIO writes may take some time to HPMC because of transaction posting. Remember: doing improperly formed down-stream transactions may crash the computer.

10.10 Error behavior examples

This section illustrates Dino's error behavior through a table of examples. Table 21 shows how some error conditions are dealt with on GSC and PCI. In the "Dino Register Fields Affected" column, a register name is given, then optionally followed by a dash with one or more register fields. All fatal error modes affect the IO_STATUS register, fields "estat" and "fe"

Initiator (Trans Type)	Bus where Error Occurs	Condition	GSC Action	PCI Action	Dino Register Fields Affected	Error Mode
GSC (DIO)	GSC	**Data Parity Error.**	**The host (which mastered the transaction) will see or generate ERRORL and take an HPMC.**	**Reads finish normally.** **Writes will not pass bad data. May or may not start on PCI.**	**IO_ERR_INFO–egsc**	**Fatal**
GSC (DIO)	PCI	**Data Parity Error.**	**The currently in progress GSC transaction completes and no further transactions are processed.**	**PCI transaction causes the PERR# signal to be asserted as described in the PCI specification.**	**PCISTS–DPD, DPE, SSE; IO_PCI_ERR_RESP; IO_ERR_INFO–vap**	**Fatal**
PCI (DMA)	PCI	**Data Parity Error**	**The currently in progress GSC transaction completes and no further transactions are processed.**	**PCI transaction causes the PERR# signal to be asserted as described in the PCI specification.**	**PCISTS–DPE, SSE; IO_ERR_INFO–epci**	**Fatal**

Initiator (Trans Type)	Bus where Error Occurs	Condition	GSC Action	PCI Action	Dino Register Fields Affected	Error Mode
PCI (DMA)	GSC	**Data Parity Error.**	**ERRORL is pulled by the data "sink" and the error is logged. GSC arbitration to the bridge is disabled.**	**PCI transaction is posted normally.**	**IO_ERR_INFO–vag; IO_GSC_ERR_RESP; PCISTS–SSE**	**Fatal**
GSC (DIO)	PCI	**PCI Target signals Target–abort.**	**Any GSC transaction involving Dino completes and no further transactions are processed.**	**PCI transaction causes the PERR# signal to be asserted as described in the PCI specification.**	**PCISTS–RTA; IO_PCI_ERR_RESP; IO_ERR_INFO–vap**	**Fatal**
GSC (DIO)	PCI	**PCI Target asserts DEVSEL# normally and then fails to eventually assert TRDY# or STOP# as required.**	**Dino will continue to process transactions normally until its FIFOs fill up or a DIO read is attempted. Once either happens Dino will not assert READYL and a GSC timeout will occur.**	**PCI will hang mid cycle as long as the target doesn't assert TRDY# or STOP#.**	**None at first.**	**None**
GSC (DIO)	GSC	**Timeout Error or Address Parity Error.**	**The processor asserts ERRORL and takes an HPMC because of the timeout on GSC.**	**PCI never saw a transaction get started. Nothing happens on PCI.**	**None.**	**Fatal**
PCI (DMA)	PCI	**Address parity error.**	**Dino's GSC interface does nothing.**	**Results in an SERR#. *This is the only case where dino will drive SERR#***	**PCISTS–SSE**	**Normal**

Initia-tor (Trans Type)	Bus where Error Occurs	Condition	GSC Action	PCI Action	Dino Register Fields Affected	Error Mode
GSC (DIO)	PCI	**Timeout error (DEV-SEL# isn't as-serted on time). This as-sumes the bridge decode was val-id.**	**On a write transac-tion READYL is as-serted (writes are posted) but data is lost. On a read READYL is not as-serted resulting in a time–out.**	**Transaction com-pletes with a Mas-ter–abort. If not a Configuration or Special cycle, the RMA bit in the PCISTS register will be set. If a con-figuration read, data of 0xFFFF_FFFF will be returned.**	**PCISTS–RMA; IO_PCI_ERR_RESP; IO_ERR_INFO–vap**	**Fatal**
PCI (DMA)	GSC	**Timeout error or Address Parity Error.**	**GSC ERRORL is asserted by the processor as a result of the time–out.**	**SERR# is asserted when the GSC timeout is detected.**	**PCISTS–SSE**	**Fatal**
PCI (DMA)	PCI	**Timeout error. (DEV-SEL# isn't as-serted on time.)**	**Any available data is transferred to GSC. If no data is transferred a GSC transaction MUST not be started.**	**This is a master ter-minated transaction on PCI. This is a more or less normal condition.**	**None.**	**Nor-mal**

Table 21. Bridge Error Condition Reference Table

11 INTERRUPTS

11.1 Overview

Dino will serve as the interrupt controller for both PCI and GSC. Dino has two independent interrupt controllers (int0 and int1). Each interrupt source can be programmed to use either of the controllers. There are two independent interrupt address registers which allow two separate IO_EIR addresses and group codes. Dino does not swap bytes when reading or writing these registers so they should be accessed in big–endian mode.

11.2 Register Definitions

There are 7 registers in Dino associated with interrupts. The registers are defined below in table 22.

Register	Symbol	Address Offset	R/W	Description
Interrupt Request Register 0	IRR0	0x00C	R	The IRR0 contains the status of all requesting interrupts that are mapped to int0. A 1 in an IRR0 bit indicates that the corresponding interrupt is pending and enabled. When an IRR0 bit is set it will cause Dino to generate an interrupt transaction to the address and group codes stored in IAR0.
Interrupt Address Register 0	IAR0	0x004	R/W	This register contains the address of the IO_EIR and the group code that will be used when a device mapped to int0 by the ICR issues an interrupt.
Interrupt Request Register 1	IRR1	0x014	R	IRR1 is identical to IRR0 except that it is for interrupts that are mapped to int1 in the ICR
Interrupt Address Register 1	IAR1	0x010	R/W	This register contains the address of the IO_EIR and the group code that will be used when a device mapped to int1 by the ICR issues an interrupt.
Interrupt Mask Register	IMR	0x018	R/W	The IMR is used to mask pending interrupts. A 1 in an IMR bit enables the corresponding pending interrupt to create an interrupt request. The IMR is cleared at reset.

Register	Symbol	Address Offset	R/W	Description
Interrupt Pending Register	IPR	0x01C	R/W	The IPR is used to latch incoming interrupts and indicate them as pending. The assertion of an internal interrupt signal causes the corresponding IPR bit to be set to 1. Writes to this register are intended for diagnostic use only and will cause the entire register to be cleared Bts in this register are only set when there is a inactive to active transition on corresponding interrupt lines.
Interrupt Control Register	ICR	0x024	R/W	The ICR is used to control whether an interrupt source is mapped to int0 or int1. This register is cleared at reset.
Interrupt Level Register	ILR	0x028	R	Bits in the ILR monitor the level of incoming interrupt lines. Asserted interrupt lines are always indicated by a 1.

Table 22. Interrupt Registers

The interrupt registers appear to be 32–bits and are accessed as such. However, most of the bits are not implemented for each register. The un–implemented bits are not affected by writes and are undefined for reads.

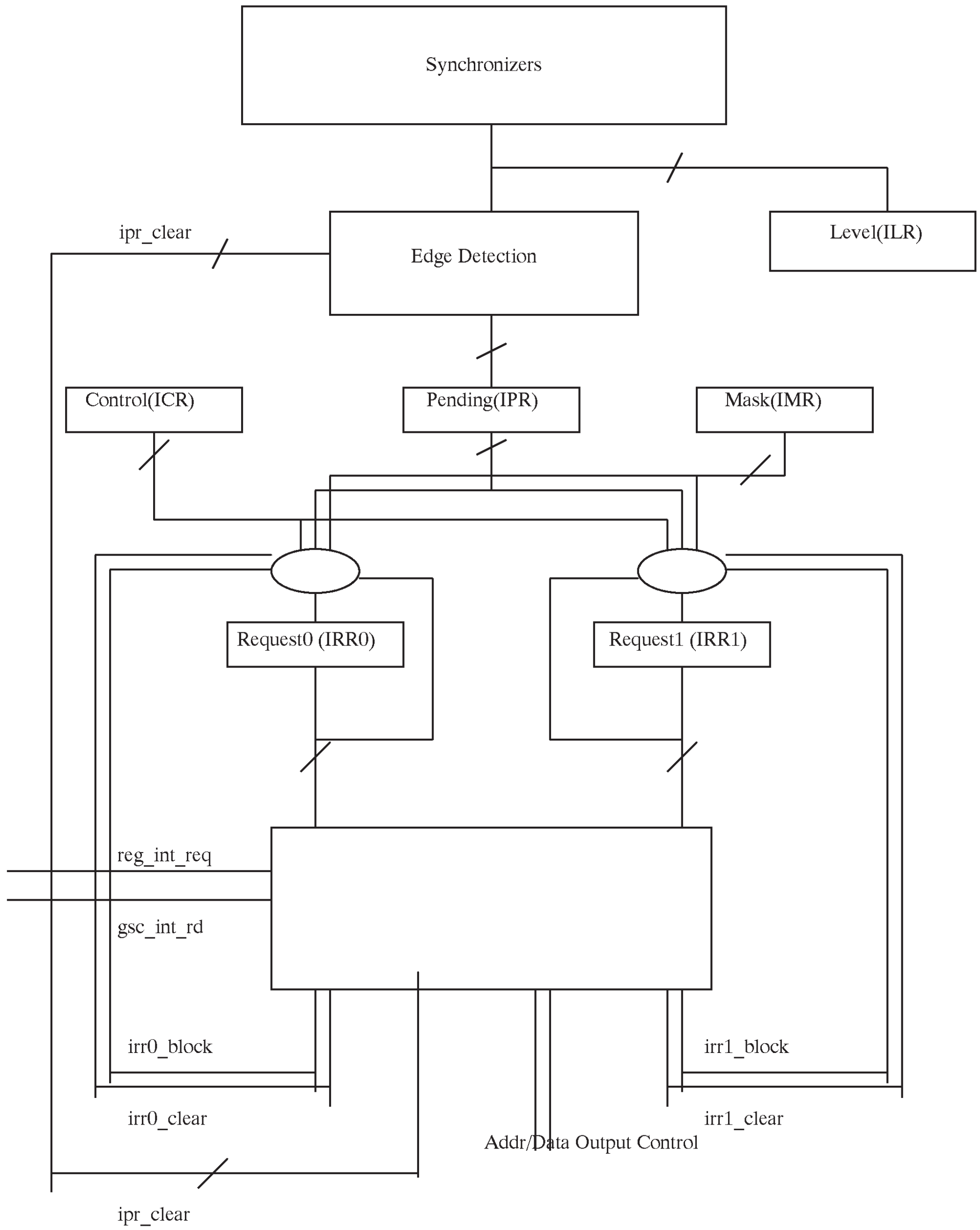
11.3 Interrupt Operation

If an interrupt source in Dino generates an inactive to active transition on its interrupt line, a corresponding bit in the IPR will be set to 1. If that interrupt is enabled (IMR bit=1) then the same bit in the ICR will determine if the corresponding bit will be set in IRR0 or IRR1. If the ICR bit is 0, the same bit of the IRR0 will be set and Dino will do a write transaction using the data and address specified in the IAR0. The contents of the IRR0 and the corresponding bits of the IPR are cleared on the clock cycle following a read of the IRR0. If the ICR bit is set, then IRR1 and IAR1 will be used instead of IRR0 and IAR0.

The ILR is used to monitor the state of interrupt inputs and is always assumed active high, regardless of whether the interrupt state is negative of positive true.

Note: The ICR register should not be modified unless all interrupts have been masked. (That is IMR set to all zeros.)

The following figure shows the connection of all the interrupt registers.


Synchronizers
ipr_clear
Edge Detection
Level(ILR)
Control(ICR)
Pending(IPR)
Mask(IMR)
Request0 (IRR0)
Request1 (IRR1)
reg_int_req
gsc_int_rd
irr0_block
irr1_block
irr0_clear
irr1_clear
Addr/Data Output Control
ipr_clear

11.4 Interrupt Register Bit Assignments

Table 23 shows the implemented bits for the IRR0, IRR1, IMR, ICR and IPR in Dino. This table shows both big and little endian bit numbering.

Bit number		Interrupt Source
Big	Little	
31	0	PCI INTA
30	1	PCI INTB
29	2	PCI INTC
28	3	PCI INTD
27	4	PCI INTE
26	5	PCI INTF
25	6	GSC External Interrupt
24	7	Bus Error for "Less Than Fatal Mode"
23	8	PS2
22	9	Unused
21	10	RS232

Table 23. ILR, IPR, IMR, ICR, IRR0 and IRR1 Bit Definition

The format for the IAR0 and IAR1 registers is shown below in figure 8.

big endian	0 … 26	27 … 31
	IO_EIR Address	Group code
little endian	31 … 5	4 … 0

Figure 8. Interrupt Address Registers

If there is an interrupt request Dino will master the follow one word write on the GSC bus.

Address

big endian	0 … 26	27 … 31
	IO_EIR Address	5'b0
little endian	31 … 5	4 … 0

Figure 9. Interrupt Address

Data

big endian	0 … 26	27 … 31
	27'b0	Group Code
little endian	31 … 5	4 … 0

Figure 10. Interrupt Data

12 ARBITRATION

12.1 Dino Arbitration Overview

GSC and PCI use a central arbitration scheme. A PCI arbitration controller is located inside the Dino chip. A GSC arbitration controller is not included in the Dino chip.

12.2 GSC Arbitration Control

The original definition of Dino included a GSC Arbiter. This functionality has been eliminated, and is assumed to exist elsewhere (e.g. the Clark chip).

12.2.1 GSC Arbitration Mask Register

The GSC arbitration mask register (GMASK) is a read/write register located in Dino's HPA space (offset 0x800). The GMASK provides a means to prevent Dino from requesting the GSC bus.

Bit	Symbol	Description
0	GMASK	GSC Request mask – if GMASK=1'b1, the Dino Bridge will never request the GSC bus.

Table 24. GSC Arbitration Mask Register

Software must clear the GMASK bits before any other device can request the bus.

12.3 PCI Arbitration Control

The PCI Arbiter can support seven devices. Six of the seven are external to the Dino chip. Dino will support a simple arbitration priority scheme. If no other device is requesting the bus, the PCI bus will be parked on Dino. Table 25 identifies the potential PCI bus masters.

Device	Location	Signal Sense
PCIA	outside Dino	negative true
PCIB	outside Dino	negative true
PCIC	outside Dino	negative true
PCID	outside Dino	negative true
PCIE	outside Dino	negative true

Device	Location	Signal Sense
PCIF	outside Dino	negative true
Dino Bridge	inside Dino	negative true

Table 25. PCI Bus Masters

12.3.1 PCI Arbitration Mask Register

The PCI arbitration mask register (PAMR) is a read/write register located in Dino's HPA space (offset 0x804). The PAMR provides a means to prevent a PCI device from being granted the bus. Table 26 defines the PAMR bits. If a bit in the PAMR is set to a 1, the request is disabled and the associated device(s) will never be granted the PCI bus.

Bit	Symbol	Description
31–7		Undefined
6	PERMF	PCI ext. request mask F – if PERMF=1, the devices connected to the PCIF arbitration lines will never be granted the bus.
5	PERME	PCI ext. request mask E – if PERME=1, the devices connected to the PCIE arbitration lines will never be granted the bus.
4	PERMD	PCI ext. request mask D – if PERMD=1, the devices connected to the PCID arbitration lines will never be granted the bus.
3	PERMC	PCI ext. request mask C – if PERMC=1, the devices connected to the PCIC arbitration lines will never be granted the bus.
2	PERMB	PCI ext. request mask B – if PERMB=1, the devices connected to the PCIB arbitration lines will never be granted the bus.
1	PERMA	PCI ext. request mask A – if PERMA=1, the devices connected to the PCIA arbitration lines will never be granted the bus.
0	PIRM	PCI int. request mask – if PIRM=1, the Dino bridge will never be granted the bus. This bit is hardwired to 0.

Table 26. PCI Arbitration Mask Register

Dino's bridge will always be enabled for arbitration. For more information on disabling Dino please refer to the IO_CONTROL register in the Bridge Registers Chapter.

All external PCI devices power–up with arbitration disabled. Software must clear the PAMR bits before any external device can request the bus.

12.3.2 PCI Arbiter Priority Configuration

Dino will support a dual round–robin arbitration scheme, allowing each device to be software configured as high priority or low priority. For each low priority device the Dino PCI Arbiter services, each of the high priority devices are serviced.

The PCI arbitration priority register (PAPR) is a read/write register located in Dino's HPA space (offset 0x808). The PAPR allows the PCI devices to be configured as a high or low priority arbitration device. Table 27 defines the PAPR bits. If a bit in the PAPR is set to a 1, the request is a high priority request.

Bit	Symbol	Description
31–7		Undefined
6	PCIPF	PCI priority F – if PCIPF=1, the external request line PCIF will be set to high–priority.
5	PCIPE	PCI priority E – if PCIPE=1, the external request line PCIE will be set to high–priority.
4	PCIPD	PCI priority D – if PCIPD=1, the external request line PCID will be set to high–priority.
3	PCIPC	PCI priority C – if PCIPC=1, the external request line PCIC will be set to high–priority.
2	PCIPB	PCI priority B – if PCIPB=1, the external request line PCIB will be set to high–priority.
1	PCIPA	PCI priority A – if PCIPA=1, the external request line PCIA will be set to high–priority.
0	BRDGP	Bridge priority – if BRDGP=1, the PCI Bridge request line will be set to high–priority.

Table 27. PCI Arbitration Priority Register

The PCI Arbitration Priority Register will power–up cleared, with all devices receiving equal priority. Software can set the appropriate bits depending on which devices require arbitration priority. Note, with this scheme, high–priority devices collectively share n/(n+1) of the bus bandwidth, where n is the number of high priority devices. Thus, as more devices are assigned high priority the bandwidth available to each decreases.

12.3.3 Disabling the PCI Arbiter

The Dino chip can be configured to work on a PCI bus when it is not the PCI arbiter. The bit PARB_SLAVE in the DAMODE register must be set high if Dino is not the PCI arbiter.

When Dino is not the PCI arbiter, the pcigntd_L signal becomes Dino's PCI bus request signal and the pcireqd_L signal becomes Dino's PCI grant signal.

Note, when Dino is in PARB_SLAVE mode, it is an arbitration slave–only and Dino will not ever assert its pcignt[a,b,c,e,f]_L signals.

Note: If Dino is in external arbitration mode, then Dino may incorrectly assume that it owns the PCI bus one clock after GNT# goes away. If the arbiter deasserts Dino's GNT# coincident with another GNT# on the last cycle of a frame, then Dino may drive fight with the other PCI

device. Although it is possible to design an external arbiter that works around this problem in Dino, FSL does not plan to support external arbitration mode in Dino.

12.3.4 Expansion Mode Arbitration

Each arbitration pair is programmable for expansion mode, bits 7–1 of the DAMODE register. If the corresponding bit is set in the DAMODE register, that device is in expansion mode. In expansion mode once a device is granted the bus that device will keep the bus until request is deasserted.

12.3.5 Dino Arbitration Mode Register

The Dino arbitration mode register (DAMODE) is a read/write register located in Dino's HPA space (offset 0x80C). The DAMODE register is a eight bit register used to control PCI arbiter Expansion Mode and PCI arbiter slave mode. Table 28 defines the DAMODE bits.

Bit	Symbol	Description
31–8		Undefined
7	EMODEF	if EMODEF=1, the devices connected to PCIF lines will be in Expansion mode.
6	EMODEE	if EMODEE=1, the devices connected to PCIE lines will be in Expansion mode.
5	EMODED	if EMODED=1, the devices connected to PCID lines will be in Expansion mode.
4	EMODEC	if EMODEC=1, the devices connected to PCIF lines will be in Expansion mode.
3	EMODEB	if EMODEB=1, the devices connected to PCIF will be in Expansion mode.
2	EMODEA	if EMODEA=1, the devices connected to PCIF will be in Expansion mode.
1	EBRD	if EBRD=1, Dino's bridge will be in Expansion mode.
0	PARB_SLAVE	PCI Arbitration Slave – if PARB_SLAVE=1, then Dino will become an arbitration slave. (See *Disabling the PCI Arbiter* section above).

Table 28. Dino Arbitration Mode Register

Note, if software wishes to change PAPR, Expansion Mode settings or PARB_SLAVE, this change should only be made while all bits in the PAMR register are set (we don't want to drive grant signals while it's arbitration mode is being changed!).

Note that the Expansion Mode bits makes no difference if the PARB_SLAVE bit is set (Expansion mode only applies when Dino is the PCI arbiter).

13 CLOCKS AND RESET

13.1 Introduction

Dino receives clock signals for GSC and PCI; both of these clocks can run at different frequencies, On the GSC side, Dino receives a synchronous GSC reset signal. The system power supply generates an asynchronous “glitch–free” power–on (PON) signal that puts the bridge I/O pads into the proper state for system power on. PCI reset is generated by Dino by program control via an HPA register or as a result of the GSC reset signal.

Electrically, clock signals for GSC are differential ECL running at PECL levels, while the PCI clock is point to point TTL. All the reset signals are assumed to use TTL levels.

13.2 Features

- GSC clock 0 – 80MHz —> GCLK is 0–40MHz
- Dino receives synchronous reset from the CPU.
- Dino responds to broadcast reset.
- Supports PCI clock rates up to 33MHz.
- PCI reset can be activated via program control through the GSC HPA registers
- Asserting PON activates a weak pull–up on GSC signals.

13.3 Dino Clocking and Reset Block Diagram

A block diagram showing how clocks and resets are connected to and from Dino is shown in Figure11.

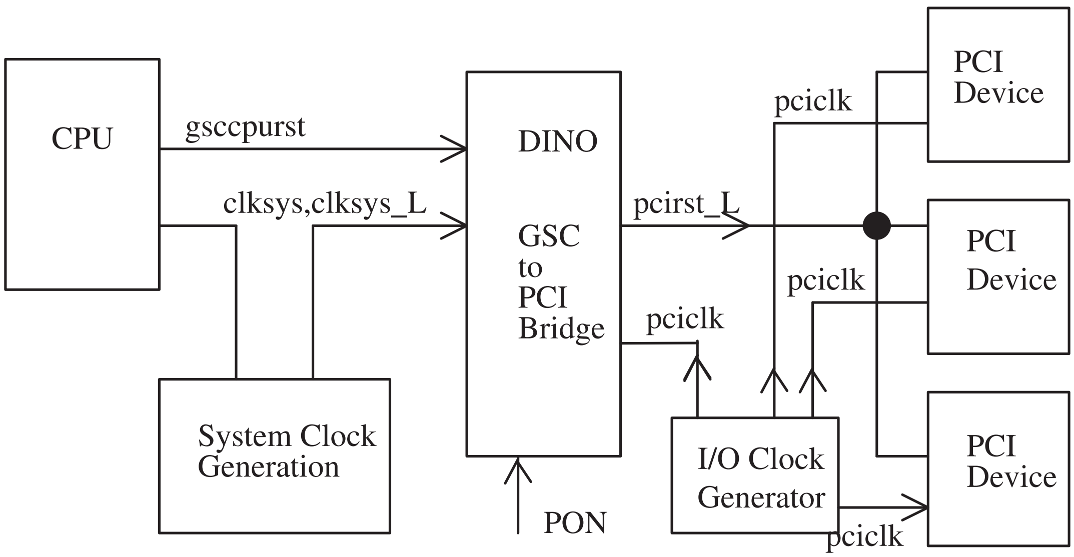


Figure 11. Dino Clocking and Resets

13.4 PCI and GSC Clock Relationship

PCI and GSC DO NOT require a fixed clock relationship under normal conditions. These two clocks can be generated by completely different oscillators. However, in the event that there is a bridge synchronization problem the clocks will be verified to operate with GSC and PCI running at the same frequency (33MHz). The relationship between the two clocks and reset for this contingency is shown in Figure 12.

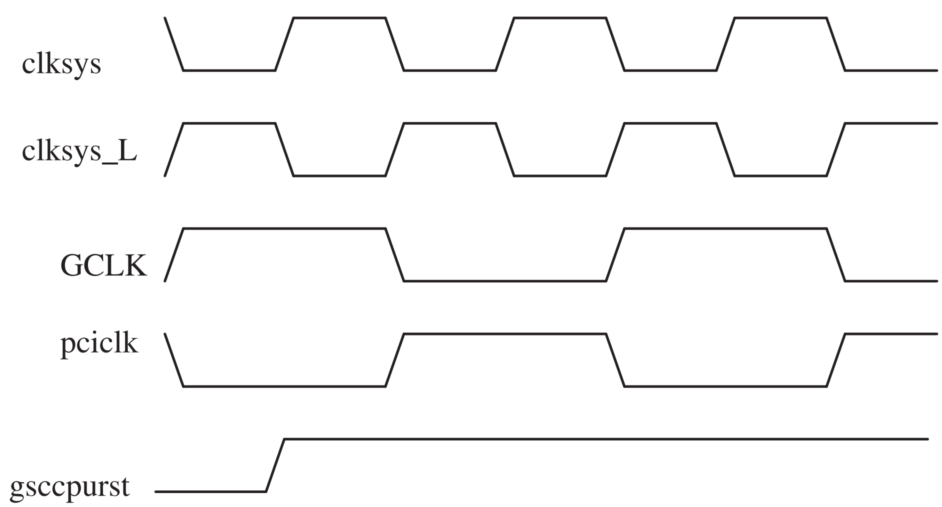


Figure 12. Possible PCI/GSC Clock Relationship

13.5 Broadcast Reset

Upon receiving broadcast reset Dino will reset itself and generate a PCI reset of the proper length. Software will need to wait 1.5ms after a broadcast reset before using PCI. As defined in the PA architecture broadcast commands do not generate a READYL on the GSC bus (just a reminder).

14 DINO–ON–A–CARD MODE

14.1 Introduction

HP plans to develop GSC add–in cards that use Dino along with PCI chip sets to allow PCI I/O functionality (such as 100BT LAN) to be added to legacy HP systems that do not have a PCI bus. Dino Revision 3.0 has a new mode, called card–mode, to support this effort. Dino in card–mode behaves identically to Dino in built–in bridge–mode with the following exceptions:

- The IODC header changes from reporting that Dino is a “Bus Bridge to a Foreign Bus” to reporting that Dino is a “Type A DMA Device.” (Needed to get firmware on legacy HP systems to boot.)
- IODC submodules 1, 2, and 3 are disabled. This means that RS–232 and PS2 functionality is not available in card–mode. (Needed to get firmware on legacy HP systems to boot.)
- The “gsc_sl[3:0]” signals are tristated after GSC RESETL is deasserted. No diagnostic information will be sent to the “gsc_sl[0:3]” signals. (Needed to prevent false GSC bus requests from occurring in some HP systems.)
- Added a new register, called “PCI_MEM_DATA,” to allow access to PCI memory space using only the bridge hard physical address (HPA) Registers in Dino’s IODC register space. (Needed since legacy firmware would not know how to allocate PCI memory addresses in PA I/O address space.)

14.2 How to Enable Card–Mode

The “rclk_dli” pin on Dino was redefined to the “brdg_mode” pin on Dino Revision 3.0. To put Dino in card–mode, the brdg_mode pin must be tied to 0. To put Dino into built–in bridge–mode, the brdg_mode pin must be connected to 1, or the “rclk_dlo” pin, or be a no–connect.

14.3 Setting up Dino for PCI Memory Devices in Card–Mode

PCI memory devices should be mapped to addresses in the 0xF000_0000 to 0xFFFF_FFFF range. The corresponding bits in the IO_ADDR_EN register should be set to tell Dino which addresses are owned by PCI vs. GSC. The “DPCIHIT” bit of the BRDG_FEAT register should be set to 1, so GSC transactions to Dino’s PCI memory addresses are not intercepted by Dino.

14.4 Accessing PCI Memory Space thru PCI_MEM_DATA

In card–mode, PCI memory space is accessed through two 32–bit HPA registers: PCI_CONFIG_ADDR (HPA offset 0x064) and PCI_MEM_DATA (HPA offset 0x070). Accessing the PCI memory Space is a two step process:

- Write of the target memory word address to the PCI_CONFIG_ADDR register.
- Perform a read or write to the PCI_MEM_DATA register.

The 32–bit PCI memory address is written to PCI_CONFIG_ADDR. The two LSBs of PCI_CONFIG_ADDR must be 2’b00. Sub–word or single–word GSC reads or writes to the PCI_MEM_DATA register are translated to the equivalent PCI memory transaction. (Note that this mechanism only allows single–word and sub–word accesses.) No byte swapping will occur on the PCI_CONFIG_ADDR register. The contents of the PCI_MEM_DATA register will be byte swapped between GSC and PCI.

15 RS–232 SERIAL INTERFACE

15.1 Introduction

The Serial interface emulates the National Semiconductor NS16550A device with some updates.

15.2 Feature Summary

This implementation includes all of the features of a standard NS16550A. See a commercial data sheet for detailed information. Virtually all of the interactions for the part are the same, so there is no need to rewrite already existing software. This implementation will have Receive and Transmit FIFOs each 16 bytes deep. Supported baud rates range from 50 to 454k. The hardware handshaking option will allow data input at up to 227k baud.

15.3 Register Definitions

The registers used for RS232 reside in HPA submodule 3.

Description	Offset	R/W	D7	D6	D5	D4	D3	D2	D1	D0
Reset Register	0x000	R/W	X	X	X	X	X	X	X	X
TEST	0x004	W	X	X	X	X	X	X	CLK_SEL	PAR_LOOP
IODC_ADDR	0x008	W								
IODC_DATA0	0x008	R	HVERSION							
IODC_DATA1	0x008	R	SVERSION							
Undefined	0x00c–0x05F									
RS232 clock freq. control (DITHER)	0x060	R/W	Data Bit 7	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0
Undefined	0x064–0x7FF									
Receiver Buffer Register (RBR)	0x800 DLAB = 0	R	Data Bit 7	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0
Transmitter Holding Register (THR)	0x800 DLAB = 0	W	Data Bit 7	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0
Interrupt Enable Register (IER)	0x801	R/W	0	0	0	0	Enable MSI	Enable LSI	Enable THREI	Enable RDAI
Interrupt Ident Register (IIR)	0x802	R	Fifos Enabled	Fifos Enabled	0	0	Int ID Bit 2	Int ID Bit 1	Int ID Bit 0	Int Not Pending

Description	Offset	R/W	D7	D6	D5	D4	D3	D2	D1	D0
Fifo Control Register (FCR)	0x802	W	Rx Trig MSB	Rx Trig LSB	X	X	DMA Mode	Tx Fifo Reset	Rx Fifo Reset	Fifo Enable
Line Control Register (LCR)	0x803	R/W	DLAB Bit	Set Break	Stick Parity	Even Parity	Parity Enable	Num of Stop Bits	Wrd Len Bit 1	Wrd Len Bit 0
Modem Control Register (MCR)	0x804	R/W	0	0	0	Loop Back	Unused	*See Note 1*	RTS	DTR
Line Status Register (LSR)	0x805	R	Error In Rx Fifo	Txmitter Empty	Tx Hold Reg Emp	Break Interrupt	Framing Error	Parity Error	Overrun Error	Rx Data Avail
Modem Status Register (MSR)	0x806	R/W	DCD (RLSD)	RI	DSR	CTS	Delta DCD	Trail Edge RI	Delta DSR	Delta CTS
Scratch Register (SCR)	0x807	R/W	Scratch Bit 7	Scratch Bit 6	Scratch Bit 5	Scratch Bit 4	Scratch Bit 3	Scratch Bit 2	Scratch Bit 1	Scratch Bit 0
Divisor Latch Reg LSB (DLL)	0x800 DLAB = 1	R/W	Divisor Bit 7	Divisor Bit 6	Divisor Bit 5	Divisor Bit 4	Divisor Bit 3	Divisor Bit 2	Divisor Bit 1	Divisor Bit 0
Divisor Latch Reg MSB (DLM)	0x801 DLAB = 1	R/W	Divisor Bit 15	Divisor Bit 14	Divisor Bit 13	Divisor Bit 12	Divisor Bit 11	Divisor Bit 10	Divisor Bit 9	Divisor Bit 8
Undefined	0x808–0xFFF									

Table 29. RS232 Register Definitions

Note 1—Bit 2 of the MCR is a read/write bit which is used as the control bit for the Hardware Handshaking functionality, where 1 implies normal RTS operation and 0 implies hardware protocol. Note that the reset state for bits in this register is 0, which needs to be overridden for normal RTS operation.

15.3.1 Detailed Register Descriptions

See The National NS16550A data sheet for a detailed description of all of the registers except the UART_RESET and UART_TEST registers. All registers except UART_RESET and UART_TEST are byte accessible only.

A cheat sheet for the Interrupt ID Register is shown below.

15.3.1.1 Interrupt ID Register

The upper nibble (bits 7–4) of the IIR register will return 0xC in fifo mode, and 0x0 in non–fifo mode.

The lower nibble (bits 3–0) will return the following interrupt IDs, in priority order.

First Priority	Receiver Line Status	0x6
Second Priority	Character Timeout Indication	0xC
Second Priority	Receiver Data Available	0x4
Third Priority	Transmitter Holding Register Empty	0x2
Fourth Priority	Modem Status	0x0
No Interrupt	No Interrupt	0x1

15.3.1.2 UART_RESET Register

Writing to the UART_RESET register will reset the Serial Interface. This resetting of the Serial Interface causes all of its registers except the divisor latches to return to their power–up state.

15.3.1.3 UART_TEST Register

Writing a 1 to the lsb of the UART_TEST register will set **par_loop**. **Par_loop** allows for parallel loopback testing of the UART FIFOs. This is primarily of benefit for chip level testing where the excessive time to perform serial loopback is a problem. If **par_loop** and **fifo_en** (FCR) are set, reading from the scratch register will actually read from the transmitter FIFO. Writing to the scratch register will actually write into the receiver FIFO. In other words, the transmitter FIFO would be tested by writing into the transmitter holding register and reading from the scratch register. The receiver FIFO would be tested by writing into the scratch register and reading from the receiver buffer register.

Writing a 1 to bit 1 of the UART_TEST register will select the GSC clock to generate the baud clock. This will result in a frequency of 7.2727Mhz. Writing a 0 to bit 1 of of the UART_TEST register will select the PCI clock to generate the baud clock, 7.407Mhz. Both methods are within 1.5% of the 7.3728Mhz used in Snakes such that the existing divisors will work fine. This bit needs to be a 0 to enable the DITHER clock generation feature.

15.3.1.4 IODC Registers

Writing all zero’s to this register will select IODC_DATA0 as the register that is going to be read. By writing 32’b0000_0004 to this register, IODC_DATA1 will be the register that is going to be read. The contents of IODC_DATA0 will be 0x0070_804A and the contents of IODC_DATA1 will be 0x0000_8C00.

15.3.1.5 DITHER Register

Some users of Dino expressed an interest in using the RS232 function in systems without a 40MHz GSC clock or 33.33MHz PCI clock. In this situation, the existing RS232 clock generation circuitry does not work. A new method was designed which can work with any reasonable GSC clock speed to produce a output frequency of:

CLK_RATE = GSC_FREQUENCY*DITHER/512. (ideal = 7.3728MHz)

If the DITHER register is 0x0 (the reset value), this feature is disabled. A ’0’ must be written to bit 1 of the UART_TEST register. This register is written to as a 32 bit word with the active bits in the LSByte of the word. A sample calculation example with a 30MHz GSC clock is:

DITHER = int(7.3728*512/30 + .5) = 126. The error compared to the 7.3728MHz is 0.14%, with +/– 33ns (i.e. one GSC clock period) of jitter.

15.4 Hardware Handshaking Control

15.4.1 Overview

The hardware handshaking implemented here is basically what has been done in the 375/380, and the various 700's since that time. The power up state is intended to be compatible with the Gecko implementation.

15.4.2 How to Enable

For hardware handshaking to be enabled, two bits in the **Modem Control Register** (**MCR**) need to be affected. Bit 2 is the hardware handshaking disable bit (normRTS), and should be set to a 0. This is its power up state. Note that the power up state for this bit in Snakes/Scorpio/PACE was 1. The power up state of 0 is consistent with the Gecko implementation. The RTS bit (bit 1) must also be set to a 1 for hardware handshaking to be enabled. It powers up cleared (0). A write of 0x02 to the MCR would be the correct way to enable hardware handshaking. To disable hardware handshaking, bit 2 should be set, and bit 1 should be cleared unless there is a need to keep $\overline{\text{RTS}}$ asserted for some other reason.

15.4.3 Hardware Gating

For those who want to know what the hardware is really doing to combine $\overline{\text{RXRDY}}$, $\overline{\text{RTS}}$, and normRTS, here's the scoop. $\overline{\text{RXRDY}}$ is an active low signal. It is low when the receiver has reached the trigger level. $\overline{\text{RTS}}$ is active low when used by the hardware outside of the register. For example, writing a 1 to the RTS bit to assert $\overline{\text{RTS}}$ will drive a logic 0 externally. The logic then looks like: !(normRTS | $\overline{\text{RXRDY}}$) | $\overline{\text{RTS}}$. These means that unless normRTS is cleared, $\overline{\text{RXRDY}}$ will be blocked, and unless $\overline{\text{RTS}}$ is asserted, $\overline{\text{RXRDY}}$ qualified by normRTS will be blocked. This is why normRTS must be cleared and RTS must be set to enable hardware handshaking. The final result when hardware handshaking is enabled is that $\overline{\text{RXRDY}}$ will appear inverted on the $\overline{\text{RTS}}$ line.

15.4.4 $\overline{\text{RXRDY}}$ Behavior

The National NS16550A data sheet gives a reasonable description of the operation of $\overline{\text{RXRDY}}$, but I will run through one example here. Let's assume we have set up a receiver trigger level of 8 bytes, and have chosen DMA mode 1. To do this, we would have written 0x89 to the **FCR** <u>after</u> having enabled the fifos. Initially, $\overline{\text{RXRDY}}$ will be deasserted, so with hardware handshaking enabled, $\overline{\text{RTS}}$ will be asserted. This tells the transmitting device that it is okay to send data. $\overline{\text{RXRDY}}$ will stay deasserted until the trigger level (8 bytes in this case) has been reached in the receive fifo. Once $\overline{\text{RXRDY}}$ asserts and hence $\overline{\text{RTS}}$ deasserts, $\overline{\text{RXRDY}}$ will stay asserted until the receive fifo is empty. In other words, the transmitting device will continue to be told not to send data until the receive fifo is completely empty.

15.4.5 Caveats

The biggest source of confusion has typically been the assumption that the prevention of hardware overflow eliminates the possibility of data overrun. Unfortunately this is not true. Memory buffers can also overflow, so in many cases, Software handshaking (XON/XOFF) is still required. It is my understanding that work is underway with the HP–UX driver to eliminate this shortcoming, so that $\overline{\text{RTS}}$ can temporarily be forced deasserted to halt the further transmission of data and hence prevent the overflow of Software buffers.

15.5 Software Differences/Clarification

The 16550 megacell is intended to function just like the real National NS16550A. This includes behavior that often seems rather stupid. The National NS16550A was chosen as the model over the WD16C552 because the National part came first, and the WD part is supposed to be compatible. There are a few minor differences between the NS16550A and the Stiletto megacell.

There is a bug in the Stiletto megacell that has not been fixed in Dino. The bug is that the receive data available interrupt indication does not get updated when the receiver FIFO is reset. What this means is that if a receive data available interrupt is indicated when the receiver FIFO is reset, the interrupt will continue to be indicated even though there is no data in the receiver FIFO. The current HP–UX workaround is to not reset the receiver FIFO.

Unlike the ASP serial implementation which uses the WD16C552 externally, there is no need for the software to worry about minimum cycle time requirements with the Stiletto megacell. The hardware will guarantee no violation of minimum cycle time specifications for register accesses.

The NS16550A data sheet never really says what happens if a **divisor** value of 0x0000 is loaded for baud rate generation. As it turns out, the NS16550A treats the ”0” as if a divisor value of 0x0020 were loaded. The Stiletto megacell treats it as if a 0x0001 were loaded, since this is the smallest meaningful value.

The NS16550A data sheet does not say if the **Modem Status Register** is writable, or what should happen if it is written to. The lower 4 bits (3:0) are indeed writable, and they will set or clear the modem ”delta” bits (and the interrupt if enabled) just like real changes in the modem control lines. The Stiletto megacell has duplicated this operation.

The table in the NS16550A data sheet indicates that bit 4 (loopback) of the **Modem Control Register** will always return a 0 on reads. This is not true. It will return the value that was written into this bit. The Stiletto megacell has duplicated this operation.

If the THRE (transmitter holding register empty) interrupt is currently enabled, and there is no THRE interrupt pending, and the THRE interrupt is disabled and re–enabled, the NS16550A will generate a new THRE interrupt. The Stiletto megacell has duplicated this operation. However, if the THRE interrupt is already enabled, and the **Interrupt Enable**

Register is written so as to keep the THRE interrupt enabled (no change on that bit), the NS16550A will cause a THRE interrupt. The Stiletto megacell will not.

If the transmitter fifo and the transmitter buffer are both empty, one would not expect changing to/from fifo mode to cause a THRE interrupt. The UART is transitioning from empty to empty, so there is no edge which should cause an interrupt. However, a THRE interrupt is generated whenever going to/from fifo mode. In addition, clearing the transmitter fifo via bit 2 of the **Fifo Control Register** also causes the THRE interrupt (regardless of whether it was already empty). The Stiletto megacell has duplicated this operation.

The **Line Status Register** is writable in the NS16550A for factory testing. The **Line Status Register** in the Stiletto megacell is not writable. With scan testing employed at the IC level, this was not necessary.

For the TEMT bit (transmitter empty) in the **Line Status Register**, it is never really defined when the transmitter shift register should be considered empty. The Stiletto megacell waits until all of the data bits AND all of the stop bits have been sent out before considering the transmitter empty. This is probably the desired behavior if this bit is being checked to see if it is okay to change baud rates. However, if in loopback mode and data is transmitted, receive data available will be indicated before TEMT, because only the detection of the first stop bit is required for receiving data. This isn't necessarily a problem, but it may be unexpected. I don't really know what the real NS16550A does.

The NS16550A clears the receive fifo registers when it reads from them. The Stiletto megacell does not. When the fifo empties, the NS16550A will always return 0's on reads. The Stiletto megacell will return the value that was previously in that fifo location. It seems unwise to rely on a fifo value when receive data available (line status) indicates it is an empty fifo location. I don't want to hear any whining about this.

The NS16550A data sheet says that a break is defined as received data being 0 for one full character time (start + data + parity + stop bits). However, it really calls something a break if received data is 0 for start + data + parity + 1/2 first stop bit. Basically, the NS16550A indicates all of the receiver line status interrupts (and receive data available) half way through the first stop bit. The Stiletto megacell has duplicated this operation.

The NS16550A says that on a framing error, it considers the 0 stop bit to really be a start bit, so it samples it twice and then starts looking for the next data bit. Who knows what they are trying to say. The operation of the NS16550A is non–deterministic in this case. Normally, the NS16550A will behave as if the 0 stop bit is a start bit, and it will take the next bit as data. However, sometimes it treats the next bit as a start bit rather than data. The Stiletto megacell always treats the next bit as data. Additionally, the Stiletto megacell does a 2 out of 3 sample on that stop bit anyway, so it shouldn't falsely be seen as a 0.

The NS16550A data sheet lies about the line status interrupts for the receiver and when they are cleared. It says that reading the **Line Status Register** is the only way to clear a break indication, an overrun error, a parity error, or a framing error. This is true in non–fifo mode, but in fifo mode, if the **Line Status Register** is not read, but the receive data is read, the **Line Status Register** will be updated with the info from the next fifo location. If the next location doesn't have any errors, the receiver line status errors have effectively been cleared, while never having read the **Line Status Register**. The Stiletto megacell has duplicated this operation.

From looking at the ordering of the interrupts in the table for the **Interrupt ID Register** in the NS16550A data sheet, it would appear that a receiver data available interrupt would have priority over a character timeout interrupt. Both are considered ”second” priority, but the receiver data available interrupt is listed first. In reality, the character timeout interrupt has priority. With the fifo trigger level set at 1 byte, an interrupt caused by receiver data being available will turn into an indication of a character timeout interrupt if the receiver is not read soon enough.

16 PS2 INTERFACE FOR KEYBOARD/MOUSE

16.1 Introduction

Dino supports two PS2 ports. The registers used for PS2 reside in HPA submodule 1. This submodule provides 1 page (4KBytes) of register space so that the register offset of the PS2 support registers will exactly match the requirements of existing drivers. Dino implements the keyboard and mouse interfaces as simple serial ports conforming to the de facto industry standard PS/2 specification. Each user input device has a dedicated serial port of its own. The interface ports rely on the software to provide all of their intelligence, therefore, they do not interpret the characters passing through them in either direction. The interface to the host processor is through 6 one–byte registers for each port.

16.2 Registers

Register Label	Register Name	Address offset (word aligned)	Reset Value	Access
IODC_ADDR	IODC Adress	0x008	*note 1*	R
IODC_DATA 0	IODC HVERSION	0x008	*note 1*	R
IODC_DATA 1	IODC SVERSION	0x008	*note 1*	R
ID	Keyboard ID Register	0x800	*note 1*	R
RESET	Keyboard Interface reset register	0x800	0xXX	W
RCVDATA	Keyboard Received data register	0x804	0xXX	R
XMTDATA	Keyboard Transmit data register	0x804	0xXX	W
CONTROL	Keyboard Control register—read/write	0x808	0x00 *note 2*	R/W
STATUS	Keyboard Status register—read only	0x80c	0x00	R
ID	Mouse ID Register	0x900	*note 1*	R
RESET	Mouse Interface reset register	0x900	0xXX	W
RCVDATA	Mouse Received data register	0x904	0xXX	R
XMTDATA	Mouse Transmit data register	0x904	0xXX	W

CONTROL	Mouse Control register—read/ write	0x908	0x00 *note 2*	R/W
STATUS	Mouse Status register—read only	0x90c	0x00	R

Note 1: Each PS2 device returns a unique hardwired ID code in bits 3:0 of ID.
Note 2: Resetting the block disables it (see Table 26).

To address the Keyboard registers AD[11:8] = 4’b1xx0. To address the Mouse registers AD[11:8] = 4’b1xx1.

Table 30. PS2 Interface Registers

16.3 IODC Registers

Writing all zero’s to this register will select IODC_DATA0 as the regiseter that is going to be read. By writing 32’b0000_0004 to this register, IODC_DATA1 will be the register that is going to be read. The contents of IODC_DATA0 will be 0x0070_804A and the contents of IODC_DATA1 will be 0x0000_9600.

16.4 ID Register

Bit	Symbol	Name	Description
3:0	ID	ID Code	Hardwired physical identification bits. (0=key-board;1=mouse)
7:4	Reserved		

Table 31. PS2 ID Register

16.5 Reset Register

Any write to this register will cause the interface to be reset: all buffers will be emptied and the receive/transmit state machines will be reset. The value of the data written will be discarded. Note that this does not cause the external device to be reset; it must be reset explicitly through a command sent from the host. A reset will leave the interface in the disabled state.

16.6 Rcvdata Register

This register provides access to the received data buffer in the interface. Each read operation from this register removes one character from the received data buffer. Receive Buffer Not Empty (bit 0 of the STATUS register) is set to 1 whenever there is one or more characters in the receiver buffer and returns to 0 when the buffer is empty. The port asserts its interrupt line when the receive buffer goes from empty to not empty. It is the responsibility of the host to read the rcvdata register until the Receive Buffer Not Empty bit returns to a 0 value. Note that a port does not generate an interrupt for each received character, only when the first character is placed in an empty buffer. Dino implements the Received Data Buffer with 4 characters of storage.

16.7 Xmtdata Register

This register provides data to the transmitter section of the interface. Since each character transmitted requires an acknowledge character, the transmit buffer is only one character deep. The PS2 protocol allows for outgoing data to interrupt and override any incoming data. Dino will receive any incoming character before starting the transmit process. The transmitter will assert control as soon as a received character is finished and send the character in the transmit buffer. Transmit Buffer Not Empty (bit 1 of STATUS register) is set to 1 when there is a character in the transmit buffer. It is the responsibility of the host to determine when the buffer is empty by monitoring TBNE, which will go to 0 when the buffer is empty. A PS2 port will ignore an attempt to write to the Xmtdata register while TBNE is 1. No interrupts are generated in the transmit process, except for acknowledge characters returned by the external device.

16.8 Control Register (R/W)

Bit	Symbol	Name	Description
0	ENBL	Enable	Set status of interface: 0 = disabled, 1 = enabled. When disabled, port will neither receive data from external device nor generate interrupts to host.
1	LPBXR	Loopback Xmt/Rcv mode	Set to 1 for loopback diagnostic mode: transmitter output is connected to receiver input. In this case, a much faster internal clock is used to transfer the data (2 MHz).
4:2	Reserved		
5	DIAG	Diagnostic mode	When set to 0, bits 6 and 7 have no effect on the external Data and Clock lines. When set to 1, bits 6 and 7 directly control the value of the external Data and Clock lines for diagnostic purposes.
6	DATDIR	External data line direct control	Provides direct control of value of external Data line while in diagnostic mode.
7	CLKDIR	External clock line direct control	Provides direct control of value of external Clock line while in diagnostic mode.

Table 32. PS2 Control Register

16.9 Status Register (Read only)

Bit	Symbol	Name	Description
0	RBNE	Receive buffer not empty	0 = receive buffer empty, 1 = receive buffer not empty
1	TBNE	Transmit buffer not empty	0 = transmit buffer empty, 1 = transmit buffer not empty
2	TERR	Timeout Error	Normally set to 0. See Note 2
3	PERR	Parity Error	Normally set to 0. See Note 1
4	CMPINTR	Composite interrupt	OR of interrupt lines of all PS2 ports
5	Reserved		
6	DATSHD	Data line shadow	Copy of current value of external data line
7	CLKSHD	Clock line shadow	Copy of current value of external clock line

Table 33. PS2 Status Register

Note 1:
When the receiver detects a parity error, bit 2 of STATUS register is set to 1. The incorrect character is placed in the receive buffer. Busy is asserted to the external device to halt any further transmissions. The host must recognize the parity bit and empty the receive buffer since there may be valid characters in the buffer ahead of the incorrect one. The last character in the buffer is always the incorrect one. In order to clear the parity error, the interface must be reset. It is the responsibility of the host to request a resend from the external device if desired.

16.10 Addressing

One 4Kbyte block of Gecko IO space is allocated for both PS2 interfaces. Each interface is assigned a 256 byte block of its own. The keyboard port is located a offset 0x000 within the 4K block and the mouse is located at offset 0x100. There is no logical difference between the keyboard and mouse interface other than their address and the contents of the ID register. All registers are 1–byte wide and are aligned on word boundaries. All addresses are multiply–mapped so that a read/write to any address in the 4k block will access a legitimate register.

16.11 Interrupt processing

The Dino interrupt handler supports only one interrupt line for both PS2 ports. The host must poll both ports to determine which ones have data. Each port asserts its interrupt line until its buffer is empty. Both of the interrupt lines are OR'd together to make a single interrupt signal. An interrupt will be issued when it sees a positive edge on this signal. Thus there is an interrupt for the first character to arrive at either port, but none for following characters until both of the ports have been emptied. The host must cycle through both ports, emptying the data from each until both are empty, including data which arrives during the service process. The host determines that both ports are empty by the Composite Interrupt signal (*cmp_intr*), which is copied into bit 4 of the STATUS register of both ports. Since the host must continue to read data from ports until *cmp_intr* is 0, the next arriving character forces *cmp_intr* to 1, which causes an interrupt. It is anticipated that in most cases each arriving character will cause an interrupt and will be serviced before the next character arrives.

16.12 Timing

The PS2 specification maximum transfer rate is about 1 character per millisecond (80 microseconds per serial bit for 11 bits plus some overhead). The fastest input expected from a keyboard would be about 16 characters per second. If the keyboard generates 3 characters per keypress (character code on downstroke, up code on release, followed by character code again), then we can expect characters about every 20 ms. A mouse is programmable for sample spacing (20 to 200 samples per second, default 100) and sends 3 characters for each XY sample. Thus a mouse might push close to the 1 ms character spacing. There should be no problems responding to interrupts at this rate.