UTURN ERS

REVIEW COPY

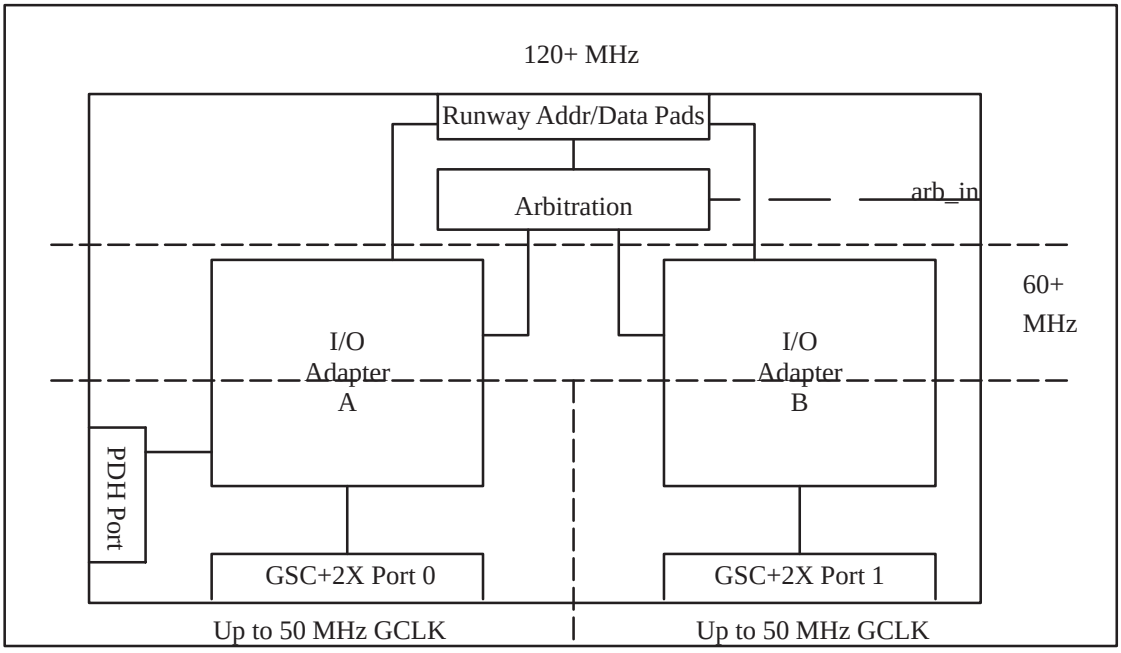
Version 0.95

May 17, 1996

Systems Technology Division
General Systems Lab

1. Introduction

The UTurn I/O Subsystem Interface provides a bus converter path between the Runway (processor–memory) bus and two GSC+2X busses. UTurn will be implemented in CMOS26G and will be packaged in a 432 pin CPGA. The following diagram illustrates how both GSC+2X interfaces are supported within a single UTurn:



Note that there can be up to four separate frequencies on the chip. The Runway Arbitration block and all Runway Pads will run at a bus speed of 120+ MHz. The Runway interface block within each I/O Adapter will run at one–half of this external Runway frequency. Each GSC+2X interface has an independent frequency of up to 50 MHz (GCLK). In GSC 2X data transfer mode, write data mastered by UTurn can be transferred at a peak rate of as high as 100 MWords/sec.

This document describes the functional design of UTurn, including architectural requirements, overall I/O performance, clock–domain specific operation, testability, and packaging. The remainder of this chapter is simply an introduction.

The UTurn design is heavily leveraged from the U2 design. UTurn adds GSC+2X capability, corrects a few minor U2 anomalies, improves DMA read performance, and incorporates internal and external timing enhancements for higher frequency of operation and improved design margins.

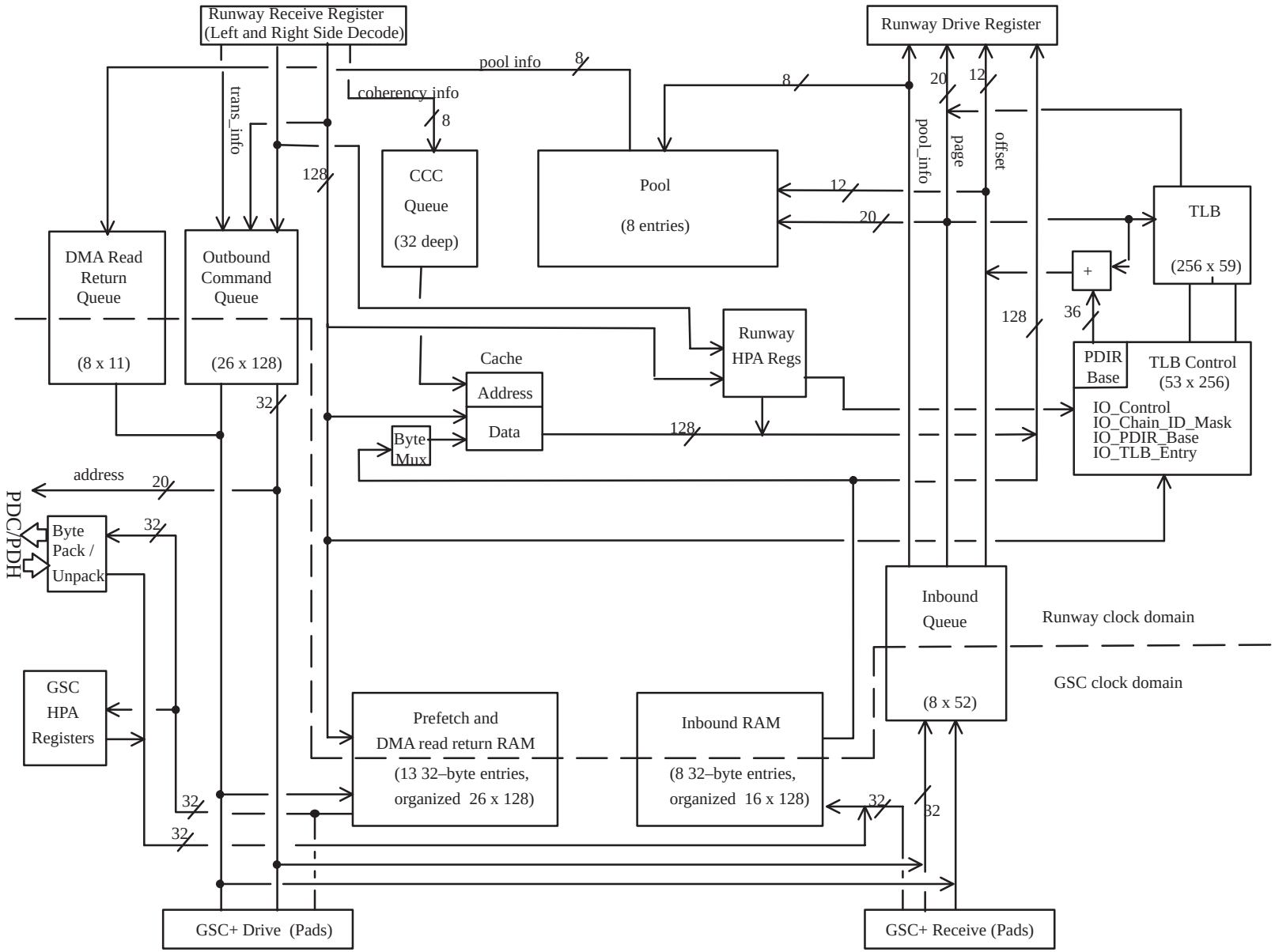
1.1. Functional Operation

On the following page is a block diagram of a single I/O Adapter within a UTurn. As transactions flow from Runway to GSC+, transaction information (such as address, operation, transaction identification, size, and status) is stored in order in either the outbound command queue (OutQ) or the DMA read return queue (RRQ). Corresponding data is stored either in the OutQ or in a designated DMA read return RAM (RRR) location. The GSC+ interface block extracts transactions in order from the two queues unless a connected GSC+ read has been issued. In this case the return data must be advanced in front of the queues to avoid deadlock on GSC+. There is a high water mark near the tail of the OutQ. If the number of OutQ entries exceeds this high water mark limit, the I/O Adapter will limit further CPU and memory mastered transactions to I/O address space (with the exception of DMA read returns, which utilize the DMA read return queue). This prevents overflow of the OutQ.

Transactions going from GSC+ to Runway are placed in the inbound queue (InQ) in order. Associated data is stored in a corresponding inbound RAM (InRAM) location. Once at the head of the InQ, the physical page map (I/O TLB) is accessed for the appropriate address mapping. If there is a miss on the I/O TLB access, the I/O Adapter may be configured to access the I/O Page Directory (I/O PDIR) in main memory to obtain the address translation, updating the I/O TLB with this new mapping. Now the original request can be issued over Runway with the new translation.

The Runway pool and prefetch buffers store transaction information for outstanding I/O Adapter mastered Runway reads. When memory data corresponding to an I/O Adapter mastered read is returned over Runway, the data is stored in the read return RAM (RRR) and the transaction information is obtained from the pool or prefetch buffer and placed in the DMA read return queue (RRQ). The I/O Adapter masters prefetches from memory over Runway when indicated by a GSC+ guest via the XQL hint. The corresponding transaction information is stored in the prefetch buffer until the GSC+ guest explicitly requests the data, at which point the transaction information is pooled (if data has not yet been received) or queued in the RRQ (if data is valid on chip).

The cache is required for performing partial cache line writes (of less than 16 bytes) and semaphores, where private ownership of a cache line is required. The existence of a cache requires the I/O Adapter to participate in the Runway snoopy coherence protocol. Coherent transactions mastered by other Runway modules (processors or I/O Adapters) require snooping. These transactions are stored in the cache coherency check (CCC) queue and are responded to in order. The I/O Adapter cache is only one line deep and in most cases will not contain a valid entry, so the I/O Adapter is expected to respond very quickly to outstanding coherent requests. The I/O Adapter only needs to queue coherent transactions when it holds a cache line privately.


Runway Receive Register
(Left and Right Side Decode)
Runway Drive Register
pool info
8
coherency info
8
trans_info
128
CCC
Queue
(32 deep)
Pool
(8 entries)
8
20
12
pool_info
page
offset
12
20
TLB
(256 x 59)
+
36
128
DMA Read
Return
Queue
(8 x 11)
Outbound
Command
Queue
(26 x 128)
Runway
HPA Regs
Cache
Address
Data
32
128
Byte
Mux
PDIR
Base
TLB Control
(53 x 256)
IO_Control
IO_Chain_ID_Mask
IO_PDIR_Base
IO_TLB_Entry
address
20
PDC/PDH
Byte
Pack /
Unpack
32
Inbound
Queue
Runway clock domain
GSC clock domain
(8 x 52)
GSC
HPA
Registers
Prefetch and
DMA read return RAM
(13 32–byte entries,
organized 26 x 128)
Inbound RAM
(8 32–byte entries,
organized 16 x 128)
32
32
32
32
GSC+ Drive (Pads)
GSC+ Receive (Pads)

1.2. Feature Set

Each of UTurn's I/O Adapters include several features to optimize performance. These are described below.

Hardware Coherency is featured in the I/O Adapter to optimize performance by eliminating both the memory utilization required for partial cache line writes and the software overhead required to perform dflushes and dpurges prior to initiating or after completing a DMA. On the I/O Adapter, hardware I/O coherency requires an I/O TLB for performing physical to virtual address translation, a cache RAM location for containing private copies of data, and a CCC queue to implement snoopy bus protocol. The I/O TLB will be merged with the physical page map which is required for mapping the 32 bit I/O address space to the expanded PA and Runway address space, for future upgradability (such as PCX–U).

Each I/O TLB entry has bits indicating page type, which are manipulated by software drivers, to impose certain hardware behavior when accessing a given page. UTurn will perform inbound full cache line DMA using Runway WRITE_PURGEs. For half cache line DMA writes to fast DMA pages, Runway WRITE_16_PURGEs are used. For writes of less than half a cache line, or for half cache line writes to safe DMA pages, the I/O Adapter will issue Runwy READ_PRIVATEs, followed by modifies on chip, followed by WRITEBACKs to memory.

Processors could implement a special uniprocessor mode bit for maximizing performance while coexisting with the I/O Adapter. In this mode, software drivers will maintain coherency by executing the appropriate flushes and purges. A processor is not required to broadcast flushes and purges or to eavesdrop on WRITE(16)_PURGEs and READ_SHAR_OR_PRIVs. However, the I/O Adapter will operate identically in this mode issuing coherent transactions such as private reads for semaphores or partial cache line writes, and the processor is expected to eavesdrop and respond to these.

Buffering is provided in each I/O Adapter for all inbound and outbound writes and for pended DMA read requests. The depth of the inbound queue and the corresponding DMA read return queue is currently eight entries. The depth of the outbound command queue is twenty–six entries, providing adequate graphics word write performance and minimizing the likelihood of filling the OutQ, which results in the issuance of STOP_IO on Runway for flow control.

Memory Prefetch is implemented based on two levels of hints. GSC+ devices may issue a DMA read prefetch hint by asserting the GSC+ XQL line, indicating that the next address will be accessed. Each entry in the I/O TLB contains a prefetch enable bit indicating whether prefetech accesses to a given page are permitted. Both the TLB prefetch enable bit and the GSC+ XQL signal will be evaluated by the I/O Adapter to determine whether a prefetch is issued.

Fully Asynchronous Synchronization is implemented in the I/O Adapter using synchronizing FIFOs, which are deep enough to compensate for handshake synchronization delays. The synchronization circuitry has been used on several HP VLSI products and is well tested. The I/O Adapter is designed to work with a Runway:GSC frequency ratios between 2:1 and 4:1 (Runway ckrw: GSC GCLK).

Processor Dependent Hardware is supported through a separate physical port on one of UTurn's I/O Adapters. Because PDC must be accessible at all times (especially during errors), special precautions are taken to ensure that this path is always available.

GSC1.5X and GSC2X capability is supported for DIO writes (mastered by UTurn), enabling breakthrough data transfer performance for writes from processors to I/O devices. This capability is primarily utilized to enhance graphics performance.

1.3. Seldom Used UTurn Features

The following list of UTurn features are seldom used but may come in handy in certain system applications:

1. Arbitration support for two UTurns.
2. Special CMD_Reset interpretation for GSC+: When CMD_Reset is issued by a processor to one of UTurn's IOAs, RESETL will be asserted on the corresponding GSC bus if an internal configuration bit is set. This feature is intended for use in manufacturing test.
3. Performance counters and programmable testability features for the purposes of monitoring UTurn and the system.

1.4. Sizing of UTurn Internal Structures

The original GSC specification and the GSC+ and GSC+2X extensions place limits on the number of outstanding DMA read transactions possible on a given GSC bus. These GSC limits determine the size of some of UTurn's internal structures required to avoid deadlock. The following explains the GSC rules and how they affect the sizes of UTurn internal structures. Sizing of internal structures also has performance implications, which are addressed briefly in the performance chapter of this ERS.

1.4.1. GSC and GSC+ Limitations

The maximum number of GSC guests allowed per bus is determined by the frequency goals, electrical loading, and UTurn pin count limits. For UTurn, each GSC bus is limited to a maximum of six bus–mastering guests (any mix of GSC and GSC+ guests).

The GSC specification defines only connected reads, so only one read can be in progress at a time. The GSC+ extensions allow pended reads, increasing the maximum number of DMA reads simultaneously in progress to one per guest. Since the GSC+ extensions do NOT include a transaction ID capability, and ordering of DMA read responses from memory is not guaranteed, GSC+ can not support more than one outstanding DMA read per guest. Thus, assuming a GSC bus fully loaded with GSC+ guests, there can be at most six formally–requested DMA reads outstanding at any one time. These formally–requested reads do not include prefetches.

The GSC+ extension for prefetching allows each guest to have one prefetch read outstanding in addition to one formally–requested read. Assuming a GSC+ bus with six guests all prefetching, there can be a maximum of six prefetch reads outstanding to memory at any one time.

1.4.2. UTurn Limitations

Due to Tornado limitations on the number of transaction ID bits implemented on Runway, each IOA is limited to having only eight outstanding read transactions at any one time. Internally, each transaction ID is assigned to a pool location. Thus, the limit of eight transaction IDs allows the pool size to be limited to eight entries per IOA. This also means that a maximum of eight reads per IOA (16 per UTurn) may issued on Runway (with no data returned) at any one time.

1.4.3. Sizing of UTurn Internal Structures

Since the depths of the internal queues and pool in each of UTurn's IOAs are configurable, anyone altering the default sizes should be aware (and beware) of the following rules regarding structure sizes. When

reducing the size of the queues or pool(s), one should make a corresponding reduction in the number of GSC+ guests per bus to avoid deadlocks and overruns.

Read Return RAM size rule: for all outstanding DMA reads (reads with data back as well as reads in the pool waiting for data), each of UTurn's IOAs must have a place to store the data. This implies that the size of the Read Return RAM (RRR) in each IOA must be greater than or equal to the TOTAL number of outstanding GSC reads from that IOA (formally–requested + prefetch = 2 x number of GSC guests).

- size of RRR >= 2 x number of GSC guests

For UTurn, which supports a maximum of six GSC guests, RRR = 12 entries. Each entry is one cache line (256 bits) wide.

Pool size rule: for all outstanding DMA reads on Runway (including prefetch reads), each of UTurn's IOAs must have a place to store the transaction information for that IOA's outstanding reads (GSC guest ID and length of read requested). This implies that the size of the pool must be greater than or equal to the TOTAL number of outstanding GSC reads (formally–requested + prefetch = 2 x number of GSC guests). The size of pool can thus be computed as follows:

- size of (RRQ + pool) >= (2 x number of GSC guests).

UTurn has a implemented 8 physical pool entries, since only 8 unique transaction IDs are supported per IOA. It is thus possible with 5 or 6 guests to have more requests queued inbound than can be accommodated by the pool (if prefetches are included). UTurn's Runway inbound logic ensures that only 8 reads can be outstanding on Runway at a time, preventing pool overflow.

Read return queue size: there must be enough RRQ entries to accommodate all explicitly requested pended reads on GSC. Since each GSC guest is limited to one pended read request at a time, the RRQ must contain 6 entries.

Extreme conditions: Under some conditions, all formally–requested GSC reads (one per guest) could be in the pool, where all reads have been issued on Runway and no data has been returned. Similarly, data returns for all GSC–issued pended reads could be in the RRQ (all reads have had data returned and the data needs to be returned to the guests). UTurn's IOAs accommodate these two allowable (but unlikely) extreme conditions.

1.5. Transaction Map

In the following table, transactions on the left side are mastered by a Runway module, and destined for the I/O Adapter. All transactions require a minimum amount of activity on receipt, such as address and transaction information being placed in a queue and data being placed in the RAM. Subsequent activity is required when the transaction reaches the head of the queue, and the right side of this table reflects that activity. The first four transactions in the table have transaction type encodings on Runway. The last transactions are read responses (memory data returns or cache to cache copies) resulting from a previous I/O Adapter mastered read request.

Runway Transaction	I/O Adapter Activity

write_short	if (address==IOA HPA) write data internally
	if (address==GSC+) issue GSC+ write1, write2, 2 write1s, or writeV
	if (address==PDH) issue PDH write

read_short	if (address== IOA HPA) read internally
	if (address==GSC+) issue GSC+ read1 or read2
	if (address==PDH) issue PDH read

broadcast_error	log as hard error
directed_error	log as hard error
ttype[2]: any coherent transaction	coherency response
data read return (trans_id==connected)	store data in RAM, discard pool entry, form entry in reserved head of read return queue, and drive connected GSC+ return
data read return (trans_id==prefetch)	store data in RAM, note data returned, and wait for GSC+ to issue address
data read return (trans_id==pool)	store data in RAM, discard pool entry, and form read return queue entry for the tail of the read return queue
data read return (trans_id==cache)	this is a partial cache line write, so store data in cache and modify, then issue write_back to memory and discard pool entry
data read return (trans_id==tlb)	store data in the appropriate TLB location and discard pool entry

Note that a prefetch entry converts to a standard pool entry when the GSC+ guest formally requests data from the prefetch address while the read return is still outstanding. A pool entry can indicate both connected and cache, meaning that the return data is stored both in the cache and driven over GSC+ (for a GSC+ clear transaction).

The next table shows GSC+ guest mastered transactions in the left column and corresponding I/O Adapter activity in the right column. On inbound traffic, all transactions require a minimum amount of activity, such as placing the request in the queue, obtaining a virtual index and expanded physical address from the I/O TLB and Physical Page Map, and checking to see if the data wasn't already prefetched or contained in the on chip cache (in the case of a semaphore). Once it is determined that the request at the head of the inbound queue must be issued over Runway, the I/O Adapter operates on the transaction and the right column in this table reflects that activity.

GSC+ Transaction	I/O Adapter Activity
GSC+ connected read (GSC+ LSL asserted)	issue Runway read_priv return data gets stored in cache and returned to GSC+ guest
GSC+ reads (no assertion of GSC+ LSL)	issue Runway read_shar return data gets queued for GSC+
GSC+ write 1,2,4 (safe page) (No LSL) OR GSC+ write 1,2,4 with LSL asserted	check cache to see if read_priv previously issued: if cache hit: modify in cache and issue Runway write_back if cache miss: issue read_priv, modify, and write_back
GSC+ write 4 words (half cache line) (fast page) (No LSL)	issue Runway write16_purge
GSC+ write 8 with LSL asserted	check cache to see if read_priv previously issued: if cache hit: modify in cache and issue Runway write_back if cache miss: issue write_purge
GSC+ write 8 words (cache line) (fast or safe page) (No LSL)	issue Runway write_purge
GSC+ write1 to I/O address	issue Runway write_short to ”F”–extended I/O address
GSC+ non–write1 to I/O address	signal and log GSC error
GSC+ data read return	issue Runway read return transaction
GSC+ clear transaction	issue Runway read_private; when data is back, return data to GSC and store data in cache, clear first word pointed to by address, and write_back data to memory
GSC+ error transaction	log hard error on GSC side
GSC+ reads that match previous prefetch (previous transaction from GSC+ guest was a read with XQL asserted and this read matches a pool entry)	Runway read_shar_or_priv has already been issued. Mark up pool entry to indicate that the GSC+ guest has ”formally” requested the data. Update pool with formal transaction information.

Note that the GSC+ XQL line may be indicated and prefetch issued on any sequential 16 and 32 byte reads. Likewise, the GSC+ LSL line may be indicated on any of the above transactions, resulting in the assertion of STOP_MOST on Runway. The following table entries reflect these orthogonal behaviors:

GSC+ XQL asserted GSC+ read half or full cache line (prefetch enable from TLB)	issue Runway read_shar_or_priv and place read entry into the pool; increment address by transaction size and issue subsequent Runway read_shar_or_priv and place prefetch entry in the pool
GSC+ LSL asserted	issue STOP_MOST
GSC+ transaction with no LSL asserted	deassert STOP_MOST and issue transaction on Runway release any cache line owned by this IOA
Data return (from GSC+, PDH or HPA reg)	deassert STOP_MOST and issue return on Runway release any cache line owned by this IOA
GSC+ transaction on a no lock page	deassert STOP_MOST and issue transaction on Runway
GSC+ LSL deasserted on GSC+	deassert STOP_MOST release any cache line owned by this IOA

2. Architectural Requirements

This chapter documents the architectural features of each I/O Adapter within a single UTurn. Specific topics discussed include the register set layouts, register definitions, address space mapping, architected I/O writes, and error strategy.

When describing registers and their contents, is is often necessary to identify the specific value of a field. To differentiate between hex, decimal, and binary values, the following syntax is used:

n'b	indicates that the following n digits are in binary
n'd	indicates that the following n digits are in decimal
n'h	indicates that the following n digits are in hexadecimal
X'	indicates that the following digits are in hexadecimal
x'	indicates that the following digits are in hexadecimal

Many registers contain undefined fields that are reserved for future definition. Often, the single letter R is used to describe these reserved fields. When read, the bits in a reserved field are zeroes. Writes to reserved fields have no effect.

The reader should beware that there are some ordering concerns on processor–mastered I/O reads and writes of Runway clock domain registers in UTurn. Runway does not define slave acknowledgement for write transactions. Therefore, there is no way for UTurn to hold off subsequent accesses until an I/O write completes internally. Additionally, writes to UTurn internal registers that reside in the Runway clock domain will take affect within 3 Runway cycles, whereas reads of these Runway registers are queued, passing through the OutQ and back through the InQ prior to actual extraction of a read value. Neither reads nor writes will bypass writes, but writes of Runway registers can bypass reads of Runway registers. It is the responsibility of software to follow UTurn–internal Runway I/O register writes with a UTurn I/O register read (such as IO_STATUS). Upon receipt of the read response, software can continue with the confidence that its write has completed. This "read sweep" must occur on a per–processor basis, since the relationship of I/O reads and writes mastered by different processors is in general unknown. There are no ordering concerns for accesses to GSC clock domain register, because all accesses to these registers are queued in the OutQ and processed in the queued order.

2.1. Runway Hard Physical Address (HPA) Space

The Runway HPA space consists of three register sets: the Runway Supervisory Register Set, the Runway Auxiliary Register Set, and the Runway Bus Specific Register Set. Runway is allocated 256 KBytes of HPA space within I/O address space. As a Runway module, UTurn responds to some of the addresses in this region. For KittyHawk, the Runway HPA region is in the address range from X'FFFFF80000 to X'FFFFFBFFFF. Hard physical 40–bit addresses to architected I/O registers take the following form:

0 3	4 11	12 21	22 24	25	26	27	28 33	34 37	38 39
1111	extend	flex	001	UTurn	ioa	0	regset	offset	00

where:

1111	indicates an I/O address.
extend	allows for extended I/O address space. (On UTurn, this field is tied to 8'b11111111.)
flex	indicates bus to which UTurn is connected. (For Runway, the flex field is hardwired to 10'b1111111110.)
UTurn	indicates UTurn's location on the bus. It is set to Runway CLIENT_ID [2].

ioa indicates which of two internal IOAs within the selected UTurn; corresponds to Runway CLIENT_ID[3]. Together with the above UTurn client encoding, this field allows for a distinct IOA address space corresponding to a client_id value of 4, 5, 6, or 7. (Bits 22 through 27 define the fixed field.)

regset indicates the register set within the HPA. The register sets are as follows:

Supervisory Register Set:	0
Auxiliary Register Set:	1
UTurn Specific Register Set:	17 decimal

offset indicates the address offset of a given register. Registers are described in more detail in the following sections.

Note that the field specified by bits [22:27] corresponds to the fixed field (as defined by the PA–RISC I/O ACD).

2.1.1. Runway Supervisory Register Set (Register Set 0)

The following registers comprise the Runway Supervisory Register Set:

Offset	Access Modes	Class	Name	Address LSB
2	R	A	IO_DC_DATA	.. 008
2	W	A	IO_DC_ADDRESS	.. 008
12	W	A	IO_COMMAND	.. 030
13	R	A	IO_STATUS	.. 034
14	RW	A	IO_CONTROL	.. 038

2.1.1.1. Runway IO_DC_DATA Register

The IO_DC_DATA register are used to read an IOA's IODC. The architecturally specified locations of IODC are defined as follows:

Byte Address	Name	Description
0 – 1	IODC_HVERSION	Hardware version number
2	IODC_SPA	Soft physical address capability
3	IODC_TYPE	Type of module
4 – 7	IODC_SVERSION	Software version number

The contents of these bytes are hardwired, as follows:

0 4	5 11	12 15	16	17	18	19 23	24	25	26	27 31
bus=01011	hvmod=0000000	hv_rev	io=0	0	sv=0	shift=8	mr=0	wd=1	R	type=01100

32 35	36 55	56	57	58	59	60 63
sv_rev	svmod = 5'h0000B	R	sv.opt=1	R	mc=1	R

where	bus	is set to an encoded value to indicate Runway. This value is 5'b01011.
	hvmod	specifies the module hardware implementation. This value is 7'b0000000.
	hv_rev	indicates hardware version. For UTurn rev 1.3 (1MM6–0001) and rev 1.4 (1MM6–0002), this value is 4'b1111.
	io	set to 0, indicating that soft physical address space is in memory address space.
	0	is simply hardwired to 0.
	sv	indicates if the following shift field is valid. This value is set to 0.
	shift	specifies the maximum TLB space size in entries. This value is set to 5'b01000. (This field is emulated by PDC.)
	mr	more bit is 0 to indicate that no more than the first 8 bytes of IODC are implemented.
	wd	word bit is 1 to indicate that a full word of address justified data is provided on reads from IO_DC_DATA.
	R	reserved. Reserved fields are always 0 when read.
	type	specifies the module type. This value is set to 5'b01100 to indicate TP_IOA.
	sv_rev	indicates feature enhancements. For UTurn, this value is 4'b0001.
	svmod	specifies software interface to module. This value is set to 5'h0000B, indicating that the upper port is a coherent I/O bus converter port..
	R	reserved. Reserved fields are always 0 when read.
	sv.opt	set to 1, indicating that UTurn implements greater than 32 bits of physical address.
	R	reserved. Reserved fields are always 0 when read.
	mc	module category. This value is set to 1.
	R	reserved. Reserved fields are always 0 when read.

Because a write to the IO_DC_ADDRESS register determines which IO_DC_DATA word is returned, software must ensure that at least 2 Runway cycles transpire between the IO_DC_ADDRESS write and the IO_DC_DATA read.

2.1.1.2. Runway IO_DC_ADDRESS Register

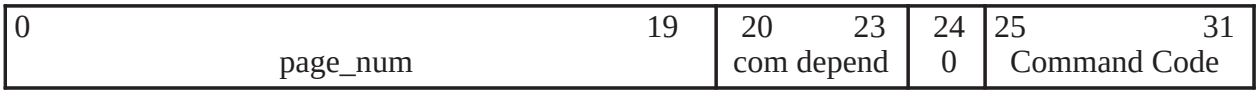
The IO_DC_ADDRESS register specifies which 32 bit word of an IOA's IODC is to be returned on a read from the IO_DC_DATA register. IO_DC_ADDRESS need only be a single bit register as shown below:

If IO_DC_ADDRESS[wd] is set to 1, reads from IO_DC_DATA will return IO_DC_DATA[32:63]. If IO_DC_ADDRESS[wd] is set to 0, reads from IO_DC_DATA will return IO_DC_DATA[0:31].

This register powers up with bit 29 being set to 0. Its value is unchanged as a result of a CMD_Reset or a CMD_Clear.

2.1.1.3. Runway HPA IO_COMMAND Register

The Runway IO_COMMAND register has the following format:



Bits 0..19 of the command dependent field contain the I/O virtual page number (called page_num in the following table) for the TLB_Purge and TLB_Insert commands.

The following commands are supported:

Command Name	Command Code	Command Operation
Reset	5	Reset module
TLB_Purge	2'd33	Delete TLB entry specified by page_num
TLB_Insert	2'd34	Fetch TLB entry specified by page_num
TLB_Direct_Write	2'd35	Write TLB entry specified by page_num
Clear	3	Clears IO_STATUS error bits
TEST_Ld_OutQ	2'd64	Load dummy Outbound Queue Entry
TEST_Ld_RRQ	2'd65	Load dummy Read Return Queue Entry

The Runway IO_COMMAND register is a write only register. This is because the I/O Adapter simply executes the command specified, as opposed to storing a value corresponding to the command code. The Reset command is decoded by the Runway receive block and reset signals are driven and synchronized to the GSC+ block. Software must synchronize TLB updates with transactions in progress.

CMD_Reset causes the I/O Adapter to behave identically to a power_on reset, with the exception that the only registers which change values are IO_STATUS, IO_CONTROL, and the IO_ERR registers on both the GSC+ and Runway ports. Additionally, the GSC+ port will assert ErrorL under certain circumstances on GSC+ during a CMD_Reset, to conclude any outstanding transactions. However, the contents of all queues (inbound, outbound, and data read return) and the pool entries are invalidated. Note that a CMD_Reset will take 32 – 40 Runway cycles, during which time the I/O Adapter will not recognize transactions in progress. Subsequent writes will appear discarded and reads will result in a timeout.

CMD_TLB_Purge causes the I/O Adapter to mark invalid the I/O TLB entry, corresponding to the I/O virtual page number specified by the page num field.

CMD_TLB_Insert causes the I/O Adapter to fetch the I/O TLB entry, corresponding to the I/O virtual page number specified by the page num field, from the memory I/O PDIR table.

CMD_TLB_Direct_Write causes the I/O Adapter to fill the I/O TLB entry, corresponding to the I/O virtual page number specified by the page num field, with the value in the I/O_TLB_ENTRY_L and I/O_TLB_ENTRY_R registers.

CMD_Clear causes the I/O Adapter to clear the soft error indication in the Runway IO_STATUS register.

The TEST commands instruct the I/O Adapter to fake a GSC+ DMA transaction, for test purposes. The dummy GSC+ DMA test is described in the testability section.

TEST_Ld_OutQ triggers the Runway slave block to load the contents of the TEST_ADDRESS and TEST_INFO into the Outbound Command Queue.

TEST_Ld_RRQ triggers the Runway slave block to load the contents of the TEST_ADDRESS and TEST_INFO into the Read Return Queue.

2.1.1.4. Runway IO_STATUS Register

The Runway IO_STATUS register has the following format:

0 11	12 14	15	16 21	22	23	24	25	26 27	28 30	31
HV	R	HV	estat	se	he	fe	ry	R	000	pf

This register serves two main purposes: to indicate when the Runway IO_COMMAND register is ready for a new command and to relay error information. Bit 28 is zero, indicating that this is an upper port (Runway) register. Bits 29 and 30 are zero, indicating that the lower port bus has power available. Since both the upper and lower port busses are powered off of the same supply, it is safe to assume that the lower port has power if UTurn has power.

Bit 31 would typically indicate a power fail occurrence on the remote (GSC+) bus. However, on UTurn this bit simply indicates whether the lower port GSC+ bus is in the midst of a reset (GSC+ RESETL asserted). No transactions are allowed on GSC+ during a reset, so software must check the pf bit prior to issuing transactions to the GSC+ bus if a reset may be in progress. If the value of the pf bit is 1, then the lower port is unavailable and subsequent transactions directed to GSC+ may be lost. If the value of the pf bit is 0, then transactions to GSC+ should proceed normally. Accesses to PDC space and to UTurn internal registers can continue regardless of the value of the pf bit.

The ry bit is set when UTurn is available to accept a new command. UTurn will respond to only one IO_STATUS read when it is busy processing an IO_COMMAND; for such a read, the ry bit set to 0. After completion of an IO_COMMAND, a subsequent IO_STATUS register read will return a value of 1 for the ry bit.

The se (soft error), he (hard error), and fe (fatal error) bits indicate the severity of the error. The estat field is encoded to indicate error type. Values for this field are presented in the error handling section of this chapter, along with more discussion of the errors in general.

This register is cleared (set to x'00000041) at power–on and set to x'00000040 after a CMD_Reset is issued or after a CMD_Clear is issued and the IO_Status Register has logged a soft error. If a CMD_Clear occurs and the IO_STATUS Register does not have a soft error logged, then the register is unchanged.

2.1.1.5. Runway IO_CONTROL Register

The Runway IO_CONTROL register, which controls the forwarding of transactions, has the following format:

0 13	14 15	16 21	22	23 24	25 31
HV	TLB	R	HV	mode	reserved

The mode field indicates the address translation required for the transactions forwarded from Runway to GSC+, and is defined as follows:

Mode Name	Value	Definition
Off	0	Opaque to matching addresses.
Include	1	Transparent for matching addresses.
Peek	3	Map matching addresses.

The Runway IO_CONTROL Register defaults to Off mode when a power–on occurs or when a CMD_Reset is received. This mode will cause Runway transactions which match the I/O range specified by the IO_IO_LOW and IO_IO_HIGH registers to be ignored.

In Include, all addresses within the I/O range specified by the IO_IO_LOW and IO_IO_HIGH registers are transparently forwarded. This is the I/O Adapter's normal operating mode.

Peek mode is used during system configuration to initialize the GSC+ bus. In Peek mode, Runway Write_Shorts in the address range specified by IO_IO_LOW and IO_IO_HIGH are forwarded through the I/O Adapter, but with a modified address. The GSC+ address is remapped to the Broadcast Physical Address space by setting the 14 high order address bits of the 32 bit GSC+ address to ones.

On UTurn, the TLB field is used to manipulate the I/O TLB to affect transactions which are forwarded from GSC+ to Runway. The TLB field is defined as follows:

TLB Mode	Value	Description
Real	0	No TLB translation (virtual address affected by physical address)
Error	1	Software fills the TLB manually and misses logged as fatal error.
Normal	2	IOA fetch TLB misses from the memory IO–PDIR.

The TLB field of the Runway IO_CONTROL defaults to Real mode when a power–on occurs. Real mode is used during system configuration, before an I/O PDIR exists in memory. When in real mode, the I/O–TLB will not be accessed to obtain the virtual address of a memory space GSC+ transaction. Instead, the address is directly mapped and the virtual address is composed of selected physical bits.

Error mode provides the operating system the flexibility of not implementing an I/O PDIR. In this mode, software is responsible for manually inserting I/O TLB entries using the CMD_TLB_Direct_Write mechanism described in the Runway IO_COMMAND Register discussion. However, if a GSC+ transaction is issued and there is no corresponding I/O TLB entry, the miss is logged as a hard error.

Typically, each IOA is expected to operate in Normal mode. When in this mode, the I/O Adapter performs fetches from the memory resident IO–PDIR on TLB misses.

Since the Runway IO_CONTROL register impacts the Runway slave and master operation, it will reside on the Runway side of the chip. It is accessed by PDC and the OS during configuration. Writes to this register will bypass the queues and take effect immediately. Reads from this register are queued in the outbound command queue and the inbound queue. This design approach means that a Runway IO_CONTROL register read could be bypassed by an IO_CONTROL register write. Additionally, software must to be cautious when issuing a write to the IO_CONTROL register while I/O is in progress, as updates to the register would have an immediate affect on the way in which these I/O transactions are issued over Runway. Therefore software which updates this register might want to prevent an update until the queues are known to be empty (no I/O is in progress).

The Runway IO_CONTROL is set to X'00000000 when a power–on occurs. The HV fields will always be set to zeroes. The register is unchanged, as a result of a CMD_Clear. When a CMD_Reset occurs, the TLB field should be unaffected, but the remaining fields are set to zero.

2.1.2. Runway Auxiliary Register Set (Register Set 1)

The following registers comprise the auxiliary register set:

Offset	Access Modes	Class	Name	Address LSB
0	R	A	IO_ERR_RESP	.. 040
1	R	A	IO_ERR_INFO	.. 044
2	R	A	IO_ERR_REQ	.. 048
3	R	HV	IO_ERR_RESP_HI	.. 04c
4	RW	HV	IO_TLB_ENTRY_M	.. 050
5	RW	HV	IO_TLB_ENTRY_L	.. 054
7	RW	HV	IO_PDIR_BASE	.. 05c
8	RW	HV	IO_IO_LOW_HV	.. 060
9	RW	HV	IO_IO_HIGH_HV	.. 064
11	RW	HV	IO_CHAIN_ID_MASK	.. 06c
14	RW	A	IO_IO_LOW	.. 078
15	RW	A	IO_IO_HIGH	.. 07c

It should be noted that there is a requirement for an IO_TLB_SIZE register, but this functionality is emulated by PDC.

2.1.2.1. Runway IO_ERR_RESP_HI and IO_ERR_RESP Registers

These registers indicate the module which was responsible for responding to a Runway operation which failed. The slave address of the transaction is stored. The format of these registers are as follows:

0 23	24 31
000000000000000000000000	sys–resp–address[0:7]

0 31
sys–resp–address[8:39]

The contents of these registers are not necessarily valid. Validity is indicated by the rsi bit in the Runway IO_ERR_INFO register.

These registers do not have a defined power–on value and are unaffected by CMD_Clear or CMD_Reset.

2.1.2.2. Runway IO_ERR_INFO Register

This register indicates the validity of the Runway IO_ERR_RESP(_HI) and IO_ERR_REQ registers. The format of this register is as follows:

0 6	7	8 13	14 19	20	21 28	29	30	31
undefined	0	R	source = 111111	sv	R	rm	rsi	rqi

If rqi is 0, the contents of the IO_ERR_REQ register are valid and if rsi is 0, then contents of the IO_ERR_RESP(_HI) registers are valid. The rm bit indicates whether the requestor is on the remote bus (GSC+). If this bit is set to 1, then the System Requestor Fixed and Flex fields in the IO_ERR_REQ

Register specify the GSC+ guest id. If this bit is a 0, then the System Requestor Fixed and Flex fields specify the Runway Master_ID of the requestor of the erroneous transaction. The sv bit indicates the source of a Runway read return (assuming the logged transaction is indeed a read return). Since the responding processor can not be identified on a cache–to–cache copy, the sv bit is set to 0 on C2C_Writes. However, on memory read returns, the sv bit is set to one, and the source field indicates that the errored response was issued by the main memory controller.

This register is set to x'0003F003 when either of the following three conditions occur: a CMD_Clear is issued and the IO_STATUS register currently has a soft error logged, a CMD_Reset is issued, or at power–on. If a CMD_Clear occurs and the IO_STATUS register does not have a soft error logged, then the register is unchanged.

2.1.2.3. Runway IO_ERR_REQ Register

This register logs the HPA of the requester of an erroneous Runway operation. The format of this register is as follows:

0 7	8 15	16 25	26 31
11111111	extend	System Requester Flex	Sys Req Fixed

The value of the System Requester Flex is set to the Runway Flex value (which is hardwired), if this I/O Adapter detected this error as a Runway slave. If the error is the result of a transaction which this I/O Adapter attempts to master to Runway on behalf of a GSC+ guest, then the GSC+ Flex field is set based on the GSC+_SHADOW_FLEX register.

If this I/O Adapter is a Runway slave of the transaction, then the fixed field is set as follows:

26	27	28 30	31
1	0	master_id	0

where master_id corresponds to the Runway master of the failing transaction.

If this I/O Adapter is a Runway master (on behalf of a GSC+ guest) of the transaction, then the fixed field has the following format:

26	26 29	30 31
0	guest_id	00

where guest_id corresponds to the GSC+ guest master of the failing transaction. Note that this is not the entire fixed field of a GSC+ guest. Bits 30 through 31 typically correspond to the GSC+ guest submodule. Unfortunately, UTurn does not have visibility of the submodule.

The contents of this register is not necessarily valid. Validity is indicated by the rqi bit in the Runway IO_ERR_INFO register.

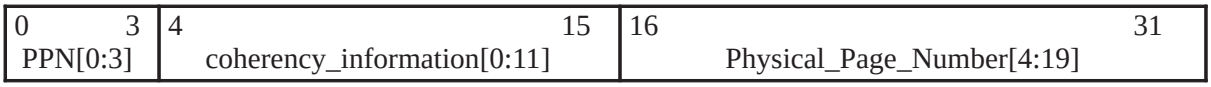
The IO_ERR_REQ register is undefined at power–on and is unaffected by CMD_Clear or CMD_Reset.

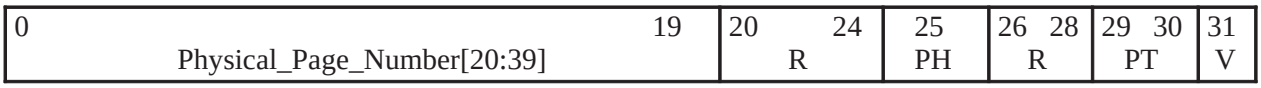
2.1.2.4. IO_TLB_ENTRY Registers

The IO_TLB_ENTRY registers are used by the CMD_TLB_Direct_Write. Software must load these registers with the entry it intends to store in the TLB, and then issue a CMD_TLB_Direct_Write to the Runway IO_COMMAND with the target TLB location specified in the page_num field.

Because an I/O–PDIR entry is 64 bits in width, two IO_TLB_ENTRY registers (IO_TLB_ENTRY_M and IO_TLB_ENTRY_L) are required to denote the most significant and least significant contents. The width of the I/O TLB is 52 bits which include the physical page number, eight bits of the coherency information, the sequential prefetch hint, the page type field, and the valid indicator. These fields are explained in more detail in the Inbound Runway Chapter.

The format of the registers is identical to the IO–PDIR format, which is as follows, with the most significant (IO_TLB_ENTRY_M) register illustrated first, followed by the least significant register (IO_TLB_ENTRY_L):





where: PPN[0:3] are Physical_Page_Number[0:3]

R	is a reserved field (all bits in reserved fields are zero when read)
PH	is the prefetch hint (bit indicates prefetch enable)
PT	is the page type indicators
V	is a valid indicator

These registers are located in the Runway inbound block. Writes to these registers will take effect immediately, bypassing the queues, while reads are queued. This introduces the possibility of a read being bypassed by a subsequent write to the register. This sequence can be avoided by having software wait for read returns before issuing subsequent writes.

The contents of these registers will not change as a result of a CMD_Reset or a CMD_Clear, and are undefined as a result of a power–on.

2.1.2.5. IO_PDIR_BASE Register

The IO_PDIR_BASE register contains the base address of the IO–PDIR translation table. When the I/O Adapter is in ”normal mode”, as indicated by the IO_CONTROL register, the Runway inbound block will access the TLB for translations on coherent accesses. If there is a TLB miss, the value in the IO_PDIR_BASE register together with the virtual I/O address of the transaction at the head of the inbound queue are used to fetch the missing translation from the IO–PDIR in main memory.

Because the IO–PDIR is guaranteed to begin on a page boundary and reside in the first 4 GB of memory, only 20 bits of the address need to be stored. These 20 bits correspond to REAL_ADDR[8:27] (which maps to Runway bus ADDR_DATA[32:51]). The format of the register is therefore as follows:

This register is located in the Runway inbound block. It should be initialized by that part of the kernel which sets up the IO_PDIR in memory. Following initialization, subsequent writes should be unnecessary. Writes to this register will take effect immediately, bypassing the queues, while reads are queued. This introduces the possibility of a read being bypassed by a subsequent write to the register. This sequence can be avoided by having software wait for read returns before issuing subsequent writes.

The contents of this register will not change as a result of a CMD_Reset or a CMD_Clear. At power–on, the value of the IO_PDIR_BASE field is undefined and the value of bits 20 through 31 is zero.

2.1.2.6. IO_CHAIN_ID_MASK Register

The IO_CHAIN_ID_MASK register is initialized by software, based on the size of the I/O Adapter resident TLB (specified by IO_TLB_SIZE). It is used as a mask that can extract the ChainID from an I/O Virtual Address, on inbound coherent transactions. This extracted ChainID becomes the index into the I/O TLB. Note that the IO_CHAIN_ID_MASK must not change once the TLB has valid entries, otherwise the TLB must be completely re–initialized. Because the ChainID is composed of the most significant 20 bits of the I/O virtual address, its format is as follows:

0 19	20 31
IO_CHAIN_ID_MASK	000000000000

This register is located in the Runway inbound block. Writes to this register will take effect immediately, bypassing the queues, while reads are queued. This introduces the possibility of a read being bypassed by a subsequent write to the register. This sequence can be avoided by having software wait for read returns before issuing subsequent writes. As this register is initialized by software prior to going from real to virtual mode, subsequent writes should be unnecessary and avoided once the TLB has valid entries.

The contents of this register will not change as a result of a CMD_Reset or a CMD_Clear. At power–on, the value of the IO_CHAIN_ID_MASK field is undefined and the value of bits 20 through 31 is zero.

2.1.2.7. IO_IO_LOW(_HV) and IO_IO_HIGH(_HV) Registers

IO_IO_LOW and IO_IO_HIGH set the lower and upper bounds of the I/O Adapter address space, respectively. They both have the following format:

0 7	8 15	16 31
11111111	11111111	address

Note that there are two sets of these registers. This is to accommodate two disjoint address space regions per GSC+ port. Each incoming Runway transaction address is compared with both sets of IO_IO_LOW and IO_IO_HIGH to determine if the address is in the range greater than or equal to IO_IO_LOW and less than IO_IO_HIGH. If the incoming Runway transaction address is outside of the specified address range, then the transaction is not intended for this UTurn. Note: It is not an error to have IO_IO_LOW equal to or greater than IO_IO_HIGH. In fact, this is the only method to avoid specifying an address space region.

In order for a Runway address to reside within GSC+ extended address space:

Runway Address [0:7]	must identically compare to 8’b11111111.
Runway Address [8:11]	must be equal to IO_IO_LOW(_HV)[16:19]
Runway Address [12:23]	must be greater than or equal to IO_IO_LOW(_HV)[20:31] and less than IO_IO_HIGH(_HV)[20:31].
Runway Address [24:39]	is not used in the comparison.

When the Runway transaction is forwarded to GSC+, the GSC+ address is as follows:

GSC+ Address[0:3]	4’b1111

GSC+ Address[4:29]	Runway Address[12:37]
GSC+ Address[30:31]	2'b00

These registers reside in the Runway outbound block, as Runway slave transactions must be checked against these bounds registers to determine if the slave transaction address is in the range of this I/O Adapter. These registers must be initialized by PDC, once the lower bus is interrogated and address space is defined. The operating system will modify the architectural IO_IO_LOW and IO_IO_HIGH registers following the PDC initialization. However, the hardware version dependent IO_IO_LOW and IO_IO_HIGH registers should not be subsequently altered by the OS. Writes to both sets of registers will take effect immediately, bypassing the queues, which ensures that subsequent Runway transactions are checked against the updated bounds values. However reads are queued, introducing the possibility of a read being bypassed by a subsequent write to the same register. This sequence can be avoided by having software wait for read returns before issuing subsequent writes.

The contents of these registers will not change as a result of a CMD_Reset or a CMD_Clear. At power–on, the value of bits 0 through 15 are ones and the value of the address field is undefined.

2.1.3. UTurn Specific Register Set (Register Set 17)

The following registers comprise the UTurn Specific Register Set:

Offset	Access Modes	Class	UTurn Specific Register Name	Address LSB
0	RW	HV	QUEUE/POOL_DEPTH_CTL	.. 440
1	RW	HV	EIM_MONARCH_AND_GROUP	.. 444
2	RW	HV	TOC_MONARCH_CLIENT_ID	.. 448
3	R	HV	READ_TLB_TAG	.. 44c
4	R	HV	READ_TLB_M	.. 450
5	R	HV	READ_TLB_L	.. 454
6	RW	HV	TEST_ADDRESS	.. 458
7	RW	HV	TEST_INFO/CONFIG	.. 45c
8	R	HV	GSC+_SHADOW_FLEX	.. 460
9	RW0	HV	PERF_CTR1	.. 464
10	RW0	HV	PERF_CTR2	.. 468
11	RW	HV	PERF_MODE	.. 46c

2.1.3.1. QUEUE/POOL_DEPTH_CTL Register

The purpose of the QUEUE/POOL_DEPTH_CTL Register is to indicate the queue depths of the inbound queue, read return queue, and outbound queue and the size of the pool. This degree of configurability is expected to enhance performance measurements and testability.

The format of the QUEUE/POOL_DEPTH_CTL Register is as follows:

0 2	3 7	8 15	16 19	20 26	27 31
R	OQ_hyst_ctl	unused_pool	inQ_unusd_loc	R	OutQ_unusd_loc

The inQ_unusd_loc and OutQ_unusd_loc fields indicate the unused portions of the respective queues. The outbound command queue has a high water mark, and when the queue depth reaches this point, the signal STOP_IO is asserted on Runway to flow control processor initiated read_short and write_short transactions to I/O space. For the Outbound Command Queue, the OutQ_unusd_loc field determines the placement of this high water mark. Additionally, the field OQ_hyst_ctl is the OutQ's hysteresis control. Once STOP_IO is asserted, this field indicates how many queue locations must drain, prior to the deassertion of STOP_IO. The unused_pool field contains one bit for every pool entry. This bit indicates whether the corresponding pool entry is unavailable for use.

Since the depths of the UTurn internal queues and pool are configurable, anyone altering the default sizes should be aware (and beware) of the following UTurn rules. When reducing the size of the queues or pool, one should make a corresponding reduction in the number of GSC+ guests to avoid deadlocks and overruns.

Pool size rule: for all outstanding reads on RUNWAY, UTurn must have a place to store the transaction information (which GSC guest, and length of read requested). UTurn is limited to 8 unique transaction IDs on Runway per master ID (per IOA). Thus, the pool has 8 entries. If fewer than 4 guests are connected, the pool could be smaller, but its minimum size must be twice the number of GSC guests.

Strange conditions: Under some conditions, all formally–requested GSC+ reads (one per guest) could be in the pool, if all reads have been issued on RUNWAY and no data has been returned. Similarly, all reads could be in the read_return_queue, if all reads have had data returned and the data needs to be returned to the guests. UTurn accommodates these two allowable (but unlikely) extreme conditions.

The minimum depth of the outbound command queue (OutQ) is a function of how many Runway transactions can be issued following the transaction that causes UTurn to hit its OutQ high water mark. UTurn is implemented such that a maximum of 7 Runway transactions can be issued in the time it takes UTurn to detect that the OutQ high water mark has been reached, assert STOP_IO to flow control Runway, and have the processors all see STOP_IO and halt further I/O operations. Therefore the OutQ_unusd_loc field must be set to no fewer than 7. The range of valid values for the OQ_hyst_ctl field is 1 through 26.

The minimum allowed value for inQ_unusd_loc is 1.

The QUEUE/POOL_DEPTH_CTL register is located in the Runway inbound block. Only PDC is allowed to manipulate the contents of this register. This register must not be written to affect a queue which has any valid entries. More specifically, PDC must issue a CMD_Reset immediately prior to altering the QUEUE/POOL_DEPTH_CTL register to ensure that there are no pending transactions to HPA, BPA, PDC, or GSC+ space in the outbound queue and no DMAs in the inbound queue. Software is also required to issue a CMD_Reset immediately following a write to the POOL_DEPTH_CTL register.

On power on, the OutQ_unusd_loc field is set to 1'd7, the OQ_hyst_ctl and inQ_unusd_loc fields are set to 1'd1, and all remaining bits are set to zero. The contents of this register are unaffected as a result of a CMD_Reset or a CMD_Clear. Its power–on value is X'01001007.

2.1.3.2. EIM_MONARCH_AND_GROUP Register

The EIM_MONARCH_AND_GROUP register provides the information needed to convert a GSC+ InterruptL assertion into a processor external interrupt message.

The register consists of two programmable fields. The client_id field is a four bit value which PDC must load with the Runway CLIENT_ID of the processor which has been identified as the monarch for receipt of IO_EIR writes. This four bit value corresponds to the fixed field of the monarch processor and is con-

catenated between the hardwired flex field and register information to form the the address of the monarch processor's IO_EIR register. The second programmable field is called the group field, which comprises the data portion of the IO_EIR write to the monarch processor. This 6 bit group encoding indicates a bit in the processor's 64 bit EIR control register.

The following diagram illustrates the format of the EIM_MONARCH_AND_GROUP register and is used to compose both the address of the monarch processor's IO_EIR and the write data contents:

0 3	4 13	14	15 18	19	20 25	26 31
1111	flex = 1111111110	1	client_id	0	reg_set = 000000	group

The client_id field (bits 15 thru 18) and the group field (bits 26 through 31) are loaded on writes to this register. However the full contents of the register are returned on a read.

When a GSC+ INTERRUPTL signal is detected by the GSC+ block, an entry is placed in the inbound queue which triggers the Runway inbound block to master a WRITE_SHORT of the encoded interrupt value to the monarch processor's IO_EIR register. This address is computed as follows:

io_eir_addr <– 0xFFFFF000 & EIM_MONARCH_AND_GROUP (and F extend to 40 bits)

The significant data corresponding to this write is simply the group field. Therefore,

io_eir_data <– EIM_MONARCH_AND_GROUP

The EIM_MONARCH_AND_CLIENT register is located in the Runway inbound block. It is the responsibility of PDC to set up the contents of this register. Due to its location, writes to this register occur immediately, bypassing the queues, whereas reads from this register are queued. This approach allows writes to bypass previously issued reads stored in the queue. Therefore it is recommended that software wait for read responses from this register before issuing stores.

On power on, the client_id and group fields are set to zero, causing a register value of X'FFFA0000. The contents of this register will not change as a result of a CMD_Reset or a CMD_Clear.

2.1.3.3. TOC_MONARCH_CLIENT_ID Register

The TOC_MONARCH_CLIENT_ID register is just a four bit value which PDC must load with the Runway CLIENT_ID of the processor which has been identified as the monarch for receipt of directed CMD_Resets. 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 the address of the monarch processor's IO_COMMAND register.

The following diagram illustrates the format of the TOC_MONARCH_CLIENT_ID register and corresponds to the address of the monarch processor's SRS IO_COMMAND:

0 3	4 13	14	15 18	19	20 25	26 29	30 31
1111	flex = 1111111110	1	client_id	0	regset = 000000	reg=1100	00

The client_id field (bits 15 thru 18) are loaded on writes to this register. However the full contents of the register are returned on a read.

The I/O Adapter will issue a directed CMD_RESET write (value equal to x'00000005) to the monarch processor's IO_COMMAND register in response to a user–initiated Transfer–of–Control (Control–B TC at the console routed through the Access Port or assertion of the panel switch).

The TOC_MONARCH_CLIENT_ID register is located in the Runway inbound block. When a TOC assertion is detected by the Runway inbound block, it will master a WRITE_SHORT with the CMD_RESET command code to the address specified by this register. It is the responsibility of PDC to set up the contents of this register. Due to its location, writes to this register occur immediately, bypassing the queues, whereas reads from this register are queued. This approach allows writes to bypass previously issued reads stored in the queue. Therefore it is recommended that software wait for read responses from this register before issuing stores.

On power on, the client_id field is set to zero, causing a register value of X'FFFA0030. The contents of this register will not change as a result of a CMD_Reset or a CMD_Clear.

2.1.3.4. READ_TLB_TAG Register

The READ_TLB_TAG register allows software to access the tag associated with a given I/O TLB lookup. To read the tag, first the TEST_ADDRESS Register must be loaded with an I/O TLB address (which indicates the I/O TLB lookup or chain id). Then a read of the READ_TLB_TAG Register will return the tag (or block id) associated with the specified I/O TLB chain id.

This register accompanies the READ_TLB Registers described below. Specifically, if the READ_TLB_TAG register returns a block id which matches the block id in the TEST_ADDRESS Register, then by reading the READ_TLB registers, the TLB entry associated with the TEST_ADDRESS can be obtained. However, if the READ_TLB_TAG register returns a block id which differs from the block id in the TEST ADDRESS Register, then a read of READ_TLB registers will still return the TLB entry associated with the chain id, although technically this is a miss.

The format of the READ_TLB_TAG Register is as follows:

0 7	8 19	20 31
undefined	tag (or block id)	undefined

Note that the READ_TLB_TAG register is read only. It is located in the Runway block. The Runway outbound controller manages the TLB read mechanism, by forming an Outbound Queue Entry which contains the page_num field when a read from READ_TLB_TAG occurs. This queue entry gets redirected to the Inbound Queue and when it reaches the head of the queue, the inbound Runway controller obtains the TLB tag and issues the read return transaction across Runway. As with all registers located in the Runway block, writes occur immediately, whereas reads are queued. Therefore, it is recommended that the READ_TLB_TAG read return is received before the processor overwrites the TEST_ADDRESS register.

On power–on, the READ_TLB_TAG register is undefined. The READ_TLB_TAG register is unaffected by CMD_Reset or CMD_Clear.

2.1.3.5. READ_TLB Registers

The READ_TLB_M and READ_TLB_L registers facilitate reading entries from the I/O TLB. To read an entry in the I/O TLB, first the TEST_ADDRESS must be loaded with the I/O TLB address (which corresponds to the I/O Virtual Page Number). Then a read of either READ_TLB_M or READ_TLB_L will return the I/O TLB entry specified by the I/O Virtual Page Number stored in the TEST_ADDRESS register.

The format of the READ_TLB_M and READ_TLB_L registers is illustrated, respectively, below:

0 7	8 15	16 23	24 31
R	coherency_info[4:11]	R	Phys_Page_Num[12:19]

0 19	20 24	25	26 28	29 30	31
Physical_Page_Number[20:39]	R	prefetch	R	PT	V

Note that the READ_TLB_M and READ_TLB_L registers are read only. A read from either of these two registers returns the entire 64 bit entry. However, the processor simply expects the most significant portion of the entry on the high order bits when it reads the register offset of 4 and the least significant portion of the entry on the low order bits when it reads the register offset of 5.

Both registers are located in the Runway block. The Runway outbound controller manages the TLB read mechanism, by forming an Outbound Queue Entry which contains the page_num field, when a read from either READ_TLB_M and READ_TLB_L occurs. This queue entry gets redirected to the Inbound Queue and when it reaches the head of the queue, the inbound Runway controller obtains the TLB contents and issues the read return transaction across Runway. Because the processor must issue a read of both registers before it obtains a full entry, it is responsible for ensuring that the TEST_ADDRESS register is not overwritten between these two reads. As with all registers located in the Runway block, writes occur immediately, whereas reads are queued. Therefore, it is recommended that both read returns are received before the processor overwrites the TEST_ADDRESS register.

On power–on, both registers have undefined values. Likewise, these register are unaffected as a result of a CMD_Reset or a CMD_Clear.

2.1.3.6. TEST_ADDRESS Register

The TEST_ADDRESS Register is used for testing the I/O Adapter. When a read from the I/O TLB is requested, the TEST_ADDRESS Register contains the I/O Virtual Page Number. When a dummy GSC+ DMA transaction is created for test purposes, this register contains the GSC+ Slave Address. The I/O TLB test is described in more detail in the COMMAND_READ_TLB Register section and the dummy GSC+ DMA test is described in the testability section.

There are two formats for the TEST_ADDRESS Register, based on the type of test in progress. The formats below show first the format for the I/O TLB test, and second the format for the GSC+ DMA test. Note that in both cases, bits 30:31 are guaranteed to be zero.

0 19	20 31
I/O Virtual Page Number	not relevant

0 29	30 31
GSC+ Slave Address	R

This register is located in the Runway block. As with all registers located in the Runway block, writes occur immediately, whereas reads are queued. Because this register influences the operation of a test sequence, it is recommended that the test completes, determined by the completion of read returns, before the processor overwrites the TEST_ADDRESS register.

On power–on, the TEST_ADDRESS register has an undefined value. This register is unaffected by a CMD_Reset or a CMD_Clear.

2.1.3.7. TEST_INFO/CONFIG Register

The TEST_INFO/CONFIG Register is used for testing and configuring the I/O Adapter. The TEST_INFO portion of this register specifies the type of dummy GSC+ DMA transaction to create for test purposes. The TEST_ADDRESS together with this TEST_INFO determine the queue entry contents to be loaded in the queue designated by the TEST_COMMAND. The dummy GSC+ DMA test is described in detail in the testability section. The CONFIG portion of this register provides enables/disables for various capabilities, including parity error checking, error modes, and viewport.

The format for the TEST_INFO/CONFIG Register is as follows:

0 2	3 6	7 10	11	12	13	14	15 17	18 24	25	26	27	28	29	30	31
gst_id	byt_enb	type	con	loc	prft	Adr	WinL	R	spd	any	ato	ret	err	par	vpt

where:

gst_id	specifies the mastering GSC+ guest_id (values range from 0 to 5).
byt_enb	specifies the GSC+ byte enable of the intended transaction
type	corresponds to the GSC+ transaction type
con	means that the specified transaction should be connected
loc	means that the specified transaction is to be locked by assertion of LSL
prft	means that the intended transaction must specify prefetch
Adr	together with the guest_id, determines the outbound RAM address
WinL	specifies the GSC+ word in line (word address offset within cache line)
R	is a reserved field (all bits are set to zero in reserved fields)
spd	DMA read speed–up disable (default is 0 – speed–up enabled)
any	enables / disables the any_trans backoff mechanism
ato	enables / disables the atomic backoff mechanism
ret	enables / disables the ret_only backoff mechanism
err	enables / disables error modes
par	enables / disables Runway parity checking
vpt	enables / disables blocks dumping to viewport

Note, all fields are positive true, with the exception of the byte_enb field which corresponds identically with the GSC+ byte enable bus, which is negative true.

DMA read speed–ups are incorporated in the UTurn design, reducing latency for DMA reads compared to the U2 design. The spd bit, when set, allows these speed–ups to be disabled in case of some unforeseen problem with the speed–ups.

When error modes are disabled (set to 0), the Runway side of this IOA will still detect and log any observed errors. However, it will not go into either fatal or hard error mode. This allows subsequent transactions to proceed. When parity checking is disabled (set to 0), this IOA will not perform parity checking or logging on any Runway transactions. However, UTurn will always continue to generate proper Runway parity. When the viewport is enabled, the PDC address port will dump state information of various blocks within UTurn. This is programmable through the PDC data port. The viewport functionality is documented in detail in the testability chapter of this document.

The TEST_INFO/CONFIG register is located in the Runway block. As with all registers located in the Runway block, writes occur immediately, whereas reads are queued. Because this register influences the operation of a test sequence, it is recommended that the test completes (determined by the completion of read returns) before the processor overwrites the TEST_INFO/CONFIG register.

On power–on, the value of the TEST_INFO/CONFIG register is undefined except for the least significant 3 nibbles– x'?????039. This register is unaffected by a CMD_Reset or a CMD_Clear.

2.1.3.8. GSC+_SHADOW_FLEX Register

The format of the read–only GSC+_SHADOW_FLEX register is as follows:

0 3	4 13	14 30	31
1111	flex	00000000000000000	enb

The flex field in the GSC+_SHADOW_FLEX register specifies the programmable portion of the I/O Adapter's HPA space for the GSC+ port. This value is loaded by PDC and the OS during system configuration. The GSC+_SHADOW_FLEX register also contains the enable bit (enb), which indicates enabling and disabling of GSC+ modules from arbitration for bus mastership.

This register is readable by indicating register set 17 and register offset 8 in the Runway address. Its readable contents are simply a reflection of the GSC+ IO_FLEX register in GSC+ Broadcast Physical Address Space. Therefore, to change the value contained in this register, the Runway IO_CONTROL register must be placed in peek mode and a Runway WRITE_SHORT transaction must be issued with Runway address bits 0 through 3 being ones, address bits 4 through 13 falling between an IO_IO_LOW(_HV) and corresponding IO_IO_HIGH(_HV) field, address bits 14 and 15 being zeroes, and address bits 16 through 31 equal to 4'h0020.

The GSC+_SHADOW_FLEX register is located in the Runway block so that software can obtain the lower port flex value associated with this Runway block, since the broadcast GSC+ IO_FLEX is unreadable.

At power–on, the register is set to X'FFF80000. The register is unaffected by CMD_Reset or CMD_Clear.

2.1.3.9. Runway PERF_CTR1 and PERF_CTR2 Registers

The two performance counters on the Runway side of the chip count event cycles and occurrences as programmed in the PERF_MODE Register (see below). They are readable; reads do not alter the value in the register. A write to these registers will automatically clear the value in the register, independent of the data value written.

The format of the PERF_CTR1 and PERF_CTR2 Registers is as follows:

0 31
count

The Runway PERF_CTR registers are intended to be accessed by test code only. Writes to these registers will bypass the queues and take effect immediately. Reads from these registers are queued in the outbound command queue and the inbound queue. This means that a Runway PERF_CTR register read could be bypassed by a write. Additionally, software must to be cautious when clearing a PERF_CTR register while transactions are in progress, as these transactions may be computed into the next set of counts. Therefore software which updates this register might want to prevent an update until the queues are known to be empty (no I/O is in progress).

The Runway PERF_CTR1 and PERF_CTR2 registers power–up undefined. The registers are left unchanged as a result of a CMD_Clear or CMD_Reset.

2.1.3.10. Runway PERF_MODE Register

The Runway PERF_MODE Register is programmed to manipulate the Runway PERF_CTR Registers. This register will enhance performance measurements and testability.

The format of the PERF_MODE Register is as follows:

0 11	12 15	16 27	28 31
R	mode1	R	mode2

where mode1 and mode2 are defined as follows:

0000	number of tlb accesses
0001	number of tlb accesses resulting in a miss
0010	number of cycles stop_most is asserted
0011	number of stop_most occurrences
0100	number of cycles stop_io is asserted
0101	number of stop_io occurrences
0110	number of prefetches
0111	number of prefetch hits
1000	number of cycles ccc responses are stalled due to a private hit
1001	number of ccc response stall occurrences
1010	number of cycles that pool is full
1011	number of pool full occurrences
1100	number of Runway data cycles (cycles in which data_valid is asserted)
1101	number of Runway clock cycles / 2 (R2clk)
1110	number of 32–byte transactions on Runway
1111	number of c2c–writes on Runway

The Runway PERF_MODE register resides on the Runway side of the chip, and is intended to be accessed by test code only. The register is both readable and writeable. Writes to this register will bypass the queues and take effect immediately. Reads from this register are queued in the outbound command queue and the inbound queue. This means that a Runway PERF_MODE register read could be bypassed by a write. Additionally, software must to be cautious when issuing a write to the PERF_MODE register while transactions are in progress, as these transactions may be computed into the counts. Therefore software which updates this register might want to prevent an update until the queues are known to be empty (no I/O is in progress).

The contents of the Runway PERF_MODE Register are undefined when a power–on occurs. The register is unaffected by CMD_Clear or CMD_Reset.

2.2. GSC+ Hard Physical Address (HPA) Space

The GSC+ HPA space consists of eleven register sets: the GSC+ Supervisory Register Set, the GSC+ Auxiliary Register Set, UTurn RAM Register Sets 1 through 7, the GSC+ Bus Specific Register Set, and the GSC+ Performance Counter Register Set. Each GSC+ port is allocated 256 KBytes of HPA space within I/O address space. As a GSC+ module, UTurn must respond to addresses in this region. This region is programmed by PDC and the operating system as part of the system configuration process. GSC+ hard physical addresses to architected I/O registers take the following form:

Runway addr bits	0 11	12 21	22 27	28 33	34 37	38 39
GSC address bits	0 3	4 13	14 19	20 25	26 29	30 31
	ones	flex	111111	regset	offset	00

where: ones indicates an I/O address.
flex indicates GSC+ port to which this IOA is connected. (This field is programmed by PDC and OS.)
111111 corresponds to the fixed field which is UTurn's location on the GSC+ port. Since UTurn is always host of the GSC+ bus, this field is hardwired to 6'b111111.
regset indicates the register set within the HPA. The register sets are as follows:

Supervisory Register Set:	0
Auxiliary Register Set	1
UTurn RAM Test Register Set 1	2'd16
UTurn RAM Test Register Set 2	2'd17
UTurn RAM Test Register Set 3	2'd18
UTurn RAM Test Register Set 4	2'd19
UTurn RAM Test Register Set 5	2'd20
UTurn RAM Test Register Set 6	2'd21
UTurn RAM Test Register Set 7	2'd22
GSC+ Bus Specific Register Set	2'd30
GSC+ Performance Ctr Register Set	2'd31

offset indicates the address offset within a register set of a given register. Registers are described in more detail in the following sections.

In order for UTurn to recognize this address region, PDC must configure the GSC+ flex field to reside within one of UTurn's sets of IO_IO_LOW and IO_IO_HIGH registers.

2.2.1. GSC+ Supervisory Register Set (Register Set 0)

The following registers comprise the GSC+ Supervisory Register Set:

Offset	Access Modes	Class	Name	LSB Address
2	R	A	IO_DC_DATA	.. 008
2	W	A	IO_DC_ADDRESS	.. 008
13	R	A	IO_STATUS	.. 034
14	RW	A	IO_CONTROL	.. 038

2.2.1.1. GSC+ IO_DC_DATA Register

The IO_DC_DATA register is used to read an IOA's IODC. The architecturally specified locations of IODC are defined as follows:

Byte Address	Name	Description
0 – 1	IODC_HVERSION	Hardware version number
2	IODC_SPA	Soft physical address capability
3	IODC_TYPE	Type of module
4 – 7	IODC_SVERSION	Software version number

These bytes are hardwired, as follows:

0 4	5 11	12 15	16	17	18	19 23	24	25	26	27 31
bus=01010	hvmod=0000001	hv_rev	io=0	0	sv=0	shift=0	mr=0	wd=1	R	type=00111

32 35	36 55	56 58	59	60 63
sv_rev	svmod = 5'h0000C	R	mc=0	R

where bus is set to an encoded value to indicate GSC+. This value is 5'b01010.
hvmod specifies the module hardware implementation. This value is 7'b0000001.
hv_rev indicates hardware versions. For the first release, this value is 4'b0001, matching the initial production revision of the U2 chip.
io set to 0 to indicate that SPA space is in I/O address space.
0 is simply hardwired to 0.
sv indicates if the following shift field is valid. This value is set to 0.
shift specifies the maximum SPA space size in bytes. This value is set to 0.
mr more bit is set to 0 to indicate that only the first 8 bytes of IODC are implemented.
wd word bit is set to 1 to indicate that a full word of address justified data is provided on reads from IO_DC_DATA.
R reserved. Reserved fields are always 0 when read.
type specifies the module type. This value is set to 5'b00111 to indicate TP_BCPORT.
sv_rev indicates feature enhancements. For UTurn, this value is 4'b0001.
svmod specifies software interface to module. This value is set to 5'h0000C.
R reserved. Reserved fields are always 0 when read.
mc module category. This value is set to 0.
R reserved. Reserved fields are always 0 when read.

2.2.1.2. GSC+ IO_DC_ADDRESS Register

The IO_DC_ADDRESS register specifies which 32 bit word of an IOA's IODC is to be returned from the IO_DC_DATA register. IO_DC_ADDRESS need only be a single bit register corresponding to bit 29 of the 32 bit address passed to the GSC port, as shown below:

0 28	29	30 31
undefined	wd	undefin

If IO_DC_ADDRESS[wd] is set to 1, reads from IO_DC_DATA will return IO_DC_DATA[32:63]. If IO_DC_ADDRESS[wd] is set to 0, reads from IO_DC_DATA will return IO_DC_DATA[0:31].

This register powers up with bit 29 being set to 0. Its value is unchanged as a result of a CMD_Reset or a CMD_Clear.

2.2.1.3. GSC+ IO_STATUS Register

The GSC+ IO_STATUS register has the following format:

0 11	12 14	15	16 21	22	23	24	25	26 27	28 31
HV	R	HV	estat	se	he	fe	ry	R	1000

The four least significant bits indicate that this is a lower port (GSC+) register, and that the remote port bus (Runway) has power available. Since both the upper and lower port busses are powered off of the same supply, it is safe to assume that if UTurn has power, then the upper port has power.

The ready bit (ry) has no significance for the GSC+ side of UTurn, since there is no GSC+ IO_COMMAND register. Its value will always be set to 1'b1.

The se (soft error), he (hard error), and fe (fatal error) bits indicate the severity of a detected error. UTurn does not implement the se bit; its value is always 0 when read. The estat field is encoded to indicate error type. Values for this field are presented in the error handling section, along with more discussion of the errors in general.

This register is set to X'00000048 at power–on or after a CMD_Reset is issued to UTurn. This register is unaffected by CMD_Clear.

2.2.1.4. GSC+ IO_CONTROL Register

The GSC+ IO_CONTROL register, which controls the forwarding of transactions from GSC+ to Runway, has the following format:

0 15	16 21	22	23 24	25 31
HV	R	HV	mode	reserved

The mode field indicates the behavior of the GSC+ block when processing GSC+ transactions bound for Runway:

Mode Name	Value	Definition
Off	0	Opaque.
Exclude	2	Transparent.

The GSC+ IO_CONTROL Register defaults to Off mode at power–on or when a Runway CMD_Reset is received. In this mode, GSC+ bus arbitration is disabled for guests (guest bus requests are ignored).

In Exclude mode, all addresses are forwarded. This is considered the normal operating mode for the GSC+ block. Because all transactions mastered by guests are forwarded, guest–to–guest transactions cannot be supported.

Since the GSC+ IO_CONTROL register impacts the GSC+ master operation, it resides in the GSC+ clock domain of an IOA. It is accessed by PDC and the OS during configuration. Writes to and reads from this register are queued, resulting in no out of order execution.

The power–on and CMD_Reset value of the GSC+ IO_CONTROL register is X'00000000 (mode = off). This register is unaffected by CMD_Clear.

2.2.2. GSC+ Auxiliary Register Set (Register Set 1)

The following registers comprise the auxiliary register set:

Offset	Access Modes	Class	Name	Address LSB
0	R	A	IO_ERR_RESP	.. 040
1	R	A	IO_ERR_INFO	.. 044
2	R	A	IO_ERR_REQ	.. 048

2.2.2.1. GSC+ IO_ERR_RESP Register

This register indicates the module which was responsible for responding to a GSC+ operation which failed. The slave address of the transaction is stored. The format of this register is as follows:

0	31
sys–resp–address	

where sys–resp–address is always a 32 bit GSC+ address.

The data contained in this register is not necessarily valid. Validity is indicated by the rsi bit in the GSC+ IO_ERR_INFO register.

This register is undefined at power–on and is unaffected by a CMD_Clear or CMD_Reset. A write to this register has no effect.

2.2.2.2. GSC+ IO_ERR_INFO Register

This register indicates the validity of the GSC+ IO_ERR_RESP and IO_ERR_REQ registers. The format of this register is as follows:

0 6	7	8 13	14 19	20	21 29	30	31
undefined	0	R	undefined	0	R	rsi	rqi

If rqi is 0, the contents of the IO_ERR_REQ register are valid and if rsi is 0, then contents of the IO_ERR_RESP register are valid. All undefined or reserved fields are zero when read.

This register is set to its default value of x'00000003 at power–on or after a CMD_Reset to the Runway IO_COMMAND register is issued. This register is unaffected by CMD_Clear. A write to this register has no effect.

2.2.2.3. GSC+ IO_ERR_REQ Register

This register logs the HPA of the requester of an erroneous GSC+ operation. The format of this register is as follows:

0 3	4 13	14 19	20	21 31
1111	System Requester Flex	Sys Req Fixed	rm	HV

The value of the System Requester Flex is set to either the Runway Flex value (which is hardwired) if this I/O Adapter is a GSC+ master of the transaction (on behalf of a Runway client) or the GSC+ Flex value if this I/O Adapter is a GSC+ slave of the transaction.

If this I/O Adapter received this erroneous transaction as a Runway slave, then the fixed field is obtained from the outbound command queue and is set as follows:

14	15	16 18	19
1	0	master_id	0

where master_id corresponds to the Runway master of the failing transaction.

If this I/O Adapter is a GSC+ slave of the transaction, then the fixed field is set as follows:

14	15 17	18 19
0	guest_id	00

where guest_id corresponds to the GSC+ guest master of the failing transaction. Note that this is not the entire fixed field of a GSC+ guest. Bits 18 through 19 typically correspond to the GSC+ guest submodule. Unfortunately, UTurn does not have visibility of the submodule.

The rm bit indicates whether the requestor is on the remote bus (Runway). If this bit is set to 1, then the system requestor fixed and flex fields are set to Runway values. If this bit is a 0, then the system requestor fixed and flex fields specify the GSC+ requestor of the erroneous transaction.

The data contained in this register is not necessarily valid. Validity is indicated by the rqi in the GSC+ IO_ERR_INFO register.

This register is undefined at power–on and is unaffected by a CMD_Clear or CMD_Reset. A write to this register has no effect.

2.2.3. UTurn Read Return RAM Register Sets

The following tables list the 104 registers which allow access to the UTurn Read Return RAM for test purposes. The 104 registers are broken up into 7 separate register sets, but this is a formality; the register addresses are sequential. The registers are read only, and provide the contents of the appropriate Read Return RAM (RRR) location as shown in the tables. The tables only show the details of a few registers; the remainder of the table is left as not much af an exercise to the reader. These registers are provided as a testability feature. Their usefulness is described in more detail in the testability chapter of this document. The register values are undefined at power–on and are unaffected by a CMD_Clear or CMD_Reset.

2.2.3.1. UTurn Read Return RAM Register Set 1 (Register Set 16)

Offset	Access Modes	Class	UTurn RdRtnRAM Register Name	LSB Address
0	R	HV	RdRtnRAM[0]	.. 400
1	R	HV	RdRtnRAM[1]	.. 404
.	.	.	.	.
.	.	.	.	.
15	R	HV	RdRtnRAM[15]	.. 43C

2.2.3.2. UTurn Read Return RAM Register Set 2 (Register Set 17)

Offset	Access Modes	Class	UTurn RdRtnRAM Register Name	LSB Address
0	R	HV	RdRtnRAM[16]	.. 440
1	R	HV	RdRtnRAM[17]	.. 444
.	.	.	.	.
.	.	.	.	.
15	R	HV	RdRtnRAM[31]	.. 47C

2.2.3.3. UTurn Read Return RAM Register Set 3 (Register Set 18)

Offset	Access Modes	Class	UTurn RdRtnRAM Register Name	LSB Address
0	R	HV	RdRtnRAM[32]	.. 480
1	R	HV	RdRtnRAM[33]	.. 484
.	.	.	.	.
.	.	.	.	.
15	R	HV	RdRtnRAM[47]	.. 4BC

2.2.3.4. UTurn Read Return RAM Register Set 4 (Register Set 19)

Offset	Access Modes	Class	UTurn RdRtnRAM Register Name	LSB Address
0	R	HV	RdRtnRAM[48]	.. 4C0
1	R	HV	RdRtnRAM[49]	.. 4C4
.	.	.	.	.
.	.	.	.	.
15	R	HV	RdRtnRAM[63]	.. 4FC

2.2.3.5. UTurn Read Return RAM Register Set 5 (Register Set 20)

Offset	Access Modes	Class	UTurn RdRtnRAM Register Name	LSB Address
0	R	HV	RdRtnRAM[64]	.. 500
1	R	HV	RdRtnRAM[65]	.. 504
.	.	.	.	.
.	.	.	.	.
15	R	HV	RdRtnRAM[79]	.. 53C

2.2.3.6. UTurn Read Return RAM Register Set 6 (Register Set 21)

Offset	Access Modes	Class	UTurn RdRtnRAM Register Name	LSB Address
0	R	HV	RdRtnRAM[80]	.. 540
1	R	HV	RdRtnRAM[81]	.. 544
.	.	.	.	.
.	.	.	.	.
15	R	HV	RdRtnRAM[95]	.. 57C

2.2.3.7. UTurn Read Return RAM Register Set 7 (Register Set 22)

Offset	Access Modes	Class	UTurn RdRtnRAM Register Name	LSB Address
0	R	HV	RdRtnRAM[96]	.. 580
1	R	HV	RdRtnRAM[97]	.. 584
.	.	.	.	.
.	.	.	.	.
7	R	HV	RdRtnRAM[103]	.. 59C

2.2.4. GSC+ Bus Specific Register Set (Register Set 30)

The following registers comprise the GSC+ Bus Specific Register Set:

Offset	Access Modes	Class	GSC+ Bus Specific Reg Name	LSB Address
0	RW	BS	GSC+_TRANS_TIMEOUT	.. 780
1	RW	BS	GSC+_PEND_TIMEOUT	.. 784
3	RW	BS	GSC+_CONFIG	.. 78c
4	RW	BS	GSC+_WD_TIMEOUT	.. 790
5	RW	BS	GSC_PVT_OVERRIDE	.. 794
8	RW	BS	GSC1.5X_CONFIG	.. 7A0
10	RW	BS	GSC2X_CONFIG	.. 7A8

2.2.4.1. GSC+_TRANS_TIMEOUT Register

The GSC+_TRANS_TIMEOUT register indicates the length of time that the I/O Adapter will wait from the time it masters a transaction on GSC until the corresponding slave acknowledge (READYL, RETRYL, or PACKL) is asserted by a guest. For UTurn–mastered writes, the timeout counter begins the cycle after the last data cycle is driven; for reads, the counter begins the cycle after the address cycle. Because of implementation details, the value in this register actually corresponds roughly to the number of four–GCLK ”ticks” that can pass before a timeout occurs. Thus, a value of decimal 40 in this register

corresponds to a timeout of roughly 160 GCLKs. Another result of the implementation of this register is that there is a 2 to 10 cycle uncertainty added to the timeout indicated by the value in this register (after scaling). In general, if a transaction gets anywhere near the timeout value in length, there are significant system architecture problems. In any case, care should be taken to ensure that the value written into this register takes into account the 4X multiplication factor and, for small timeout values, the uncertainty cycles.

Because GSC+ can run in a range of frequencies, initialization of this register may need to take into account the bus frequency and possibly the type of connected GSC+ guests (bus bridges may have a longer latencies). It is the responsibility of PDC to initialize this register as part of the I/O configuration process.

The format of the GSC+_TRANS_TIMEOUT register is as follows:

0 15	16 31
0000000000000000	GSC+_TRANS_TIMEOUT

The GSC+_TRANS_TIMEOUT register is located in the GSC+ block. Both reads and writes to the register are queued resulting in no out of order execution.

Although a value of zero can be written to this register, zero–value timeouts are not recommended for normal system operation. Changing from a zero–value timout to a non–zero value may require two writes to the timeout register. A timeout error is highly likely with a zero timeout value. In general, do not write a zero value to this register.

This register has a power–on value of X'00000FFF (4095 decimal). For a 40 MHz GCLK, this corresponds to a timeout of 410 us after accounting for the 4X scaling factor and the uncertainty cycles; for a 32 MHz GCLK, the timeout would be 512 us. The maximum configurable value for this register is X'0000FFFF (65535 decimal). This corresponds to a maximum timeout value of 6.55 ms with a 40 MHz GCLK or 8.19 ms with a 32 MHz GCLK. The value in the GSC+_TRANS_TIMEOUT register is unaffected by a CMD_Reset or CMD_Clear.

2.2.4.2. GSC+_PEND_TIMEOUT Register

The GSC+_PEND_TIMEOUT register indicates the length of time that the I/O Adapter will wait, from the time a pended slave acknowledge (PACKL) occurs in response to a UTurn–mastered DIO read until the corresponding pended DIO read response is issued, before timing out. UTurn will not log an error until the timeout condition is encountered AND no GSC+ guests are arbitrating for GSC bus ownership AND UTurn is able to grant the bus to a guest (no OutQ or RRQ entries are valid, InQ is not full, etc). Because of implementation details, the value in this register actually corresponds to roughly the number of four–GCLK "ticks" that can pass before a timeout occurs. Thus, a value of decimal 40 in this register corresponds to a timeout of roughly 160 GCLKs. Care should be taken to ensure that the value written into this register takes into account this 4X multiplication factor.

Because GSC+ can run in a range of frequencies, initialization of this register may need to take into account the bus frequency and possibly the number and type of GSC+ guests (as bus bridges may have longer latencies). It is the responsibility of PDC to initialize this register as part of the I/O configuration process.

The format of the GSC+_PEND_TIMEOUT register is as follows:

0 15	16 31
0000000000000000	GSC+_PEND_TIMEOUT

The GSC+_PEND_TIMEOUT register is located in the GSC+ block. Both reads and writes to the register are queued resulting in no out of order execution.

Although a value of zero can be written to this register, zero–value timeouts are not recommended for normal system operation. Changing from a zero–value timout to a non–zero value may require two writes to the timeout register. A timeout error is highly likely with a zero timeout value. In general, do not write a zero value to this register.

This register has a power–on value of X'00000FFF (4095 decimal). For a 40 MHz GCLK, this corresponds to a timeout of 410 us after accounting for the 4X scaling factor and the uncertainty cycles; for a 32 MHz GCLK, the timeout would be 512 us. The maximum configurable value for this register is X'0000FFFF (65535 decimal). This corresponds to a maximum timeout value of 6.55 ms with a 40 MHz GCLK or 8.19 ms with a 32 MHz GCLK. The value in the GSC+_PEND_TIMEOUT register is unaffected by a CMD_Reset or CMD_Clear.

2.2.4.3. GSC+_CONFIG Register

This register is used to manipulate various parameters within the GSC+ port. The format and exact definition of this register is as follows:

0 11	12 13	14	15	16 21	22	23	24	25	26 31
R	FwdPCt	FwdP	U	Pulse Grant	RST	PAR	HE	DIO	Gst Arb Disable

where:

R	Reserved (all zeroes when read).
FwdPCt	These bits configure the frequency with which the DMA forward progress mechanism enabled by the FwdP bit will interrupt a DIO write stream and allow a DMA transaction to be mastered by a GSC guest. If FwdPCt is set to 2'b00, a DMA transaction will be allowed after every 4 OutQ entries processed. If FwdPCt is 2'b01, a DMA transaction is allowed after 8 OutQ entries are processed. If FwdPCt is 2'b10, a DMA transaction is allowed after 12 OutQ entries are processed. If FwdPCt is 2'b11, a DMA transaction is allowed after 16 OutQ entries are processed.
FwdP	When set to 1 (default), DMA forward progress mechanism is invoked to prevent long streams of DIO write transactions from holding off DMA traffic. If bit is 0, the forward progress mechanism is disabled, and UTurn's GSC arbiter will behave the same as the GSC arbiter in the U2 chip.
U	Unused, but implemented (value read is value written; default = 0)
Pulse Grant	Each bit in this field corresponds to a GSC bus grant bit (BGL[n]); bit 16 corresponds to BGL[0], bit 17 corresponds to BGL[1], and so on.). When a bit in this field is set, the corresponding BGL[n] bit will be asserted for only 1 GSC bus cycle when a grant is initially issued (that is, the grant will be pulsed). Grants corresponding to zeroed bits in this field (default) will be "level grants", as currently issued by the U2 chip.
RST	When set to 1, indicates this IOA's GSC+ RESETL pin is asserted for approximately 10 ms upon UTurn's receipt of a directed CMD_Reset on the upper (Runway) port.
PAR	When set to 1, disables parity checking on this IOA's GSC+ AD[] bus.
HE	When set to 1, this IOA will not go into hard error mode, although hard

	errors will still be detected and logged.
DIO	When set to 1, this IOA will not allow pended DIO read transactions.
Gst Arb Disable	Each bit in this field corresponds to an external bus request bit. (Bit 26 corresponds to BRL[0], bit 27 corresponds to BRL[1], and so on.). When a bit in this field is set, UTurn ignores the corresponding bus request input. These bits do not prevent guests from splitting UTurn–mastered transactions, so guest bus mastership is not prevented in all cases.

The characteristics of UTurn that are controlled by the GSC+_CONFIG register are characteristics that should not change during normal system operation. It is expected that the GSC+_CONFIG register will be initialized by PDC and/or operating system "boot" code, and that the value in this register will not be changed during subsequent system operation. Changes to the value contained in this register that occur while other system/bus traffic is present can result in odd or errant UTurn operation. Before changing the value in this register, all bus activity on both sides of UTurn should be halted, and all transaction traffic internally queued in UTurn should be flushed. Extreme care should be exercised when altering the contents of this register after boot.

The GSC+_CONFIG register is located in the GSC+ block. Reads and writes to this register are queued resulting in no out of order execution.

The power–on and CMD_Reset value of this register is all zeroes. It is unaffected by a CMD_Clear.

2.2.4.4. GSC+_WD_TIMEOUT Register

The GSC+_WD_TIMEOUT register contains the number of GSC+ GCLK cycles that the I/O Adapter will wait when there appears to be a hang on the GSC+ bus – that is, when a guest owns the bus without mastering a transaction. Hang conditions can occur when an external bus grant occurs and the guest fails to master a transaction or when a guest has issued a transaction and UTurn fails to assert READYL. Because GSC+ can run in a range of frequencies, this register may require initialization based on the bus frequency and possibly the type and number of GSC+ guests (as bus bridges may have longer latencies). It is the responsibility of PDC to initialize this register as part of the I/O configuration process.

The format of the GSC+_WD_TIMEOUT register is as follows:

0 11	12 31
000000000000000000	GSC+_WD_TIMEOUT

The GSC+_WD_TIMEOUT register is located in the GSC+ block. Both reads and writes to the register are queued resulting in no out of order execution.

Although a value of zero can be written to this register, zero–value timeouts are not recommended for normal system operation. Changing from a zero–value timout to a non–zero value may require two writes to the timeout register. A timeout error is highly likely with a zero timeout value. In general, do not write a zero value to this register.

This register has a power–on value of X'0000FFFF (65535 decimal). For a 40 MHz GSC GCLK, this corresponds to a time of 1.64 ms; for a 32 MHz GCLK, the timeout is 2.05 ms. The maximum value for this register, X'000FFFFF, corresponds to a time of 26 msec for a 40 MHz GSC GCLK, or 33 msec for a 32 MHz GCLK. The value in the GSC+_WD_TIMEOUT register is unaffected by CMD_Reset or CMD_Clear.

2.2.4.5. GSC_PVT_OVERRIDE Register

The GSC_PVT_OVERRIDE register contains information related to the electrical configuration of the GSC pads. The GSC pads are designed to automatically adjust their output drive strength to compensate for UTURN process, voltage, and temperature (PVT) variations. By reading this register, the numerical value assigned to the GSC bus drive control circuit by the PVT autoconfiguration logic can be determined, and convergeance status can be obtained. By writing to this register, the bus drive control circuit can be manually overridden. Given the newness of this pad design, it was thought prudent to include a software mechanism to override the auto–configuration in case of unexpected behavior. For chip testing purposes, all pad drivers can be disabled.

The format of the GSC_PVT_OVERRIDE register is as follows:

0 2	3	4 6	7	8 10	11 15	16 22	23	24 26	27 31
000	noconverge	000	pvt_test	000	calc_pvt_val	0000000	override_en	000	override_val

The fields in the GSC_PVT_OVERRIDE register data word are defined as follows.

Field name	Field description
noconverge	When set, indicates that the PVT circuit failed to converge when attempting to match the external reference impedance, and that override_val is being used to configure the PVT circuit in the GSC pads. (If noconverge is set, it may be due to a missing resistor on UTurn's PVT sense pin). This is a read–only bit; default value is 0. This bit can be reset by setting the override_en bit, which causes the PVT autoconfiguration circuit to "try again".
pvt_test	When set, indicates that the base driver in the GSC pads is disabled; only intended use is for pad testing. This is a read/write bit; default value is 0 (base driver enabled).
calc_pvt_val	The value calculated by the PVT circuit for this IOA's GSC pads. This is a read–only field; writes to this field have no effect. There is no defined "default" value.
override_en	When set to 1, causes override_val to be sent to the GSC pads as the PVT circuit configuration value. This is a read/write bit; default value is 0 (override disabled, meaning that calc_pvt_val is sent to the pads to configure the PVT circuit). When override_en is set, the PVT autoconfiguration circuit operates continuously regardless of whether or not convergeance is achieved (the calc_pvt_val computed is not sent to the pads with override_en set, but the value can be read by software for testing or other purposes).
override_val	The 5–bit value to be used by this IOA's GSC pads to configure the PVT circuitry. A larger value in this field corresponds to greater pad output drive strength (larger FET sizes to drive signals onto bus wires). Value has no effect if override_en is 0 and noconverge is 0. This is a read/write field; default value is 5'b01101.

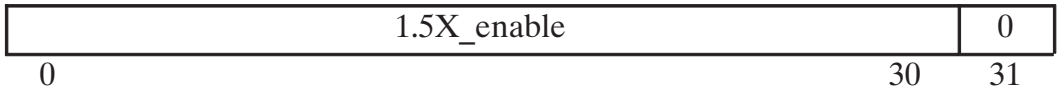
As with other GSC registers, the most significant bit of multi–bit fields are numbered 0. Thus, for example, calc_pvt_val[0] corresponds to GSC_PVT_OVERRIDE[11], and pvt_override_val[0] corresponds to GSC_PVT_OVERRIDE[27].

The GSC_PVT_OVERRIDE register is located in the GSC+ block. Both reads and writes to the register are queued resulting in no out of order execution.

The calc_pvt_val field will contain a hardware–dependent value after power–on (could be any 5 bit value). With this in mind, the entire register's power–on value is h'00??000D. The register's value is unaffected by CMD_Reset or CMD_Clear.

2.2.4.6. GSC1.5X_CONFIG Register

System software writes to UTurn's GSC1.5X_CONFIG register to indicate the individual 8–MByte I/O address spaces in which writes mastered by UTurn can be performed using the GSC variable length write transaction. It is necessary that system software determine the capabilities of the guests on a given GSC bus and correctly configure the GSC1.5X_CONFIG register in view of the guest capabilities. The format of the data written to UTurn's GSC1.5X_CONFIG register is as follows.



The fields in the GSC1.5X_CONFIG register data word are defined as follows.

Field name	Field description
1.5X_enable	Each bit in this field corresponds to an 8 MByte chunk of I/O address space. UTurn will master a variable−length write transaction on GSC if the destination address for a processor−mastered write falls within an 8 MByte I/O address space whose corresponding 1.5X_enable bit is set.

The 1.5X_enable bits correspond to I/O address space chunks as follows.

Bit #	I/O address range	Comments
0	F000 0000 − F07F FFFF	PDC space − UTurn will not coalesce accesses to space through F01F FFFF
1	F080 0000 − F0FF FFFF	Technically PDC space − use caution!
2	F100 0000 − F17F FFFF	General I/O space
3	F180 0000 − F1FF FFFF	General I/O space
4	F200 0000 − F27F FFFF	General I/O space
5	F280 0000 − F2FF FFFF	General I/O space
6	F300 0000 − F37F FFFF	General I/O space
7	F380 0000 − F3FF FFFF	General I/O space
8	F400 0000 − F47F FFFF	General I/O space
9	F480 0000 − F4FF FFFF	General I/O space
10	F500 0000 − F57F FFFF	General I/O space
11	F580 0000 − F5FF FFFF	General I/O space
12	F600 0000 − F67F FFFF	General I/O space
13	F680 0000 − F6FF FFFF	General I/O space
14	F700 0000 − F77F FFFF	General I/O space
15	F780 0000 − F7FF FFFF	General I/O space
16	F800 0000 − F87F FFFF	General I/O space
17	F880 0000 − F8FF FFFF	General I/O space
18	F900 0000 − F97F FFFF	General I/O space
19	F980 0000 − F9FF FFFF	General I/O space
20	FA00 0000 − FA7F FFFF	General I/O space
21	FA80 0000 − FAFF FFFF	General I/O space
22	FB00 0000 − FB7F FFFF	General I/O space
23	FB80 0000 − FBFF FFFF	General I/O space
24	FC00 0000 − FC7F FFFF	General I/O space
25	FC80 0000 − FCFF FFFF	General I/O space
26	FD00 0000 − FD7F FFFF	General I/O space
27	FD80 0000 − FDFF FFFF	General I/O space
28	FE00 0000 − FE7F FFFF	General I/O space
29	FE80 0000 − FEFF FFFF	General I/O space
30	FF00 0000 − FF7F FFFF	General I/O space

Note that the GSC1.5X_enable bit corresponding to the final 8 MByte range of I/O addresses is always 0. Broadcast address space falls within this final 8 MByte range. Variable length write transactions to

broadcast space are not supported by any GSC devices, and UTurn will not master variable length writes in this range.

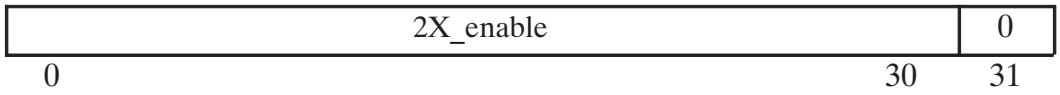
The whole concept of GSC 1.5X support is based on device characteristics that do not change over time – GSC guest capabilities, GSC bus configuration restrictions, and corresponding UTurn configuration requirements. It is expected that the GSC1.5X_CONFIG register will be initialized by PDC and/or operating system "boot" code, and that the value in this register will not be changed during normal system operation. Changes to the value contained in this register that occur while other system/bus traffic is present can result in errant UTurn operation. Before changing the value in this register, all bus activity on both sides of UTurn should be halted, and all transaction traffic internally queued in UTurn should be flushed. Realistically, this register should not be changed after boot.

The GSC1.5X_CONFIG register is logically located in the GSC+ HPA register map, but is physically located in the Runway clock domain of UTurn due to implementation considerations. As with all registers located in the Runway block, writes to this register occur immediately, whereas reads are queued.

The power–on value of this register is all zeroes. It is unaffected by CMD_Reset or CMD_Clear.

2.2.4.7. GSC2X_CONFIG Register

System software writes to UTurn's GSC2X_CONFIG register to indicate the individual 8–MByte I/O address spaces in which the data cycles of variable–length writes mastered by UTurn can be performed in a fast transfer mode, with data switching at 2X the normal GSC rate. It is necessary that system software determine the capabilities of the guests on a given GSC bus and correctly configure the GSC2X_CONFIG register in view of the guest capabilities. The general progression of this initialization is documented in a subsequent section. Note that value written to this register has implications for the value written to the GSC1.5X_CONFIG register. The format of the data written to UTurn's GSC2X_CONFIG register is as follows.



The fields in the GSC2X_CONFIG register data word are defined as follows.

Field name	Field description
2X_enable	Each bit in this field corresponds to an 8 MByte chunk of I/O address space. UTurn can master a variable−length write transaction if the destination address falls within an 8 MByte I/O address space whose corresponding 2X_enable bit is set.

The 2X_enable bits correspond to I/O address space chunks in the same fashion as the 1.5X_enable bits. See the description of the GSC1.5X_CONFIG register for details.

The GSC2X specification requires that all modules providing GSC2X capability (supporting variable–length writes with data in fast mode) must also support GSC1.5X capability (variable–length writes with data at normal bus rates). This means that if any bit is set in the GSC2X_CONFIG register, the corresponding bit should also be set in the GSC1.5X_CONFIG register. There may be bits set in the GSC1.5X_CONFIG register that are not set in the GSC2X_CONFIG register, because support of GSC1.5X capability does not imply support of GSC2X capability.

Note that the GSC2X_enable bit corresponding to the final 8 MByte range of I/O addresses is always 0 for the same reason that bit 31 of the GSC1.5X_CONFIG register is always 0 (broadcast space concerns).

The whole concept of GSC 2X support is based on device characteristics that do not change over time – GSC guest capabilities, GSC bus configuration restrictions, and corresponding UTurn configuration requirements. It is expected that the GSC2X_CONFIG register will be initialized by PDC and/or operating system "boot" code, and that the value in this register will not be changed during normal system operation. Changes to the value contained in this register that occur while other system/bus traffic is present can result in errant UTurn operation. Before changing the value in this register, all bus activity on both sides of UTurn should be halted, and all transaction traffic internally queued in UTurn should be flushed. Realistically, this register should not be changed after boot.

The GSC2X_CONFIG register is logically located in the GSC+ HPA register map, but is physically located in the Runway clock domain of UTurn due to implementation considerations. As with all registers located in the Runway block, writes to this register occur immediately, whereas reads are queued.

The power–on value of this register is all zeroes. It is unaffected by CMD_Reset or CMD_Clear.

2.2.5. GSC+ Performance Counter Register Set (Register Set 31)

The following registers comprise the GSC+ Performance Counter Register Set:

Offset	Access Modes	Class	GSC+ Perf Count Reg Name	LSB Address
0	RW	BS	GSC+_PERF_MASK_0	...7C0
1	RW	BS	GSC+_PERF_MASK_1	.. 7C4
4	RW	BS	GSC+_PERF_COMP_0	.. 7D0
5	RW	BS	GSC+_PERF_COMP_1	.. 7D4
8	RW0	BS	GSC+_PERF_COUNT_0	.. 7E0
9	RW0	BS	GSC+_PERF_COUNT_1	.. 7E4
15	RW	BS	GSC+_PERF_CONFIG	.. 7FC

In the following descriptions of each of the GSC+_PERF registers, all registers suffixed with 0 are considered a set and all registers suffixed with 1 are considered a set. As a set, the mask register and comp register combine to determine the occurrence of events which will affect the value of the corresponding count register. No register within a set can be affected by registers in the other set.

2.2.5.1. GSC+_PERF_MASK_0 and 1 Registers

The GSC+_PERF_MASK_n registers contains several fields which indicate whether a function is to be monitored by the performance counters. The format of the GSC+_PERF_MASK registers are as follows:

0 9	10 15	16	17 22	23	24	25	26	27 30	31
0000000000	BRL msk	UTrq	BGL msk	UTgt	Gmstr	LkSpl	AddvL	Type msk	DoRR

where:

BRL msk mask of GSC+ guest bus requests (BRL[0] corresponds to bit 10)

UTrq	mask of UTurn's internal bus request
BGL msk	mask of GSC+ guest bus grants (BGL[0] corresponds to bit 17)
UTgt	mask of UTurn's internal bus grant
Gmstr	mask of the logical "NAND" of all BGL lines and LSL
LkSpl	mask of inverted version of GSC+ LSL signal
AddvL	mask of GSC+ ADDVL signal
Type msk	mask of GSC+ TYPE[] bus
DoRR	mask of UTurn's internal "DMA Read Return in Progress" indicator

All fields are positive true. When a bit is set to 1, the value of the signal or condition corresponding to that bit is compared to the corresponding value in the GSC+_PERF_COMP_n register. If there is a match, then the GSC+_PERF_COUNT register is incremented. As an example, if it is desired to count the number of occurrences of assertion of GSC+ bus request by guest 3 (BRL[3] = 0), bit 20 of GSC+_PERF_MASK_n should be set to 1, and all other bits should be set to 0.

The GSC+_PERF_MASK_n registers is located in the GSC+ block. Both reads and writes to these registers are queued resulting in no out of order execution.

These registers have undefined values at power–on, and are unaffected by a CMD_Reset or CMD_Clear.

2.2.5.2. GSC+_PERF_COMP_0 and 1 Registers

The GSC+_PERF_COMP_n registers contains several fields which indicate the value of a function which is to be monitored by the performance counters. The format of the GSC+_PERF_COMP_n registers are as follows:

0 9	10 15	16	17 22	23	24	25	26	27 30	31
0000000000	BRL cmp	UTrq	BGL cmp	UTgt	Gmstr	LkSpl	AddvL	Type cmp	DoRR

where:

BRL cmp	value to be compared with GSC+ guest bus requests; 0 = guest requesting GSC+ (BRL[0] corresponds to bit 10, BRL[5] corresponds to bit 15)
UTrq	value to be compared with UTurn's internal bus req; 1=UTurn requesting GSC+
BGL cmp	value to be compared with GSC+ guest bus grants; 0 = grant to guest n (BGL[0] corresponds to bit 17, BGL[5] corresponds to bit 22)
UTgt	value to be compared with UTurn's internal bus grant; 1 = grant to UTurn
Gmstr	value to be compared with the logical "NAND" of all GSC+ BGL lines and the GSC+ LSL signal
LkSpl	value to be compared with an inverted version of the GSC+ LSL signal; 1 = LSL asserted by a guest
AddvL	value to be compared with GSC+ ADDVL signal; 0 = GSC+ ADDVL asserted
Type cmp	value to be compared with GSC+ TYPE[] bus (same sense as bus signal)
DoRR	value to be compared with UTurn's internal "DMA Read Return in Progress" indicator; 1 = Pended DMA read return in progress on GSC+ (DRRL asserted)

When a bit in the GSC+_PERF_MASK_n indicates that a particular signal or condition is to be monitored, the corresponding field in the GSC+_PERF_COMP_n register is compared with the activity on this IOA. Whenever there is a match, the GSC+_PERF_COUNT_n register is incremented.

The values to be compared for the BRL cmp, BGL cmp, and AddvL fields are low true values, matching the low true nature of the GSC+ signals BRL[], BGL[], and ADDVL respectively. Thus, to look for the condition "BRL[0] asserted", a value of 0 must be written to bit 10 of GSC+_PERF_COMP_n. Similarly, to count the condition "BGL[5] not asserted", a 1 must be written to bit 22 of GSC+_PERF_COMP_n.

As an example, assume that the condition to be counted is the number of 8 word (32 byte) write transactions mastered by GSC+ guest 1. To enable counting of this type of event, the GSC+_PERF_MASK_n register must have bits 18, 26, and 27–30 set to 1, enabling comparison on BGL[1], ADDVL, and TYPE[0:3] respectively. All other bits in GSC+_PERF_MASK_n should be set to 0. GSC+_PERF_COMP_n must the be configured such that bit 18 is set to 0 (corresponding to BGL[1] asserted), bit 26 set to 0 (corresponding to ADDVL asserted), and bits 27–30 set to 0111 (corresponding to the type code for an 8 word write). Other bits in GSC+_PERF_COMP_n can be set to either 1 or 0.

The GSC+_PERF_COMP_n registers is located in the GSC+ block. Both reads and writes to these registers are queued resulting in no out of order execution.

These registers have undefined values at power–on, and are unaffected by a CMD_Reset or CMD_Clear.

2.2.5.3. GSC+_PERF_COUNT_0 and 1 Registers

The GSC+_PERF_COUNT_n registers maintain the count of specific events on an IOA, as programmed by the GSC+_PERF_COMP/MASK_n registers and the GSC+_PERF_CONFIG register. The format of the GSC+_PERF_COUNT_n registers are as follows:

0	31
event count	

If the GSC+_PERF_CONFIG count mode is set to 0, then the GSC+_PERF_COUNT_n register increments whenever a bit in the GSC+_PERF_MASK_n indicates that a field is to be monitored and a comparison of the corresponding field in the GSC+_PERF_COMP_n register with the activity on this IOA yields a favorable result. It will also increment on every GSC+ GCLK if the count mode in the GSC+_PERF_CONFIG register equals 1.

The GSC+_PERF_COUNT_n registers is located in the GSC+ block. Both reads and writes to these registers are queued resulting in no out of order execution. Any write to either GSC+_PERF_COUNT register clears the register, regardless of the data value specified. A read of either register does not affect its value.

These registers have undefined values at power–on, and are unaffected by a CMD_Reset or CMD_Clear.

2.2.5.4. GSC+_PERF_CONFIG

The GSC+_PERF_CONFIG register affects both GSC+_PERF register sets 0 and 1. The format of the GSC+_PERF_CONFIG register is as follows:

0 2	3 5	6 30	31
count mode 0	count mode 1	R	count enb

where:

count mode 0	performance counter 0 mode
count mode 1	performance counter 1 mode
R	Reserved fields are always zeroes.
count enb	count enable (affects both sets of counters)

The performance counter modes are defined as follows:

0:	Count GCLK cycles during events determined by the corresponding mask and compare registers in this set.
1:	Count GCLK cycles
2:	Count events determined by this set's corresponding mask and compare registers
3–7:	Undefined

The count enb bit enables incrementing for the GSC+_PERF_COUNT registers. The count mode fields determine what is to be counted, according to the above description.

The GSC+_PERF_COUNT register is located in the GSC+ block. Both reads and writes to this register are queued resulting in no out of order execution.

The power–on and CMD_Reset value of the GSC+_PERF_CONFIG register is X'00000000. This register is unaffected as a result of a CMD_Clear.

2.3. Runway Broadcast Physical Address Space

Addresses in the range X'FFFC0000 through X'FFFFFFFF are called broadcast physical address (BPA) space. The first half of the BPA is considered the local broadcast register set. Issuing a write operation to a register in this range affects every module on Runway. The second region of BPA is considered the global broadcast register set. Issuing a write operation in this range affects every module on all busses. UTurn will not implement any global broadcast registers, and will therefore never recognize or forward global broadcast transactions, such as reset.

Broadcast physical 40–bit addresses to architected I/O registers take the following form:

0 21	22	23 27	28 33	34 37	38 39
1111.1111.1111.1111.1111.11	g	brd_pg	regset	offset	00

where:

1111...	indicates a broadcast I/O address.
g	set to 0 to indicate local broadcast space
brd_pg	indicates the page in the selected broadcast space. (Software must set to b'00000.)
regset	indicates the register set within the BPA. The register sets are as follows: Local Broadcast Register Set: 0 Runway Bus Specific Register Set: 30 decimal
offset	indicates the address offset of a given register. Registers are described in more detail in the following sections.

2.3.1. Runway Local Broadcast Register Set (Register Set 0)

The register map for the Runway Local Broadcast Register Set within the Runway broadcast physical address space is as follows:

Offset	Access Modes	Class	Local Register Name	LSB Address
8	W	A	Runway IO_FLEX	.. 020

2.3.1.1. Runway IO_FLEX Register

The format of the Runway IO_FLEX register is as follows:

0 3	4 13	14 30	31
1111	flex = 1111111110	00000000000000000	enb

The flex field in the Runway IO_FLEX register specifies the programmable portion of the I/O Adapter's HPA space on Runway. This value is hardwired. The IO_FLEX register also contains the enable bit (enb).

When the enable bit is set to 0, indicating disable, UTurn must not forward GSC+ transactions to Runway (unless they are read responses). However, asynchronous events (such as TOC) must be forwarded. Since there is a single inbound queue and read responses from HPA registers and PDC are critical, UTurn will disable arbitration for GSC+ guests. This will ensure that no new transactions (specifically non read returns) are generated on GSC+. GSC+ transactions issued prior to the disable are processed along with all read returns. In addition to read responses, UTurn may master the following transactions, when disabled:

- BroadErr — in response to a detected Runway control or address parity error
- DirErr — in response to a GSC+ slave_ack failure, GSC+ timeout, or GSC+ parity error on a UTurn mastered programmed I/O reads.
- CMD_Reset — in response to the TOC line being asserted true.
- PFW_Interrupt write — in response to PFail_Warning.L line being asserted.

Finally, in disable mode, UTurn will accept and process all Runway transactions, whether bound for GSC+, PDC, HPA, or BPA space.

The Runway IO_FLEX register is located in the inbound Runway block to facilitate comparisons to slave transactions.

On power–on, this register will have the value X'FFF80000. This register is unaffected as a result of a CMD_Reset or CMD_Clear.

2.3.2. Runway Bus Specific Register Set (Register Set 30)

The following register resides in the Runway Bus Specific Register Set in Runway broadcast physical address space:

Offset	Access Modes	Class	Runway Specific Register Name
8	W	BS	RUNWAY_TIMEOUT

2.3.2.1. RUNWAY_TIMEOUT Register

The RUNWAY_TIMEOUT register contains the minimum number of Runway cycles that the mastering module will wait for a response before it times out. At power–on, this register is initialized to 4096 cycles. As part of configuration, the RUNWAY_TIMEOUT register may be written by software with a value which more closely corresponds to the worst case response time. It is expected that 24 bits is the upper bound of what might be required for a timeout cycle count. The register format is therefore as follows:

0	31
RUNWAY_TIMEOUT	

The RUNWAY_TIMEOUT register is located in the Runway inbound block, since the timeout counter associated with this register must time Runway inbound read transactions. Due to its location, writes to this register will occur immediately, bypassing the queues, whereas reads are queued. Therefore, it is possible that a write to this register could bypass a queued read already in process. To avoid this, software should wait for a read response before issuing a subsequent write to RUNWAY_TIMEOUT.

The contents of this register will not change as a result of a CMD_Reset or a CMD_Clear. At power–on, the value of this register is X'00001000.

2.4. GSC+ Broadcast Physical Address Space

To write to GSC+ Broadcast Address Space, the Runway IO_CONTROL register must be placed in peek mode and the Runway address bits 0 through 7 must be ones, address bits 8 through 21 must compare to an IO_IO_LOW(_HV) and corresponding IO_IO_HIGH(_HV) field (as described in section 2.1.2.7, where bits 8–11 must equal 4'b1111 and bits 12–23 must be between the low/high range specified by bits 20–31 of the low/high registers), address bits 22 though 33 must be set zero, address bits 34 through 37 must be set to the register offset, and bits 38 and 39 are zero.

0 7	8 21	22	23 27	28 33	34 37	38 39
11111111	LOW <= field < HIGH	g	brd_pg	regset	offset	00

where: 1111... indicates an I/O address.
g set to 0 to indicate local broadcast space
brd_pg indicates the page in the selected broadcast space. (Software must set to b'00000.)
regset indicates the register set within the BPA. (Software must set to b'0000.)
offset indicates the address offset of a given register. Registers are described in more detail in the following sections.

When in peek mode, transactions in the address range specified by IO_IO_LOW and IO_IO_HIGH range are remapped to the local broadcast region before being forwarded to GSC+ (by setting the most significant 14 bits of the address to ones). This results in a GSC+ address with the following format:

0 13	14	15 19	20 25	26 29	30 31
1111.1111.1111.11	g	brd_pg	regset	offset	00

where: 1111... indicates a broadcast I/O address.
g set to 0 to indicate local broadcast space

brd_pg indicates the page in the selected broadcast space. (will always be set to b'00000.)
regset indicates the register set within the BPA. (This will always be set to b'00000.)
offset indicates the address offset of a given register. Registers are described in more detail in the following sections.

2.4.1. GSC+ Broadcast Register Set (Register Set 0)

These converted addresses are used to access the following register in the GSC+ block:

Offset	Access Modes	Class	Local Register Name
8	W	A	GSC+ IO_FLEX

2.4.1.1. GSC+ IO_FLEX Register

The format of the GSC+ IO_FLEX register is as follows:

0 3	4 13	14 30	31
1111	flex	00000000000000000	enb

The flex field in the GSC+ IO_FLEX register specifies the programmable portion of the I/O Adapter's HPA space for this GSC+ port. This value is loaded by PDC and the OS during system configuration. The IO_FLEX register also contains the enable bit (enb), which is necessary for enabling and disabling of GSC+ modules from arbitration for bus mastership.

When the enable bit is set to 0, indicating disable, UTurn must disable arbitration for GSC+ guests. However, as long as no hard or fatal errors are logged, UTurn will still drain the outbound and read return queues by mastering these transactions on GSC+. The Runway side of the I/O Adapter will not be affected.

The GSC+ IO_FLEX register is located in the GSC+ block to facilitate comparisons to queued transactions.

As with all Broadcast Physical Address Space registers, the GSC+ IO_FLEX is not readable. However, its contents are reflected in the Runway Hard Physical Address Space GSC+_SHADOW_FLEX register, which is readable.

On power–on, this register will have the value X'FFF80000. This register is unaffected as a result of a CMD_Reset or CMD_Clear.

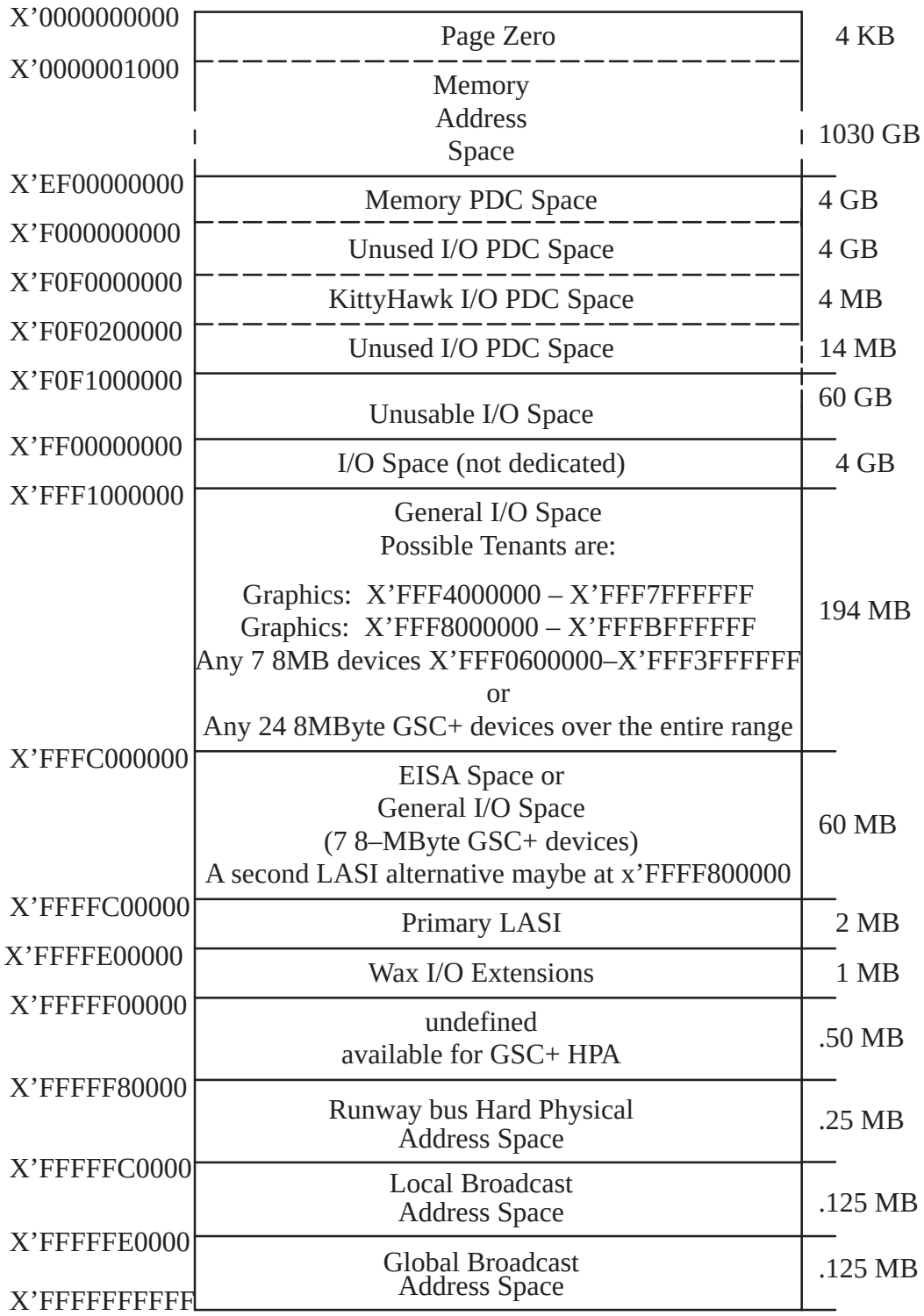
2.5. Address Decode Requirements

Each I/O Adapter within UTurn must recognize five distinct address spaces – Runway's hard physical address space, both regions of space bounded by the two sets of IO_IO_LOW and IO_IO_HIGH registers (which must encompass the GSC+ hard physical address space and the GSC+ broadcast physical space by facilitating the peek mechanism), Runway's broadcast physical address space, and PDC. To ensure that the system can be properly configured, the address spaces recognized by UTurn are prioritized to handle any case of overlap. The BPA space, HPA space, and PDC space are hardwired (thereby ensuring no overlap) and they take precedence over the region bounded by IO_IO_LOW and IO_IO_HIGH, which comes up uninitialized.

This address decode precedence is required to guarantee that a system can be reset and new addresses correctly assigned. The following initialization sequence is suggested as the basis for the address decode model:

1. A broadcast IO_FLEX transaction is issued over Runway, with the enable bit cleared, limiting the I/O Adapters' ability to master a transaction on Runway.
2. A CMD_Reset is written to the IO_COMMAND in the Supervisory Register Set of every module on Runway, putting the GSC+ IO_CONTROL Register in off mode so that transactions are not forwarded to the GSC+ bus.
3. One set of IO_IO_LOW and IO_IO_HIGH registers are opened, permitting access to the GSC+ IO_FLEX register.
4. The Runway IO_CONTROL Register is put into peek mode, permitting writes to the GSC+ IO_FLEX Register.
5. The GSC+ IO_FLEX Register is loaded disabling the GSC+ guests from bus mastership.
6. The Runway IO_CONTROL Register is placed into include mode and the GSC+ IO_CONTROL Register is placed into exclude mode, allowing access to the GSC+ guests.
7. The GSC+ bus is "walked", meaning that the GSC+ guests are polled to determine their address region requirements and they are reset.
8. Finally, the IO_IO_LOW and IO_IO_HIGH registers are loaded with values reflecting the address region required by the GSC+ guests.

2.6. Address Space Partitioning



2.7. Architected I/O Writes

UTurn must issue writes to the Runway Specific Register named PFW_INTERRUPT which is implemented by processors. I/O Adapter0 on UTurn_0 is connected to the PFAIL_WARNING.L signal and it must generate a Runway WRITE_SHORT to the PFW_INTERRUPT register located at address X'FFFFFC0780. This write to the PFW_INTERRUPT register will generate a powerfail interruption in processors. The PFAIL_WARNING.L signal must be detected by the Runway inbound block which will then generate the transaction. UTurn will not assert ResetL on GSC+ when PFAIL_WARNING.L is detected.

Other architected writes include writes to the monarch processor IO_EIR on behalf of a GSC+ InterruptL assertion and writes to the monarch processor SRS IO_COMMAND on behalf of a TOC. The interrupt is received by each I/O Adapter on behalf of its GSC+ bus and queued in the inbound queue so that it is processed in the order it is received. All TOC assertions are only received by the I/O Adapter with a client_id encoding of 4. TOC assertions will bypass the queue (with the exception of any pending write_backs from the cache) and be immediately issued to Runway. This is to ensure its execution, since TOCs are sometimes issued by the console port user in the event of a hung system. More detail regarding TOCs and interrupts are provided in the UTurn Specific Register Set section of this chapter.

GeckoBoa (which is a GSC+ guest) may issue a Directed Command Reset to a processor on behalf of an NIO client. UTurn will forward this transaction without manipulating the address. The assumption is that the NIO client can be informed of the monarch processor location programmatically.

2.8. Error Handling

UTurn detects three levels of error severity:

- soft error – Does not damage architectural state or operation of the I/O Adapter, but the error may not be corrected.
- hard error – Damages the architectural state of the I/O Adapter and may compromise its operation, but the extent of the damage is known and contained. If the hard error was detected on the Runway side (upper port of the IOA) then the I/O Adapter must discard subsequent writes from GSC+ guests and Runway clients and return PATH_ERROR (with corresponding master_id and trans_id) on subsequent reads from Runway and GSC+ clients, except those reads from HPA of the Runway port or reads from PDC. If the hard error was detected on the GSC+ side (lower IOA port) then the I/O Adapter must discard subsequent writes from GSC+ guests and Runway clients and return PATH_ERROR (with correspon–ding master_id and trans_id) on subsequent reads from Runway and GSC+ clients, except those reads from HPA of the Runway or GSC+ ports or reads from PDC. The I/O Adapter cannot master transactions and must be CMD_Reset by software.
- fatal error – Damages the architectural state of the I/O Adapter and the extent of the damage is not known and may not be contained. If the fatal error was detected on the Runway side (upper port of the IOA) then the I/O

Adapter must discard subsequent writes from GSC+ guests and Runway clients and return PATH_ERROR (with corresponding master_id and trans_id) on subsequent reads from Runway and GSC+ clients, except those reads from HPA of the Runway port or reads from PDC. If the fatal error was detected on the GSC+ side (lower port of the IOA) then the I/O Adapter must discard subsequent writes from GSC+ guests and Runway clients and return PATH_ERROR (with corresponding mas–ter_id and trans_id) on subsequent reads from Runway and GSC+ clients, except those reads from HPA of the Runway or GSC+ ports or reads from PDC. The I/O Adapter cannot master transactions and must be CMD_Reset by software.

In the previous definitions, a Runway PATH_ERROR is the simultaneous assertion of ADDR_VALID and DATA_VALID on Runway and a GSC+ PATH_ERROR is the assertion of ERRORL on GSC+.

When a hard or fatal error occurs on either the Runway side or GSC+ side of an IOA, the GSC+ port will deviate slightly from the above architected definition. Instead of discarding writes and failing to slave_ack reads on hard or fatal errors, the GSC+ port will simply disable GSC+ arbitration, effectively disallowing any further transactions. There is one exception to this approach: Should a GSC+ guest split a UTurn mastered transaction and gain ownership of the GSC+ bus even though a hard or fatal error is logged, UTurn must continue to grant them the bus. In this case, UTurn will drop any GSC+ guest mastered writes and assert errorl in response to any GSC+ guest mastered reads.

The I/O Adapter must log errors in one of the two IO_STATUS registers, along with an estat field which indicates the type of failure. The GSC+ and Runway blocks will log copies of the se, he, and fe bits in their own copies of the IO_STATUS registers. However, when one port (Runway or GSC+) goes hard or fatal, so does the other port. So, although the corresponding error bit will not be set in both register sets, both ports will behave as if they were in hard or fatal error mode. This approach allows for the implementation of distinct error logging registers on the GSC+ and Runway blocks which can then be uniquely addressed for subsequent interrogation.

The remainder of this section details the transaction type, possible errors, and subsequent I/O Adapter behavior. Throughout this discussion, the following error types are discussed:

- Unexpected Runway Response — I/O Adapter receives a read return with its master id, but the transaction id does not correspond to any transaction currently in progress.
- Runway BroadError Trans — UTurn receives a BROADCAST_ERROR transaction.
- Runway Control Parity Error — a control parity error detected by UTurn.
- Runway Address Parity Error — an address parity error detected by UTurn.
- Runway Data Parity Error — a data parity error detected by UTurn.
- Runway Mode Phase Error — Runway write_short is received, but there are no subsequent data cycles.
- Runway timeout — Failure of a Runway guest to issue a read response within the timeout window. Note: UTurn will not detect directed_errors. Therefore, read responses of this nature will also result in Runway timeouts.

• Incompatible transaction size	A Runway read return was in error because the size of the response is less than that of the request. If more data is expected and instead an address cycle occurs, this new transaction may not be handled properly. (UTurn won't detect the case of the master_id or trans_id changing in the midst of a data return.)
• Runway PATH_ERROR	I/O Adapter detects an assertion of ADDR_VALID and DATA_VALID.
• TLB fault	I/O Adapter is in ERROR (or direct insert) mode and detects a TLB fault. (When in this mode, software is expected to directly insert TLB entries.) Or I/O Adapter is in NORMAL (or fetch on miss) mode and the fetched IO PDIR entry is invalid. (When in this mode, software is expected to load the IO PDIR with valid entries.)
• GSC+ improper access	GSC+ transaction is inconsistent with the target address (such as a DMA read from an I/O address or a GSC+ write of greater than 1 word to an I/O address).
• GSC+ internal error	UTurn's GSC+ control circuitry encountered a condition that indicates either a bus arbitration violation or an internal logic error, resulting in both UTurn and a guest device "owning" the GSC bus at the same time.
• GSC+ protocol error	GSC+ guest masters a transaction with incorrect protocol, such as a two word read or write from an odd word address.
• GSC+ slave acknowledge	GSC+ guest fails to assert READYL, PACKL, or RETRYL, resulting in a connected transaction timeout.
• GSC+ pend timeout	failure of a GSC+ guest to issue a response on a pended DIO read within time limits, resulting in a pended transaction timeout.
• GSC+ watchdog timeout	GSC+ guest owns the bus (either due to explicit grant or assertion of LSL to split), but no transaction has been mastered within the timeout window.
• GSC+ Parity Error	I/O Adapter detects invalid parity on GSC+ address or data cycle
• GSC+ assertion of ERRORL	ERRORL was asserted by guest and detected by I/O Adapter (Note: that the GSC+ spec requires that ErrorL be asserted on the second cycle following the cycle in error, allowing UTurn to associate the error with a particular transaction.)
• Illegal GSC+ Response	A GSC+ pended DIO read return was in error because the size of the response did not correspond to the request.

- Unexpected GSC+ Response — A GSC+ pended DIO read return was mastered by a guest, but no pended DIO read was outstanding.
- GSC Error Transaction — A GSC+ guest has issued a GSC Error transaction type to indicate a failure. (Currently GeckoBoa is the only GSC module that issues this transaction type, which is also referred to as a GeckoBoa TOC transaction. However, this reference is discouraged, as it leads to some confusion with UTurn's TOC signal.)

Errors can be subdivided into three categories, those which are generic to a transaction (Unexpected Runway Response, Runway BroadError Transaction, GSC error transaction, and GeckoBoa TOC Assertion), those which are specific to Runway (the next ten listed above), and those which are specific to GSC+ (the remaining from the list above). In all of the following tables, the severity and estat are recorded in either the Runway block or the GSC+ block IO_STATUS register, depending on which block detected the error. Therefore, if the Runway IO_ERR_REQ and IO_ERR_RESP are valid, then the sev and estat field are stored in the Runway IO_STATUS register. When the table indicates that an IO_ERR_REQ or IO_ERR_RESP register is "not relevant", then this means that this register contents has not changed as a result of this error. When the table indicates that an IO_ERR_REQ or IO_ERR_RESP register is "not valid", then this means that the register's contents does not reflect this error and this must be indicated in the IO_ERR_INFO register. The contents of these registers may very well be valid, but they indicate the requester and responder of a previously detected error on that side of UTurn.

The Runway and GSC+ specific errors are discussed as they relate to specific transaction types – Runway to GSC+ writes, Runway to GSC+ reads, GSC+ to Runway reads, and GSC+ to Runway writes – in the following tables. Please note the definition of the following terms, which are used in these tables:

runway master	the Runway Flex and Master_ID
rw slave addr	the Runway 40bit real physical address (if logged in RW IO_ERR_RESP) the least significant 32 bits of the Runway real physical address (if logged in GSC+ IO_ERR_RESP)
gsc+ master	the GSC+ Flex and Guest Module ID (Submodule ID not available)
gsc+ slave adr	the I/O Virtual Page Number and word offset issued on the GSC+ bus
errorl	GSC+ ERRORL assertion
broaderr	IOA drives broadcast error transaction on Runway
dir_err	IOA drives directed error transaction on Runway

The following table illustrates I/O Adapter behavior resulting from a generic error.

Error	Sev	Signal	estat	rw req	rw resp	gsc+ req	gsc+ resp
unx rw resp	se	–	50	not valid	not valid	not relevant	not relevant
broaderror	he	–	13	not valid	not valid	not relevant	not relevant
gsc err tran	he	–	16	not relevant	not relevant	gsc+ master	not valid
un gsc resp	se	errorl	50	not relevant	not relevant	gsc+ master	not valid
gsc internal error	he	–	2	not relevant	not relevant	gsc+ master	gsc+ slave adr

On writes mastered by a processor to the I/O Adapter, PDC, or a slave GSC+ guest, (referred to as the ”Write responder on Runway, requestor on GSC+” in the KittyHawk Error Strategy) the following errors are possible:

Error	Sev	Signal	estat	rw req	rw resp	gsc+ req	gsc+ resp
ctl_par	fe	broaderr	5	runway master	rw slave addr	not relevant	not relevant
addr_par	fe	broaderr	5	runway master	rw slave addr	not relevant	not relevant
data_par	fe	–	5	runway master	rw slave addr	not relevant	not relevant
rw mode ph	fe	–	3	runway master	rw slave addr	not relevant	not relevant
gsc+ slave ack	he	errorl	7	not relevant	not relevant	runway master	rw slave adr
gsc+ errorl	he	–	4	not relevant	not relevant	runway master	rw slave adr

- On Runway improper accesses, writes to:

Runway unimplemented HPA	discarded / no error logged
Runway read_only HPA	discarded / no error logged
GSC+ unimplemented HPA	discarded / no error logged
GSC+ write_only HPA	discarded / no error logged

On reads mastered by a processor to the I/O Adapter, PDC, or a slave GSC+ guest, (referred to as the ”Read responder on Runway, requestor on GSC+” in the KittyHawk Error Strategy) the following errors are possible:

Error	Sev	Signal	estat	rw req	rw resp	gsc+ req	gsc+ resp
ctl_par	fe	broaderr	5	runway master	rw slave addr	not relevant	not relevant
addr_par	fe	broaderr	5	runway master	rw slave addr	not relevant	not relevant
gsc+ slave ack	se	dir_err errorl	7	not relevant	not relevant	runway master	rw slave adr
gsc+ pend timeout	se	dir_err errorl	59	not relevant	not relevant	runway master	rw slave adr
gsc+ par	se	dir_err errorl	53	not relevant	not relevant	runway master	rw slave adr
gsc+ errorl	he	–	4	not relevant	not relevant	runway master	rw slave adr
ill gsc+ rsp	se	dir_err errorl	50	not relevant	not relevant	runway master	rw slave adr

- On Runway improper accesses, reads from:

Runway unimplemented HPA	hv return / no error logged
Runway write_only HPA	hv return / no error logged
Runway BPA	timeout / no error logged
GSC+ unimplemented HPA	hv return / no error logged
GSC+ write_only HPA	hv return / no error logged
GSC+ BPA	path error / no error logged

where: hv return — means the value of the data word returned is hardware version dependent.

timeout — means that the IOA does not issue a response to this read, resulting in a Runway timeout.

On writes mastered from a GSC+ guest to memory or a processor, (referred to as the ”Write requestor on Runway, responder on GSC+” in the KittyHawk Error Strategy) the following errors are possible: (note that the GSC error transaction is considered in the generic errors table)

Error	Sev	Signal	estat	rw req	rw resp	gsc+ req	gsc+ res
ctl_par (data cycle)	fe	broaderr	53	runway master	not valid	not relevant	not relevant
ctl_par (addr cycle)	fe	broaderr	53	runway master	rw slave addr	not relevant	not relevant
addr_par	fe	broaderr	53	runway master	rw slave addr	not relevant	not relevant
TLB fault	he	–	61	gsc+ master	gsc+ slave adr	not relevant	not relevant
gsc impr ac	he	–	62	gsc+ master	gsc+ slave adr	not relevant	not relevant
gsc+ par	he	errorl	5	not relevant	not relevant	gsc+ master	gsc+ slave adr
gsc+ errorl	he	–	52	not relevant	not relevant	gsc+ master	gsc+ slave adr
gsc+ proto	he	errorl	54	not relevant	not relevant	gsc+ master	gsc+ slave adr
gsc+ wdog	he	errorl	17	not relevant	not relevant	gsc+ master	not valid

On reads mastered from a GSC+ guest to memory or a processor, (referred to as the ”Read requestor on Runway, responder on GSC+” in the KittyHawk Error Strategy) the following errors are possible:

Note: The first 9 errors listed below can also be encountered when the GSC+ guest has specified a clear16 or a partial cache line write. Since these operations require UTurn to master a read_priv on Runway followed by the specified write, the following Runway read errors are possible.

Error	Sev	Signal	estat	rw req	rw resp	gsc+ req	gsc+ res
ctl_par (data cycle)	fe	broaderr	53	runway master	not valid	not relevant	not relevant
ctl_par (addr cycle)	fe	broaderr	53	runway master	rw slave addr	not relevant	not relevant
addr_par	fe	broaderr	53	runway master	rw slave adr	not relevant	not relevant
data_par	he	errorl	53	gsc+ master	gsc+ slave adr	not relevant	not relevant
rw timeout	he	errorl	59	gsc+ master	gsc+ slave adr	not relevant	not relevant
inc tran siz	he	errorl	50	gsc+ master	gsc+ slave adr	not relevant	not relevant
rw path err	he	errorl	4	gsc+ master	gsc+ slave adr	not relevant	not relevant
TLB fault	he	errorl	61	gsc+ master	gsc+ slave adr	not relevant	not relevant
gsc impr ac	he	errorl	62	gsc+ master	gsc+ slave adr	not relevant	not relevant
gsc+ par	he	errorl	5	not relevant	not relevant	gsc+ master	gsc+ slave adr
gsc+ errorl	he	–	52	not relevant	not relevant	gsc+ master	gsc+ slave adr
gsc+ proto	he	errorl	54	not relevant	not relevant	gsc+ master	gsc+ slave adr
gsc+ wdog	he	errorl	17	not relevant	not relevant	gsc+ master	not valid

On memory read returns, it is possible for both a data parity error and Runway path error to simultaneously occur. In this case, the Runway path error is logged in the IO_STATUS estat, and the data parity error is disregarded.

Note: When the Runway block goes into either hard or fatal error mode, the GSC+ block will fire the errorl signal only relative to the data cycle if the transaction was a pended DMA read and during any cycle in the midst of a connected DMA read. Also, the GSC+ master_id is stored as a 6 whenever the error occurred during an LSL assertion. In this case, the value of the IO_ERR_INFO rqi bit is set to indicate that this field is invalid. Finally, the GSC+ block will not store the GSC+ slave address during a pended DIO read return on an ERRORL assertion.

In some cases, the GSC+ block must queue error indications to the Runway block to indicate the need to fire directed error. Hard or fatal modes are communicated between the two blocks via synchronized wires, and these will affect the IOA's ability to handle subsequent Runway or GSC+ transactions.

Also, Runway must queue error read returns to GSC+. This is to prevent deadlock and to ensure a path to PDC, should a GSC+ guest busy a DIO transaction while waiting for a pended DMA response which never occurs. In fact, should any pended DMA request get errored (as an example due to TLB fault), then a dummy pool entry must be made. It will eventually timeout, causing an errored read return to GSC+.

UTurn does not respond to Runway directed error transactions. The transaction causing the directed error will ultimately timeout and a timeout error is logged.

3. UTurn performance projections

The UTurn chip is a bus converter between a single Runway port and two fully independent 32–bit GSC+ ports. When GSC+ is operated at 33.3 MHz, the UTurn implementation provides for a sustainable aggregate inbound DMA throughput of up to 104 MBytes/sec and a sustainable aggregate outbound DMA throughput of up to 85 MBytes/sec. Individual guests may sustain throughputs of up to 104 MBytes/sec inbound and 53 MBytes/sec outbound. If the GSC bus frequency is increased to 40 MHz, maximum aggregate DMA throughputs increase to 134 and 102 MBytes/sec for inbound and outbound DMA respectively. Maximum single guest throughputs for inbound and outbound DMA transactions are 124 and 60 MBytes/sec respectively at the 40 MHz GSC bus rate. For processor–initiated direct I/O transactions (DIO transactions), the UTurn design is able to sustain an outbound transfer rate of up to 256 MBytes/sec for GSC+ at 40 MHz. The tables that follow list details. Throughput depends on many factors including bus clock frequency, length of transaction, bus loading, and GSC+ protocol implementation by guests.

With ”typical” Runway loading (moderate CPU traffic), the following tables give the aggregate (across all guests) and individual guest throughput numbers that are sustainable through UTurn. Keep in mind that DMA transactions are initiated by GSC+ guests and are directed to/from memory. DIO transactions are initiated by a processor (or perhaps by memory in the case of block moves), and are directed to/from an I/O device. Prefetch is supported only for outbound DMA transactions. UTurn supports pended (split) GSC+ read transactions in both the DMA and the DIO classes, but for DIO reads UTurn allows only a single pended read to be outstanding on GSC+ at any given time.

This chapter begins with summary tables of performance data, followed by a rather detailed description of how the numbers were calculated. The detail calculations result in several intermediate tables of data which can be used to calculate other system–dependent performance figures. The detail calculations were only performed for GSC+ frequencies of 33.3 and 40 MHz in most cases. Corresponding derivation of performance numbers for 32 MHz and/or 50 MHz GSC+ operation is left as an exercise for the reader.

Data transfer performance through UTurn is highly dependent upon the load present on the GSC bus. GSC guest devices are not necessarily efficient users of bus bandwidth. Latency to memory for connected DMA reads can cause a guest to monopolize the GSC bus for a significant percentage of the time. The data presented in this chapter attempts to address a variety of bus load conditions, from no load (best case throughputs) to saturation (worst case throughputs). The data in the first two tables generally reflects light to moderate loading on the system busses. When trying to consider the many ways a GSC bus might be used in a given system, the amount of work to compute an exhaustive set of performance tables appears to have less value than presenting the method for getting one set of answers. Separate performance calculations can then be performed based on configuration of a given system.

TABLE: Aggregate DMA Throughput, GSC+ <=> Memory through UTurn

Transaction class	Data Direction	Description	Transaction size (bytes)	Throughput (MBytes/sec) – Runway / GSC+ frequency =			
				100 / 33.3	120 / 33.3	120 / 40	120 / 50
DMA	Inbound (to mem from I/O)	Writes (four writes per guest per bus tenure)	32	112	112	134	
			16	96	96	116	
		Writes (one write per guest per bus tenure)	32	96	96	116	
			16	76	76	91	
	Outbound (from mem to I/O)	Connected reads No prefetch	32	29	33	35	
			16	16	19	20	
		Pended (split) reads No prefetch	32	77	81	92	
			16	51	54	61	
		Connected reads with prefetch	32	48	50	58	
			16	29	31	35	
		Pended (split) reads with prefetch	32	88	88	100	
			16	60	60	72	

TABLE: Aggregate DIO Throughput, Processor <=> GSC+ through UTurn

Transaction class	Data Direction	Description	Transaction size (bytes)	Throughput (MBytes/sec) – Runway / GSC+ frequency =			
				100 / 33.3	120 / 33.3	120 / 40	120 / 50
DIO	Outbound (from proc to I/O)	Variable length writes, 2X mode	32	213	213	256	320
			24	200	200	240	300
			16	177	177	213	266
			12	133	133	160	200
			8	100	100	120	150
		Variable length writes, 1.5X mode	32	118	118	142	177
			24	114	114	137	171
			16	106	106	128	160
			12	100	100	120	150
		Fixed length writes or variable length writes in 1.5X mode	8 f [1]	88	88	106	133
			8	66	66	80	133
			4 f [1]	66	66	80	100
			4	44	44	53	66
	Inbound (to proc from I/O)	Connected reads	8	11	13	14	16
			4	6	6	7	8
		Pended reads	8	9	10	11	12
			4	5	5	6	7

[1] The ”fast” DIO write transfer mode is supported for multiple writes to the same device or, in the case of 8 byte fast transfers, to multiple devices when the devices signal ready during the first data cycle (no parity error checking prior to signaling ready). The typical guest target for this transfer mode is graphics.

TABLE: Individual Guest DMA Throughput, GSC+ <=> Memory through UTurn

Transaction class	Direction	Description	Transaction size (bytes)	Throughput (MBytes/sec) – Runway / GSC+ frequency =			
				100 / 33.3	120 / 33.3	120 / 40	120 / 50
DMA	Inbound (to mem from I/O)	Writes (four writes per guest per bus tenure)	32	104	104	124	
			16	85	85	102	
		Writes (one write per guest per bus tenure)	32	76	76	91	
			16	53	53	64	
	Outbound (from mem to I/O)	Connected reads No prefetch	32	31	34	37	
			16	17	19	21	
		Pended (split) reads No prefetch	32	29	32	35	
			16	16	18	20	
		Connected reads with prefetch	32	41	42	49	
			16	24	25	29	
		Pended (split) reads with prefetch	32	46	48	55	
			16	28	29	33	

TABLE: Individual Guest DIO Throughput, Processor <=> GSC+ through UTurn

Transaction class	Direction	Description	Transaction size (bytes)	Throughput (MBytes/sec) – Runway / GSC+ frequency =			
				100 / 33.3	120 / 33.3	120 / 40	120 / 50
DIO	Outbound (from proc to I/O) Outbound (from proc to I/O)	Variable length writes, 2X mode	32	213	213	256	320
			24	200	200	240	300
			16	177	177	213	266
			12	133	133	160	200
			8	100	100	120	150
		Variable length writes, 1.5X mode	32	118	118	142	177
			24	114	114	137	171
			16	106	106	128	160
			12	100	100	120	150
		Fixed length writes or variable length writes in 1.5X mode	8 f [1]	88	88	106	133
			8	66	66	80	133
			4 f [1]	66	66	80	100
			4	44	44	53	66
	Inbound (to proc from I/O)	Connected reads	8	11	13	14	16
			4	6	6	7	8
		Pended reads	8	9	10	11	12
			4	5	5	6	7

[1] The ”fast” DIO write transfer mode is supported for multiple writes to the same device or, in the case of 8 byte fast transfers, to multiple devices when the devices signal ready during the first data cycle (no parity error checking prior to signaling ready). The typical guest target for this transfer mode is graphics.

3.1. External Assumptions

GSC+ Frequency
Range: 24 – 50 MHz
Minimum design goals (UTurn exceeds these in current systems):
1 guest at up to 50 MHz
5 guests at up to 40 MHz
6 guests at up to 33.3 MHz
GSC+ Data width: 32 bits
GSC+ Transfer sizes: 1,2,3,4,8,16,32 bytes, plus UTurn–mastered variable–length writes (1–8 words)
GSC+ cycle types:

Address	Master asserts ADDVL, drives address on AD[] bus and transaction type on TYPE[] bus
Turn	Master de–asserts ADDVL, tri–states AD[] bus, keeps bus ownership
Data	Source of data drives data on AD[]; sink of data may handshake with READYL or RETRYL (For pended DMA read return, UTurn asserts READYL in the first data cycle of the return)
Restore	Master de–asserts all control lines and tri–states AD[] bus; bus ownership relinquished
Notify	UTurn drives DRRL and one of BGL[n], informing guest n of imminent DMA read return

DMA and DIO Read protocol: Address, Turn, (1–8)Data, Restore
DMA and DIO Pended Read protocol: Address, Restore, Notify, (1–8)Data, Restore
DMA Write protocol: Address, (1–8)Data, Restore
DIO Fast Write protocol: Address, (1–2)Data, Address, (1–2)Data,..., Address, (1–2)Data, Restore
Arbitration: Fair (round robin) amongst guests;
Read returns and DIO transactions at higher priority level
GSC+ maximum tenure (timeout value): Software programmable
DIO transactions, connected (UTurn–mastered)
Maximum programmable timeout = 262140 GSC+ GCLK cycles (7.86 ms at 33.3 MHz)
DMA transactions (guest mastered)
Maximum timeout per transaction = 1.05e6 GSC+ GCLK cycles (31.5 ms at 33.3 MHz)
Maximum number of mastering GSC+ guests: 6
Maximum latency to GSC+ bus ownership: Very system specific

Memory latency: see Table ”Memory Latencies for Tower–based Systems”

For optimum performance, the only wait states introduced for guest–mastered DMA transactions on GSC+ should be those to accommodate memory latency in the case of reads. The worst case latency for the performance alternatives evaluated will assume that maximum bus tenure per module is the minimum number of bus cycles required for a particular transaction.

Note that in this chapter, 1 megabyte per second (1 MB/s) is 10e6 bytes per second, or 1 byte per microsecond. Multiply by 1.05 to get the number of 2e20 ”megabytes per second”.

3.2. Internal Timing Assumptions

The UTurn design assumes full synchronization between the GSC+ and Runway domains. Runway and GSC+ clocks are fully independent.

All outbound traffic (from Runway to GSC+) will use the following timing:

4.0 Runway cycles to queue
1.5 GSC+ cycles to synchronize
1.5 GSC+ cycles to dequeue

Total: 4.0 Runway cycles
3.0 GSC+ cycles

All inbound traffic (from GSC+ to Runway) will use the following timing:

0.5 GSC+ cycle to queue
3.0 Runway cycles to synchronize
2.0 Runway cycles for queue internal pipeline delay to demux and prep entry
8.0 Runway cycles to dequeue, translate address, and perform various checks
6.5 Runway cycles to arbitrate and win the Runway bus (minimum)

Total: 20.5 Runway cycles
0.5 GSC+ cycles (Rounds up to 1 cycle; partial cycles can't be exploited in the bus protocol)

The timing for a prefetch hit is a special case. The round trip delay for a prefetch hit is:

0.5 GSC+ cycle to queue
3.0 Runway cycles to synchronize
2.0 Runway cycles for internal pipeline delay
6.0 Runway cycles to dequeue and update pool entry
8.0 Runway cycles to determine prefetch hit and indicate valid data return (outbound)
1.5 GSC+ cycles to synchronize
1.5 GSC+ cycle to dequeue and drive data to GSC pads

Total: 19.0 Runway cycles
3.5 GSC+ cycles (Rounds up to 4 cycles; partial cycles can't be exploited in the bus protocol)

The following memory latency numbers, provided by the Tower team in October, 1994, are used in this report. (Tower is the master memory controller currently used in all systems that have Runway as the processor/memory bus.)

TABLE: Memory Latencies for Tower–based Systems

Runway frequency	Average Memory Latency (Minimum)		Average Total Latency (including Runway address and data cycles)	
	Runway cycles	Time (ns)	Runway cycles	Time (ns)
80 MHz	25.5 cycles	256.3 ns	30.5 cycles	318.8 ns
100 MHz	25.5 cycles	205.0 ns	30.5 cycles	255.0 ns
120 MHz	25.5 cycles	170.8 ns	30.5 cycles	212.5 ns

3.3. GSC+–only performance

The following table quantifies the maximum sustainable bandwidth on a GSC+ bus. The minimum number of cycles per transaction are considered at various GSC+ GCLK frequencies to determine the maxi-

mum throughput. In this table (and throughout this document), ”DMA” refers to transactions mastered by GSC guest devices. ”DIO” refers to transactions mastered by UTurn. GSC modules may be designed to master multiple transactions per bus ownership (bus tenure). This can result in a significant improvement in write transaction bandwidth, as shown in the table.

TABLE: GSC+ theoretical maximum sustainable throughput (MBytes/sec)

Transaction type	Transaction size		GSC+ cycles (avg)	Maximum sustainable throughput (MBytes/sec); GSC+ GCLK frequency =				
	Bytes	Words		24 MHz	32 MHz	33.3 MHz	40 MHz	50 MHz
DMA or DIO write (DMA to memory, DIO to I/O) – 1 write per bus tenure	1–4 [1]	<= 1 [1]	3	<= 32	<= 43	<= 44	<= 53	<= 66
	8 [1]	2 [1]	4	48	64	66	80	100
	16	4	6	64	85	88	106	133
	32	8	10	76	102	106	128	160
DMA or DIO write (DMA to memory, DIO to I/O) – 4 writes per bus tenure	1–4 [1]	<= 1 [1]	2.25	<= 42	<= 56	<= 59	<= 71	<= 88
	8 [1]	2 [1]	3.25	59	78	82	98	123
	16	4	5.25	73	97	101	121	152
	32	8	9.25	83	110	115	138	172
DMA or DIO read (DMA from memory, DIO from I/O) – connected reads	1–4 [1]	<= 1 [1]	4	<= 24	<= 32	<= 33	<= 40	<= 50
	8 [1]	2 [1]	5	38	51	53	64	80
	16	4	7	54	73	76	91	114
	32	8	11	69	93	96	116	145

[1] Not considered viable candidates for bulk DMA transfers

In the sections that follow, the effects of bus loading, bus and memory latency, arbitration, and protocol implementation will be considered.

3.4. DMA Write Performance Through UTurn’s IOAs

3.4.1. Guest–initiated (DMA) writes

UTurn buffers writes from GSC+ guests. The inbound queueing structure is deep enough and Runway generally fast enough that for most expected system configurations UTurn will be able to accept 8 word writes from GSC+ guests and forward them to memory or processors at the peak rate that they can be issued on GSC+. For 4 word writes, throughput depends on the type of page being updated in memory. Pages are classified as either ”safe” or ”fast” by system software that configures UTurn’s Runway–side translation lookaside buffer (TLB). (The TLB provides address mapping information required for translating 32–bit GSC addresses to 40–bit Runway addresses.) If a 4 word write is directed to a ”fast” page, UTurn can buffer and retire 4–word writes at maximum GSC transfer rates. For 4–word writes to ”safe” pages, and for 1 or 2 word writes to any page type, UTurn must perform an atomic cache line update sequence on Runway. This read–modify–write sequence requires approximately 60 Runway cycles per write transaction. In these cases, write throughput is limited by Runway. Guest modules are strongly urged not to use these smaller write transactions for bulk data transfer. System performance impacts can be substantial.

The following tables summarize DMA write performance expectation for various transaction sizes and bus frequency combinations. First, aggregate DMA write throughput is calculated. A guest is assumed to perform one write transaction per bus tenure (that is, each guest relinquishes bus ownership after performing a single write transaction). Guests can be designed to perform multiple writes per bus tenure, improving single–guest and aggregate DMA write bandwidth (fewer dead cycles on the bus). This generally affects the latency to bus access by other devices, however. Guests should always be designed such that they can be configured to perform only one DMA write transaction per bus tenure. Due to UTurn's implementation, average write transaction length is 1 cycle longer than the minimum write transaction length allowed by the GSC protocol. UTurn adds an extra cycle whenever bus ownership changes from one guest to another.

TABLE: GSC+ DMA writes – aggregate sustainable DMA write throughput
(one DMA write per guest bus tenure)

Runway load	Transfer size (bytes)	Memory Page type	Average cycle count	Runway / GSC+ frequency (in MHz) =			
				80 / 33.3	100 / 33.3	120 / 33.3	120 / 40
				Thruput (MB/sec)	**Thruput (MB/sec)**	**Thruput (MB/sec)**	**Thruput (MB/sec)**
Best/ Typ/ Worst	32	either	11G	**96**	**96**	**96**	**116**
	16	fast	7G	**76**	**76**	**76**	**91**
Best	16	safe	62R + 2G	**19**	**23**	**27**	**28**
Typ	16	safe	70R + 2G	**17**	**21**	**24**	**25**

If guests are designed to allow multiple write transactions per bus tenure, aggregate DMA write throughput can be improved as shown in the following table. Writes within a given bus tenure are performed back–to–back with no intervening dead cycles. The table shows the total number of bytes transferred and the total cycles for a given number of write transactions per bus tenure. Only GSC–limited writes are considered, since no GSC speedups will improve performance of 16–byte writes to safe pages.

TABLE: GSC+ DMA writes – aggregate sustainable DMA write throughput
(**One, two, and four** DMA writes per guest bus tenure; GSC–limited performance cases only)

Transfer size (bytes)	Memory page type	Transactions per bus tenure	Total bytes transferred per tenure	Total GSC cycle count	Runway / GSC+ frequency (in MHz) =			
					80 / 33.3	100 / 33.3	120 / 33.3	120 / 40
					Thruput (MB/sec)	**Thruput (MB/sec)**	**Thruput (MB/sec)**	**Thruput (MB/sec)**
32	fast/safe	1	32	11	**96**	**96**	**96**	**116**
16	fast	1	16	7	**76**	**76**	**76**	**91**
32	fast/safe	2	64	20	**106**	**106**	**106**	**128**
16	fast	2	32	12	**88**	**88**	**88**	**106**
32	fast/safe	4	128	38	**112**	**112**	**112**	**134**
16	fast	4	64	22	**96**	**96**	**96**	**116**

The single–guest sustainable DMA write throughput is less than the aggregate rate due to GSC arbitration requirements and associated overheads. The GSC specification requires that after relinquishing bus ownership, a guest must see its bus grant signal released before it can re–assert a bus request. Between this protocol requirement and UTurn's implementation, an additional 4 GSC cycles of arbitration overhead are added each time a guest relinquishes and then attempts to regain bus ownership. For optimum single–guest DMA write performance, guests should be designed such that they can optionally perform multiple write transactions per bus tenure. This capability will hurt the performance of other guests on the same bus segment, so caution should be exercised when enabling the capability. Guests should always have the ability to be configured to perform only 1 transaction per bus tenure.

Taking the extra arbitration cycles into account, a single guest DMA write throughput table can be constructed. In the table, cycle counts are specified in either GSC cycles (G) or Runway cycles (R), depending on where performance is ultimately limited. The table assumes that a guest de–asserts its bus request during the last data cycle of a write, as allowed by the GSC specification. If bus request is de–asserted later, throughput will be reduced in the GSC–limited cases.

TABLE: GSC+ DMA writes – single guest sustainable DMA write throughput
(one DMA write per guest bus tenure)

Runway load	Transfer size (bytes)	Memory Page type	Average cycle count	Runway / GSC+ frequency (in MHz) =			
				80 / 33.3	100 / 33.3	120 / 33.3	120 / 40
				Thruput (MB/sec)	**Thruput (MB/sec)**	**Thruput (MB/sec)**	**Thruput (MB/sec)**
Best/ Typ/ Worst	32	either	14G	**76**	**76**	**76**	**91**
	16	fast	10G	**53**	**53**	**53**	**64**
Best	16	safe	62R + 2G	**19**	**23**	**27**	**28**
Typ	16	safe	70R + 2G	**17**	**21**	**24**	**25**

If a guest is designed to allow 4 write transactions per bus tenure, single guest DMA write throughput can be improved as shown in the following table. Writes within a given bus tenure are performed back–to–back with no intervening dead cycles. The table shows the total number of bytes transferred and the total cycles for four write transactions. Only GSC–limited writes are considered, since no GSC speedups will improve performance of 16–byte writes to safe pages.

TABLE: GSC+ DMA writes – single guest sustainable DMA write throughput
(**One, two, or four** DMA writes per guest bus tenure; GSC–limited performance cases only)

Transfer size (bytes)	Memory page type	Transactions per bus tenure	Total bytes transferred per tenure	Total GSC cycle count	Runway / GSC+ frequency (in MHz) =			
					80 / 33.3	100 / 33.3	120 / 33.3	120 / 40
					Thruput (MB/sec)	**Thruput (MB/sec)**	**Thruput (MB/sec)**	**Thruput (MB/sec)**
32	fast/safe	1	32	14	**76**	**76**	**76**	**91**
16	fast	1	16	10	**53**	**53**	**53**	**64**
32	fast/safe	2	64	23	**92**	**92**	**92**	**111**
16	fast	2	32	15	**71**	**71**	**71**	**85**
32	fast/safe	4	128	41	**104**	**104**	**104**	**124**
16	fast	4	64	25	**85**	**85**	**85**	**102**

3.5. DMA Read Performance Through UTurn's IOAs

3.5.1. Connected DMA reads; no prefetch

A connected DMA read is the simplest form of read transaction that a GSC+ guest can initiate. The transaction is connected on GSC+ in that the guest issues an address and then, without relinquishing ownership of the bus, waits for the data to be returned by the GSC+ host. In Kitty Hawk, the host (UTurn) must issue a read transaction across the Runway bus to memory and receive the read data back from memory before returning this data to the waiting guest. As will be seen, the latency from the GSC+ read address cycle to the first cycle of data is substantial in a Kitty Hawk system. For this section, prefetching by the host is not considered (it will be in a later section).

Calculation of individual–guest and bus aggregate throughput for guests performing connected read transactions requires quantification of the latency to data across UTurn and Runway. UTurn internal delays, Runway arbitration delays, and memory access times must all be accounted for.

As noted earlier in this chapter, UTurn's internal design will impose the following delays between the read address cycle on GSC+ and issuance a corresponding Runway read transaction:

0.5 GSC+ cycle to queue
3.0 Runway cycles to synchronize
4.0 Runway cycles for queue internal pipeline delay to demux and prep entry
8.0 Runway cycles to dequeue, translate address, and perform various checks
6.0 Runway cycles to arbitrate and win the Runway bus (minimum)

Total: 21.0 Runway cycles
0.5 GSC+ cycles

In the best I/O performance case, there are no additional Runway cycles required on a read request due to a busy bus. The most common Runway transactions – cache line read requests, read returns, and writes of various flavors – average 3 cycles in length. If a typical UTurn arbitration scenario forces UTurn to wait for one Runway transaction, 3 Runway clocks will be added to the arbitration time. For a worst case, assume that a processor is in the middle of a write transaction and a memory controller read return is also queued, resulting in an additional 7 Runway cycle arb delay.

Memory latencies were listed in an earlier table, and are reproduced here. The values in the table do not reflect any of the Runway arbitration delay computed in the preceding paragraph.

TABLE: Memory Latencies for Tower–based Systems

Runway frequency	Average Memory Latency (Minimum)		Average Total Latency (including Runway address and data cycles)	
	Runway cycles	Time (ns)	Runway cycles	Time (ns)
80 MHz	25.5 cycles	256.3 ns	30.5 cycles	318.8 ns
100 MHz	25.5 cycles	205.0 ns	30.5 cycles	255.0 ns
120 MHz	25.5 cycles	170.8 ns	30.5 cycles	212.5 ns

For read returns, mastered by Tower on Runway, the minimum arbitration latency is 0 cycles, since Tower contains the Runway arbiter and can ensure no delay in many cases. For the typical case, Tower must wait for half of a processor or UTurn initiated transaction – 2 cycles roughly. For the worst case, Tower waits for an entire processor or UTurn cache line write – 5 cycles. There are certainly cases which exceed this, but these are the expected cases.

The UTurn read request transaction arbitration delay and Tower read return arbitration delays are summarized in the following table.

TABLE: Runway arbitration delays for reads from memory

Runway case	Additional UTurn arbitration delay	Runway Tower arbitration delay (for read return)	Total Runway arbitration delays
Best	0 Runway cycles	0 Runway cycles	0 Runway cycles
Typical	3 Runway cycles	2 Runway cycles	5 Runway cycles
Worst	7 Runway cycles	5 Runway cycles	12 Runway cycles

The UTurn internal delay cycles incurred between the end of the Runway read return transaction to the first data cycle on GSC+ are as follows:

4.0 Runway cycles to queue
1.5 GSC+ cycles to synchronize
1.5 GSC+ cycles to dequeue

Total: 4.0 Runway cycles
3.0 GSC+ cycles

Summing the UTurn internal delays for each read (combining delays for the request and the data return portions), the total UTurn internal delay is 25 Runway cycles and 3.5 GSC+ cycles. Only integer GSC cycle counts are sensible in this analysis, so the total UTurn internal delay is effectively 25 Runway cycles and 4 GSC cycles. When the Runway arbitration delays and memory latency values are added in, a total delay from the address cycle of the read transaction on GSC+ to first data on GSC+ can be calculated.

TABLE: Latency from address cycle on GSC+ to first data cycle on GSC+ (DMA reads) – equation

Case	Delay	Delay components
best	0R+25.5R+5R+25R+4G	R_arb_best + mem_access + R_xfer + int_delay
typical	5R+25.5R+5R+25R+4G	R_arb_typical + mem_access + R_xfer + int_delay
worst	12R+25.5R+5R+25R+4G	R_arb_worst + mem_access + R_xfer + int_delay

TABLE: Runway/memory latency to first GSC+ data cycle for connected reads – time and cycles

Case	Runway / GSC+ frequency combination							
	80 MHz / 33.3 MHz		100 MHz / 33.3 MHz		120 MHz / 33.3 MHz		120 MHz / 40 MHz	
	Time (ns)	**GSC+ cycles**	Time (ns)	**GSC+ cycles**	Time (ns)	**GSC+ cycles**	Time (ns)	**GSC+ cycles**
Best	814	**28**	675	**23**	583	**20**	563	**23**
Typical	877	**30**	725	**25**	625	**21**	605	**25**
Worst	964	**33**	795	**27**	683	**23**	663	**27**

Now, combining this latency data with GSC+ transaction length data, a per device (and aggregate) throughput value can be calculated.

TABLE: Cycle count and aggregate read throughput from a GSC+ bus through UTurn for connected DMA reads

Case	Xfer size	cycles in basic transaction	Runway/GSC+ frequency combination											
			80 MHz / 33.3 MHz			100 MHz / 33.3 MHz			120 MHz / 33.3 MHz			120 MHz / 40 MHz		
			UTurn / Rway delay	Total cycles	**Thru-put (MB/s)**	UTurn / Rway delay	Total cycles	**Thru-put (MB/s)**	UTurn / Rway delay	Total cycles	**Thru-put (MB/s)**	UTurn / Rway delay	Total cycles	**Thru-put (MB/s)**
Best	16	7	28	35	**15**	23	30	**17**	20	27	**17**	23	30	**21**
	32	11	28	39	**27**	23	34	**31**	20	31	**34**	23	34	**37**
Typ	16	7	30	37	**14**	25	32	**16**	21	28	**19**	25	32	**20**
	32	11	30	41	**26**	25	36	**29**	21	32	**33**	25	36	**35**
Worst	16	7	33	40	**13**	27	34	**15**	23	30	**17**	27	34	**18**
	32	11	33	44	**24**	27	38	**28**	23	35	**30**	27	38	**33**

In summary, aggregate read throughput using connected DMA reads is primarily affected by the size of transaction chosen – 32 byte transfers result in a 70–80% read throughput gain over 16 byte transfers.

To calculate sustainable performance for an individual guest with a variety of other guests active on the bus, GSC+ arbitration latency needs to be calculated. Best, typical and worst cases will be evaluated. For each case, transaction mixes of all connected reads, all writes/pended reads, and 50% reads / 50% writes are considered. 16 byte and 32 byte transfer size effects are also considered. The performance impact on DMA read throughput due to GSC host–mastered (DIO) transaction traffic is expected to be small (< 5%) in transaction scenarios involving disk and network accesses. DIO transaction traffic is not considered in this DMA read performance analysis.

Best case for an individual guest involves consecutive transactions from the same guest. Starting at bus idle, the first GSC+ transaction of a sequence always incurs at least 3 GSC+ cycles of overhead for arbitration (request and grant cycles; UTurn takes 2 cycles minimum to return a grant after seeing a request). Subsequent transactions from the same guest, however, may be performed without arbitration overhead.

For the typical case, a requesting guest waits for completion of a transaction in progress, then is granted the bus. The estimated latency required to gain ownership in this case to be 1/2 the number of GSC+ clocks that an average transaction is on the bus (including any GSC+–visible Runway or memory latency). The ”total cycles” entry of the ”typ” rows from the preceding table is used as the number of GSC+ cycles per bus tenure for connected reads, and the ”total cycles” column of the max sustained throughput table is used as the transaction cycle count for writes and pended reads (discussed later).

For the individual guest worst case, GSC+ is ”very busy”, and a guest must wait on average the length of two transactions by other masters before it becomes master. The ”total cycles” entry of the ”worst” rows from the preceding table is used as the number of GSC+ cycles per bus tenure for reads, and the ”total cycles” column of the max sustained throughput table per bus tenure for writes and pended reads.

TABLE: GSC+ guest arbitration delay

Case	Transfer size	Arbitration delay (GSC+ cycles); Runway/GSC+ frequency =								
		100 MHz / 33.3 MHz			120 MHz / 33.3 MHz			120 MHz / 40 MHz		
		Connected reads	Writes / Pended reads	**50/50**	Connected reads	Writes / Pended reads	**50/50**	Connected reads	Writes / Pended reads	**50/50**
Best	16	0	0	**0**	0	0	**0**	0	0	**0**
	32	0	0	**0**	0	0	**0**	0	0	**0**
Typical	16	16	3	**10**	14	3	**9**	16	3	**10**
	32	18	5	**12**	16	5	**11**	18	5	**12**
Worst	16	68	12	**40**	60	12	**36**	68	12	**40**
	32	76	20	**48**	70	20	**45**	76	20	**48**

Now, by combining this GSC+ arbitration delay with the transaction lengths calculated earlier, an expected individual guest throughput figure can be calculated.

TABLE: Single device connected DMA read transaction lengths and resulting throughputs
(traffic initiated by other GSC+ guests assumed to be 50% connected DMA reads, 50% DMA writes)

Case	Transfer size	Cycles in basic transaction	Runway / GSC+ frequency combination											
			100 MHz / 33.3 MHz				120 MHz / 33.3 MHz				120 MHz / 40 MHz			
			GSC arb delay cycles	UTurn/ Rway delay	Total GSC cycles	Thruput, MB/s	GSC arb delay cycles	UTurn/ Rway delay	Total GSC cycles	Thruput, MB/s	GSC arb delay cycles	UTurn/ Rway delay	Total GSC cycles	Thruput, MB/s
Best	16	7	0	23	30	**17**	0	20	27	**19**	0	23	30	**21**
	32	11	0	23	34	**31**	0	20	31	**34**	0	23	34	**37**
Typical	16	7	10	25	42	**12**	9	21	37	**14**	10	25	42	**15**
	32	11	12	25	48	**22**	11	21	43	**24**	12	25	48	**26**
Worst	16	7	40	27	74	**7**	36	23	66	**8**	40	27	74	**8**
	32	11	48	27	86	**12**	45	23	79	**13**	48	27	86	**14**

3.5.2. Pended DMA reads; no prefetch

A GSC+ guest may elect to ”pend” a read transaction if the host device supports this capability, dividing the read into two separate GSC+ transactions – a request phase and a response phase. The request phase is a single GSC+ cycle of address and transaction size information accompanied by assertion of a ”pend acknowledge” control line. The address cycle is followed by a restore cycle as the guest relinquishes ownership allowing other bus traffic to proceed. Some time later, the GSC+ host initiates a read return

transaction to supply the read data requested by the guest. A pended read (request and response combined) consumes one more GSC cycle than a minimum length connected read (assuming the same memory latency) due to the notification cycle required by the protocol at the beginning of the read return. The GSC+ definition supports only a single outstanding pended read transaction per guest at any given time. Thus, for 4 guests, there may be up to 4 outstanding pended DMA reads at any given time, but only one per guest. This limitation stems from the fact that there is no transaction ID or transaction sequence number defined on GSC+, ordering from memory through UTurn is not guaranteed for DMA read data returns (even to the same guest), and the read return protocol does not require the host to return an address that can be matched by the guest.

Not all guests on a given bus segment will necessarily implement pended DMA reads. With this in mind, individual guest DMA read performance tables will be generated for two different bus configurations. In one case, all DMA reads are pended, maximizing aggregate throughput. In another (probably more typical) case, there will be a mix of guests performing pended and connected reads.

UTurn internal latencies and Runway/memory latencies for pended reads will be identical to those calculated in the connected read case. The only individual guest performance difference between connected and pended reads is the fact that the host (UTurn) must arbitrate for GSC+ before performing the read return transaction to supply the data in the pended read case. The GSC+ arbitration latency for the read return phase of the transaction must be calculated. GSC+ arbitration, as implemented in UTurn, prioritizes pended DMA read returns over all other bus traffic. Best, typical, and worst cases for GSC+ read return arbitration delay are considered. First, consider a bus configuration where other GSC guests are performing an equal mix of DMA writes, connected DMA reads, and pended DMA reads.

For the best case, no other guests are arbitrating for GSC+ and no other entries blocking UTurn's read return queue when the read data is available for return to the requester. The arbitration overhead is 1 GSC+ cycle for the host to indicate "data coming" to the guest.

For the worst case, GSC+ is busy, and the host must wait on average half the number of cycles in an average–length GSC+ transaction before it can master a read return. Transactions ahead of a particular read return in UTurn's read return queue may add to this latency. To get an average transaction length, assume connected read, read request, read response, and write transactions to be equally likely to impede the read return of interest. The average length value used for a connected read comes from the aggregate throughput table for connected DMA reads calculated earlier; the cycle counts reflect the 120MHz Runway/33.3 MHz GSC frequency combination.

The typical case falls somewhere between the best and worst cases. For simplicity, assume the typical latency to be half way between the best and worst cases.

TABLE: GSC+ host (UTurn) arbitration delay for issuance of read returns
(Other guests performing DMA writes, connected DMA reads, and pended DMA reads)

Case	Transaction size	GSC+ transaction length (GSC+ cycles)					**UTurn GSC+ arbitration latency for read returns**
		Connected read	Pended read request	Pended read return	Write	Average	
Best	16	27	2	6	7	11	**1**
	32	31	2	10	11	14	**1**
Typical	16	28	2	6	7	11	**3**
	32	32	2	10	11	14	**4**
Worst	16	30	2	6	7	12	**6**
	32	34	2	10	11	15	**8**

Summing the arbitration delay from the preceding table and the internal and external delay components from the connected read analysis, a total latency can be calculated for an individual read.

TABLE: Latency from address cycle on GSC+ to first return data cycle – pended DMA read case

Case	Size	Delay (R=Runway;G=GSC+)	Delay components
best	16/32	0R+25.5R+5R+25R+4G+1G	R_arb_b+mem_access+R_xfer+int_delay+G_arb_b
typ	16	5R+25.5R+5R+25R+4G+3G	R_arb_t+mem_access+R_xfer+int_delay+G_arb_t
	32	5R+25.5R+5R+25R+4G+4G	
worst	16	12R+25.5R+5R+25R+4G+6G	R_arb_w+mem_access+R_xfer+int_delay+G_arb_w
	32	12R+25.5R+5R+25R+4G+8G	

TABLE: Runway/memory/GSC+ arbitration latency numbers – pended DMA reads without prefetch

Case	Transaction size	Pended DMA read return delay; Runway / GSC+ frequency =							
		80 MHz / 33.3 MHz		100 MHz / 33 MHz		120 MHz/33.3 MHz		120 MHz / 40 MHz	
		Time (ns)	**GSC+ cycles**	Time (ns)	**GSC+ cycles**	Time (ns)	**GSC+ cycles**	Time (ns)	**GSC+ cycles**
Best	16 / 32	844	**29**	705	**24**	613	**21**	588	**24**
Typical	16	967	**33**	815	**28**	715	**24**	680	**28**
	32	997	**34**	845	**29**	745	**25**	705	**29**
Worst	16	1144	**39**	975	**33**	863	**29**	813	**33**
	32	1204	**41**	1035	**35**	923	**31**	863	**35**

For an individual guest that is limited to a single outstanding pended DMA read transaction on GSC+ at any given time with no prefetch support, the maximum throughput will be limited by memory latency through UTurn and across Runway. (The combination of prefetch and pended read transactions to address memory latency reduction is considered in a later section.) Since UTurn internal delays for the

pended read are the same as for the connected read, and since for the pended read UTurn must re–arbitrate for GSC+ before returning data, the throughput of a GSC+ guest supporting only a single outstanding pended read request will be slightly lower than a guest supporting only connected reads. GSC+ is available for other activity while the pended read is in progress, but the single device performance will be limited. By combining the pended DMA read return delay data in the preceding table with the base transaction length and GSC guest arbitration delay from the connected read case considered earlier, we can determine the DMA read performance expected for a given guest. The GSC guest arbitration delay table is copied below.

TABLE: GSC+ guest arbitration delay

Case	Transfer size	Arbitration delay (GSC+ cycles); Runway/GSC+ frequency =								
		100 MHz / 33.3 MHz			120 MHz / 33.3 MHz			120 MHz / 40 MHz		
		Connected reads	Writes / Pended reads	**50/50**	Connected reads	Writes / Pended reads	**50/50**	Connected reads	Writes / Pended reads	**50/50**
Best	16	0	0	**0**	0	0	**0**	0	0	**0**
	32	0	0	**0**	0	0	**0**	0	0	**0**
Typical	16	16	3	**10**	14	3	**9**	16	3	**10**
	32	18	5	**12**	16	5	**11**	18	5	**12**
Worst	16	68	12	**40**	60	12	**36**	68	12	**40**
	32	76	20	**48**	70	20	**45**	76	20	**48**

Outbound data transfer performance for any one guest issuing pended DMA read transactions (without prefetch) is now calculated, based on the data in the preceeding tables.

TABLE: Individual GSC+ guest pended DMA read performance through UTurn
(single pended transaction per guest, connected reads allowed by other guests)

Case	Xfer size	cycles in basic transaction	Runway/GSC+ frequency combination								
			100 MHz / 33.3 MHz			120 MHz / 33.3 MHz			120 MHz / 40 MHz		
			Arb/ Rway delay	Total cycles	**Thruput (MB/s)**	Arb/ Rway delay	Total cycles	**Thruput (MB/s)**	Arb/ Rway delay	Total cycles	**Thruput (MB/s)**
Best	16	8	24	32	**16**	21	29	**18**	24	32	**20**
	32	12	24	36	**29**	21	33	**32**	24	36	**35**
Typ	16	8	38	46	**11**	33	41	**13**	38	46	**13**
	32	12	41	53	**20**	36	48	**22**	41	53	**24**
Worst	16	8	73	81	**6**	65	73	**7**	73	81	**7**
	32	12	83	95	**11**	76	88	**12**	83	95	**13**

Connected reads issued by other guests have by far the greatest impact on an individual guest's pended DMA read throughput. The impact is certainly seen at moderate to heavy loading conditions, where sev-

eral guests might be issuing connected DMA reads while one guest is attempting to only perform pended DMA reads. It is useful to see the impact of elimination of connected DMA reads on the expected single–guest pended DMA read throughput. Such a table is useful in cases where the total number of connected DMA reads is very small, due perhaps to low bandwidth requirements for guests with less GSC functional capability.

The following tables are a repeat of the above pended DMA read performance analysis, but connected reads from other guests are not considered.

TABLE: GSC+ guest arbitration delay – connected reads from other guests disallowed

Case	Transfer size	Arbitration delay (GSC+ cycles); Runway/GSC+ frequency =								
		100 MHz / 33.3 MHz			120 MHz / 33.3 MHz			120 MHz / 40 MHz		
		Writes	Pended reads	**50/50**	Writes	Pended reads	**50/50**	Writes	Pended reads	**50/50**
Best (no delay)	16	0	0	**0**	0	0	**0**	0	0	**0**
	32	0	0	**0**	0	0	**0**	0	0	**0**
Typical (0.5X avg trans length)	16	3	4	**4**	3	4	**4**	3	4	**4**
	32	5	6	**6**	5	6	**6**	5	6	**6**
Worst (2X avg trans length)	16	12	14	**13**	12	14	**13**	12	14	**13**
	32	20	22	**21**	20	22	**21**	20	22	**21**

TABLE: GSC+ host (UTurn) arbitration delay for issuance of read returns – connected reads disallowed

Case	Transaction size	GSC+ transaction length (GSC+ cycles)				**UTurn GSC+ arbitration latency for read returns**
		Pended read request	Pended read return	Write	Average	
Best (1 cycle arb delay)	16	2	6	7	5	**1**
	32	2	10	11	8	**1**
Typical (0.5 * (best+worst))	16	2	6	7	5	**2**
	32	2	10	11	8	**3**
Worst (0.5X avg trans length)	16	2	6	7	5	**3**
	32	2	10	11	8	**4**

Summing the arbitration delay from the preceding table and the internal and external delay components from the connected read analysis, a total latency can be calculated for an individual read.

TABLE: Latency from address cycle on GSC+ to first return data cycle – pended DMA read case

Case	Size	Delay (R=Runway;G=GSC+)	Delay components
best	16/32	0R+25.5R+5R+25R+4G+1G	R_arb_b+mem_access+R_xfer+int_delay+G_arb_b
typ	16	5R+25.5R+5R+25R+4G+2G	R_arb_t+mem_access+R_xfer+int_delay+G_arb_t
	32	5R+25.5R+5R+25R+4G+3G	
worst	16	12R+25.5R+5R+25R+4G+3G	R_arb_w+mem_access+R_xfer+int_delay+G_arb_w
	32	12R+25.5R+5R+25R+4G+4G	

TABLE: Runway/memory/GSC+ arbitration latency numbers

Case	Trans-action size	Pended DMA read return delay; Runway / GSC+ frequency =							
		80 MHz / 33.3 MHz		100 MHz / 33 MHz		120 MHz/33.3 MHz		120 MHz / 40 MHz	
		Time (ns)	**GSC+ cycles**	Time (ns)	**GSC+ cycles**	Time (ns)	**GSC+ cycles**	Time (ns)	**GSC+ cycles**
Best	16 / 32	844	**29**	705	**24**	613	**21**	588	**24**
Typical	16	937	**32**	785	**27**	685	**23**	655	**27**
	32	967	**33**	815	**28**	715	**24**	680	**28**
Worst	16	1054	**36**	885	**30**	773	**26**	738	**30**
	32	1084	**37**	915	**31**	803	**27**	763	**31**

Outbound data transfer performance for any one guest issuing pended DMA read transactions without prefetch, assuming that connected DMA reads from other guests are disallowed, is given in the following table. The ”Typ” entries from this table are the data values used for the pended DMA read throughput listed in the individual guest thoughput summary table at the beginning of this chapter.

TABLE: Individual GSC+ guest pended DMA read performance through UTurn
(single pended transaction per guest, connected reads by other guests disallowed)

Case	Xfer size	cycles in basic trans-action	Runway/GSC+ frequency combination											
			80 MHz / 33.3 MHz			100 MHz / 33.3 MHz			120 MHz / 33.3 MHz			120 MHz / 40 MHz		
			Arb/ Rway delay	Total cycles	**Thru-put (MB/s)**	Arb/ Rway delay	Total cycles	**Thru-put (MB/s)**	Arb/ Rway delay	Total cycles	**Thru-put (MB/s)**	Arb/ Rway delay	Total cycles	**Thru-put (MB/s)**
Best	16	8	29	37	**14**	24	32	**16**	21	29	**18**	24	32	**20**
	32	12	29	41	**26**	24	36	**29**	21	33	**32**	24	36	**35**
Typ	16	8	32	40	**13**	27	35	**15**	23	31	**17**	27	35	**18**
	32	12	33	45	**23**	28	40	**26**	24	36	**29**	28	40	**32**
Worst	16	8	36	44	**12**	30	38	**14**	26	34	**15**	30	38	**16**
	32	12	37	49	**21**	31	43	**24**	27	39	**27**	31	43	**29**

Computation of aggregate throughput between a GSC+ bus and memory requires characterization of the amount of parallelism provided by both the GSC+ bus specification and by the resources available inside UTurn. The UTurn design goal is to allow the maximum sustainable aggregate bandwidth for all combinations of inbound and outbound DMA transfers (guest–initiated) by adequately sizing internal data structures. Memory latencies for reads are hidden by pipelining read transactions. Aggregate throughput is close to the maximum sustainable throughput table presented earlier, except that for each bus mastership change there is an additional GSC cycle of overhead due to arbitration. Thus, the aggregate throughput must be calculated assuming that read transactions take 2 cycles longer than the minimum (adding 1 arb cycle for the request phase and 1 for the response phase). Writes also take 1 cycle longer than minimum due to arbitration overhead. The following table assumes that a guest performs only one transaction per bus tenure. Guests are allowed to perform multiple write and connected read transactions per bus tenure, and can eliminate arbitration overhead cycles (increasing aggregate throughput) by doing so. The down side is that longer average arbitration latencies may result on a per–guest basis.

TABLE: Aggregate GSC+ Guest–Initiated (DMA) Data Transfer Performance Through UTurn
(read transactions are pended; no prefetch)

Transfer type	Transfer size	GSC+ cycles	Maximum sustained bandwidth (MBytes/sec); GSC+ frequency =			
			24 MHz	32 MHz	33.3 MHz	40 MHz
Write	16	7	54	73	76	91
	32	11	69	93	96	116
Read	16	9	42	56	59	71
	32	13	59	78	82	98

The number of entries in UTurn's inbound and read return queues must be chosen to balance the amount of simultaneous activity allowed to proceed (benefiting performance) while keeping implementation realities in mind. Sustaining maximum aggregate bandwidth in the face of all possible combinations of devices, bus loadings, and transaction sequences is not an easy task, but focus on several key combinations

– the 90% cases – will ensure this performance levels for commonly used transactions. Now, consider the extent of the resources inside UTurn required to sustain high throughput, specifically the depth of the inbound queue and the read return queue in the context of the DMA calculations. The outbound queue will be addressed along with DIO concerns.

To estimate the size of the inbound queue, we should first look at the maximum amount of time that the head of the inbound queue might be obstructed, allowing GSC+–initiated traffic to accumulate behind it. There are two somewhat special transaction cases that can occur in normal operation (although with relative infrequency). These are TLB miss handling sequences and read–modify–write (RMW) sequences. The RMWs are necessary to accommodate sub–half–cache–line writes to memory. Both of these transactions cause the head of the inbound queue to be plugged while a memory read and a memory write are completed. To get a size estimate for the inbound queue, we first need to calculate the total number of cycles that elapse from the time a GSC+ transaction that results in a TLB miss or a RMW until the inbound queue location that this particular GSC+ transaction was stored in is available for another GSC+ transaction. The cycle count includes the following components:

Delay component	GSC+ cycles	Runway cycles
UTurn internal delay for inbound transactions	.5	21
UTurn Runway arbitration delay (typical)		3
Runway read request		1
Memory latency		25.5
Memory Runway arbitration delay (typical)		2
Read return transfer time		4
(Store/update line and return to queue head) or (update TLB entry)		6
UTurn Runway arbitration delay (typical)		3
Runway write line transaction		5
Return inbound queue entry to GSC+ domain	1.5	
Totals	**2**	**70.5**

To determine the number of inbound queue locations that are required to keep GSC+ busy at peak GSC+ data transfer rates, the Runway cycles need to be translated into GSC+ cycles. UTurn's design restricts the range of Runway (ckrw) to GSC (GCLK) frequency ratios between 2:1 and 4:1. (Runway at, say, 100 MHz and GSC+ GCLK at 25 to 50 MHz). As an absolute worst case, assume the 2:1 ratio (80 MHz Runway and 40 MHz GSC+ GCLK, for example). With this ratio, the 70.5 Runway cycles shown in the preceding table translate to 36 GSC+ clocks, and the total inbound queue accumulation time is 38 GSC clocks. Now, consider the following reasonable transaction sequence. A sub–half–line write to memory is followed by DMA pended read requests from 3 GSC+ modules and an even mix of 16 and 32 byte DMA writes from the remaining GSC+ module(s). Within the 38 GSC+ clock period of time required to account for the RMW, read requests from each of the 3 reading guests (9 cycles total), two 16–byte writes (6 cycles each, 12 cycles total), and one 32–byte write (10 cycles) can actually complete on GSC+, with 7 cycles "left over" to start a second 32 byte read. An inbound queue depth of 8 is required to allow these 7 transactions to proceed at GSC+ peak rate while the eighth queue location is tied up with the RMW.

When considering the size of the read request ”pool” area required for outstanding reads to memory on Runway, it is important to keep in mind that only one outstanding read request per guest is allowed with today’s GSC+ definition and UTurn implementation. Since the maximum number of GSC+ guest DMA masters that UTurn accommodates on a GSC+ bus is 6, there can be no more than 6 simultaneously outstanding read requests to memory on Runway, and thus the pool need not be any larger than 6 entries.

For support of prefetch, there is a prefetch ”pool” area very similar to the read request pool. Prefetch is supported with only a single prefetch buffer per GSC+ device (that is, per bus request/bus grant signal pair). Since only 6 guests are supported, only 6 prefetch pool entries are required.

3.5.3. Connected DMA reads with prefetch

The DMA prefetching capability defined for GSC+ and implemented in UTurn attempts to exploit the observation that most DMA traffic initiated by a GSC+ guest will be relatively large blocks of data that are contiguous in a physical address sense, and are thus easily described by an initial address and a byte count. For data that is being read from memory by a GSC+ guest, this observation leads to definition of a mechanism in which any devices that sit between the GSC+ guest and memory can be fetching data ahead along a particular address stream in order to minimize the memory latency. A GSC guest master issues a prefetch hint by asserting XQL along with ADDVL in the address cycle of a DMA read. Several different prefetch alternatives have actually been considered for UTurn. The type that has been chosen to implement is deterministic prefetch. The implication is that if UTurn receives a prefetch indication from a guest, the guest will at some point in the near future request the next sequential block of data in the DMA stream. Address and guest ID matching are still required at the time of actual request, but UTurn can deterministically prefetch on behalf of the guest without the concerns that speculative prefetch presents regarding stale data and data retirement.

The goal of prefetch is to minimize effective memory access time. Prefetched data can be supplied directly to a requesting guest, eliminating the bus–stalling waits that occur with connected reads when memory latency is long, and minimizing the need for busy/retry sequences or pended read transactions as a way to increase effective bus utilization.

The design challenges with prefetch include selection of a sufficient number of prefetch buffers, control of prefetches in progress and matching of requests to those prefetches, and addressing issues of coherence and correctness.

Support of prefetch tends to benefit individual guest maximum throughput, whereas pended read transactions tend to improve aggregate GSC+ bus throughput. Prefetch performance benefits derive from the reduced average read access time for prefetched data. The effectiveness of prefetch in UTurn will ultimately be limited if more than one DMA stream is interleaved per guest, or if traffic on Runway prevents UTurn from issuing prefetch transactions at a rate commensurate with GSC+ read requests.

An observation to be made at this point is that UTurn will always prefetch cache line quantities from memory. For Runway/Tornado, a cache line is 32 bytes. Transactions on GSC+, however, may be 16 byte quantities. This means that UTurn may indeed prefetch data that the guest has not explicitly requested. The guest may only wish to access the first half of the next line (the next 16 bytes of data), but UTurn must prefetch the entire line. Rules can (and must) be imposed on both guests and related software to ensure that this prefetch case does not lead to errant system operation.

To quantify prefetch performance, assume that a single DMA stream is active per GSC+ guest, and that each guest is correctly issuing the prefetch hint. All DMA read data, with the exception of the the first

access in a stream, will be supplied out of a prefetch buffer in UTurn. The succeeding analysis will also consider whether or not the time to prefetch across Runway exceeds the interarrival time for GSC+ requests, making Runway performance the limiter.

First, assume that GSC+ read request interarrival time does not exceed (Runway + memory) latency – this assumption will be revisited momentarily. Now, due to UTurn internal synchronization requirements and location of the prefetch buffers in the Runway clock domain, UTurn internal latency is incurred even on a prefetch hit. This latency is based on the internal UTurn delays considered earlier for connected and pended reads, and is a total of 19 Runway cycles and 3.5 GSC+ (round up to 4) cycles. This latency is inserted between the GSC+ read address cycle and the first GSC+ data cycle. Of this delay, one of the GSC+ cycles overlaps the turn cycle defined for GSC+ connected reads; the remaining (19R + 3G) cycles are visible to the GSC+ guest. Because of this delay, the individual guest theoretical maximum bandwidth for connected DMA reads can not be sustained through UTurn using only prefetching. Moving the prefetch buffers into the GSC+ clock domain to increase DMA read performance is possible, but makes UTurn internal design highly irregular and was considered an unreasonable risk.

In addition to UTurn–internal queueing and synchronization latencies, UTurn's GSC arbitration logic adds a dead cycle on GSC every time mastership changes between guests. This cycle is in addition to the restore cycle that follows the last read data cycle as part of every read transaction.

Accounting for UTurn–internal latencies and GSC arbitration overhead, a table of minimum sustainable interarrival times for guest–mastered (DMA) reads can be assembled. Note that a guest may elect to master multiple connected reads per bus tenure, in which case these interarrival times could be reduced by something less than one GSC cycle on average.

TABLE: Minimum sustainable interarrival times for GSC+ DMA reads

Transfer size	GSC+ cycles	extra arb cycle	UTurn internal delay cycles	Read interarrival times (ns) ; Runway/GSC+ frequency (in MHz) =				
				80 / 33.3	100 / 33.3	120 / 33.3	120 / 40	120 / 50
16	7G	1G	19R + 3G	568	520	489	434	379
32	11G	1G	19R + 3G	688	640	609	534	459

Now, the assumption of (Runway + memory) latency versus GSC+ read interarrival time must be explored. It is assumed that UTurn–initiated memory read requests can be pipelined on Runway. The worst case scenario, then, is when GSC+ access patterns come closest to exposing each GSC+ read to the (Runway + memory) latency. This will occur when back–to–back requests on GSC+ are along the same DMA stream. To calculate the Runway prefetch time, we must sum the Runway request and response arbitration latencies, the memory access time, and the actual Runway transfer time. The following table uses arbitration assumptions documented earlier to calculate time delays for various Runway frequencies. Memory latency values were noted earlier (25.5 Runway cycles). UTurn internal delay for sourcing the prefetch read request and for storing the return data is 11 Runway cycles (7 for request, 4 for storage).

TABLE: Runway/memory prefetch read time

Case	Delay	Delay components	Total read time, ns 80 MHz	100MHz	120MHz
best	0R+25.5R+5R+11R	R_arb_b+mem_access+R_xfer+int	519	415	346
typical	5R+25.5R+5R+11R	R_arb_t+mem_access+R_xfer+int	582	465	388
worst	12R+25.5R+5R+11R	R_arb_w+mem_access+R_xfer+int	657	535	446

Comparing the interarrival time and Runway/memory latency tables, note that any time the Runway read time exceeds the GSC+ read interarrival time for a given Runway/GSC+ frequency, Runway is the performance limiter. The tables show that a worst case load on Runway limits throughput when 16 byte reads are being performed and either Runway frequency is low or GSC+ frequency is high; otherwise, GSC+ is the limiter. Based on the data from the preceding two tables, the maximum single–stream DMA read bandwidth sustainable through UTurn is now calculated, assuming a single prefetch buffer per DMA stream and use of connected reads on GSC+. To get the minimum read arrival times (in GSC+ cycles) in the table below, the times shown in the preceding two tables are compared and the largest time value is converted to GSC+ cycles. Throughput is then calculated based on this cycle count.

TABLE: GSC+ connected read with prefetch – aggregate sustainable DMA read throughput (single prefetch buffer and single DMA stream active per guest; single DMA read per guest bus tenure)

Runway case	Transfer size	Cycle counts and throughputs; Runway / GSC+ frequency =							
		80 / 33.3 MHz		100 / 33.3 MHz		120 / 33.3 MHz		120 / 40 MHz	
		Cycle count	**Thruput (MB/ sec)**	Cycle count	**Thruput (MB/ sec)**	Cycle count	**Thruput (MB/ sec)**	Cycle count	**Thruput (MB/ sec)**
Best	16	19	**28**	18	**29**	17	**31**	18	**35**
	32	23	**46**	22	**48**	21	**50**	22	**58**
Typical	16	20 [1]	**26** [1]	18	**29**	17	**31**	18	**35**
	32	23	**46**	22	**48**	21	**50**	22	**58**
Worst	16	22 [1]	**24** [1]	18 [1]	**29** [1]	17	**31**	18 [1]	**35** [1]
	32	23	**46**	22	**48**	21	**50**	22	**58**

[1] Throughput limited by Runway prefetch rate.

Note that DMA read throughput is independent of Runway loading if 32 byte GSC+ read transactions are used.

The single–guest sustainable DMA read throughput is less than the aggregate rate due to GSC arbitration requirements and associated overheads. The GSC specification requires that after relinquishing bus ownership, a guest must see its bus grant signal released before it can re–assert a bus request. Between this protocol requirement and UTurn's implementation, an additional 4 GSC cycles of arbitration overhead are added each time a guest relinquishes and then attempts to regain bus ownership. For optimum single–guest DMA read performance, guests should be designed such that they can optionally perform multiple read transactions per bus tenure. This capability will hurt the performance of other guests on the same bus segment, so caution should be exercised when enabling the capability. Guests should always have the ability to be configured to perform only 1 transaction per bus tenure.

Taking the extra arbitration cycles into account, a single guest connected DMA read throughput table can be constructed. All values in this table are limited by GSC interarrival times, and are not affected by any of the expected Runway latency scenarios noted earlier.

TABLE: GSC+ connected read with prefetch – single guest sustainable DMA read throughput (one DMA read per guest bus tenure)

Runway case	Transfer size	Cycle counts and throughputs; Runway / GSC+ frequency =							
		80 / 33.3 MHz		100 / 33.3 MHz		120 / 33.3 MHz		120 / 40 MHz	
		Cycle count	**Thruput (MB/ sec)**	Cycle count	**Thruput (MB/ sec)**	Cycle count	**Thruput (MB/ sec)**	Cycle count	**Thruput (MB/ sec)**
Best/ Typ/ Worst	16	23	**23**	22	**24**	21	**25**	22	**29**
	32	27	**39**	26	**41**	25	**42**	26	**49**

3.5.4. Pended DMA reads with prefetch

When pended DMA read transactions are combined with memory prefetching, the single stream memory latency impact can be minimized while aggregate bus throughput is improved. In this section, data from the pended–only and prefetch–only sections will be joined to assess the benefit of implementing both. With prefetch being employed to minimize the impact of memory latency on a single DMA stream, and with pend capability facilitating initiation of several simultaneous DMA reads, an interesting I/O system performance range is implementable. As seen earlier, the throughput for a single DMA stream is actually slightly worse when pended read transactions are utilized on GSC+, independent of prefetch capability, due to the need for UTurn to arbitrate for GSC+ to issue the read return. The benefits of combining prefetch and pended reads are really seen, however, when a total system performance picture is drawn. Since GSC+ is not locked up while waiting for data even from the prefetch buffer, other transaction traffic such as inbound DMA data or outbound DIO data (graphics or other) can flow through UTurn and across GSC+, improving overall system throughput. In addition, there are a limited number of prefetch buffers, and if the number of outbound DMA streams active exceeds the available prefetching resources in UTurn (or if a particular DMA device is not capable of indicating prefetch hints), the memory latency will be exposed to that DMA stream. Connected reads in this case plug up the entire GSC+ bus for a significantly longer time, whereas pended reads simply allow that much more time for other GSC+ bus activity to proceed. A combination of the two clearly benefits overall system performance.

To calculate the single guest pended DMA read throughput when prefetching is in use, the basic pended read cycle count is combined with UTurn–internal latencies and expected GSC arbitration delays. The UTurn–internal latencies for pended readswith prefetch are longer than those for connected reads with prefetch by the amount of GSC arbitration latency encountered in the DMA read return phase. This additional arbitration latency was calculated for pended reads without prefetch. GSC arbitration latencies for the request phase will be identical to those calculated for pended reads without prefetch. In this section, only bus configurations of guests issuing pended reads will be considered. Combining transaction lengths and delays, a table of single–guest throughput values can be generated.

First, we calculate the UTurn–internal latency for a pended read of prefetched data, which is the sum of the UTurn–internal delay for a prefetch hit plus the arbitration delay for the read return portion of the read.

Earlier, it was noted that the UTurn–internal latency for a prefetch hit is 19 Runway cycles plus 4 GSC cycles. One of these cycles overlaps a bus tristate cycle required by the bus protocol, so the net additional delay due to UTurn internal logic is 19 Runway cycles and 3 GSC cycles. The arbitration delay for the data return was calculated when pended reads without prefetch were analyzed. This table is reproduced here (note the assumption that all other guests are performing either DMA writes or pended DMA reads; connected DMA reads will cause the arbitration delays to be longer).

TABLE: GSC+ host (UTurn) arbitration delay for issuance of read returns – connected reads disallowed

Case	Transaction size	GSC+ transaction length (GSC+ cycles)				**UTurn GSC+ arbitration latency for read returns**
		Pended read request	Pended read return	Write	Average	
Best (1 cycle arb delay)	16	2	6	7	5	**1**
	32	2	10	11	8	**1**
Typical (0.5 * (best+worst))	16	2	6	7	5	**2**
	32	2	10	11	8	**3**
Worst (0.5X avg trans length)	16	2	6	7	5	**3**
	32	2	10	11	8	**4**

Now, combine the GSC+ host arbitration delay numbers with the fixed UTurn internal delay cycles to come up with the number of additional delay cycles (in excess of the minimum required by the protocol) between the address cycle and the return notification cycle of a pended DMA read that encounters a prefetch hit.

TABLE: UTurn–internal latency – pended DMA reads with prefetch

Case	Transaction size	Pended DMA read return delay; Runway / GSC+ frequency =							
		80 MHz / 33.3 MHz		100 MHz / 33.3MHz		120 MHz / 33.3MHz		120 MHz / 40 MHz	
		Cycle counts	**GSC+ cycles**	Cycle counts	**GSC+ cycles**	Cycle counts	**GSC+ cycles**	Cycle counts	**GSC+ cycles**
Best	16 / 32	19R+4G	**12**	19R+4G	**11**	19R+4G	**10**	19R+4G	**11**
Typical	16	19R+5G	**13**	19R+5G	**12**	19R+5G	**11**	19R+5G	**12**
	32	19R+6G	**14**	19R+6G	**13**	19R+6G	**12**	19R+6G	**13**
Worst	16	19R+6G	**14**	19R+6G	**13**	19R+6G	**12**	19R+6G	**13**
	32	19R+7G	**15**	19R+7G	**14**	19R+7G	**13**	19R+7G	**14**

The pended read protocol allows a pended read request to be issued by a guest immediately following receipt of a pended read data return to that guest. UTurn will support this timing as long as the guest asserts its bus request earlier than the last data cycle of the read return (and as long as other higher–priority bus modules are not requesting the bus). Thus, the arbitration delays encountered in the single–guest

connected read case are not present for pended reads. By combining the transaction length information and the UTurn–internal delay data, single–guest performance for pended DMA reads with prefetch can be calculated.

TABLE: Individual GSC+ guest performance through UTurn –pended DMA reads with prefetch (single pended transaction per guest, connected reads by other guests disallowed)

Case	Xfer size	cycles in basic trans-action	Runway/GSC+ frequency combination											
			80 MHz / 33.3 MHz			100 MHz / 33.3 MHz			120 MHz / 33.3 MHz			120 MHz / 40 MHz		
			Arb/ Rway delay	Total cycles	**Thru-put (MB/s)**	Arb/ Rway delay	Total cycles	**Thru-put (MB/s)**	Arb/ Rway delay	Total cycles	**Thru-put (MB/s)**	Arb/ Rway delay	Total cycles	**Thru-put (MB/s)**
Best	16	8	12	20	**26**	11	19	**28**	10	18	**29**	11	19	**33**
	32	12	12	24	**44**	11	23	**46**	10	22	**48**	11	23	**55**
Typ	16	8	13	21	**25**	12	20	**26**	11	19	**28**	12	20	**32**
	32	12	14	26	**41**	13	25	**42**	12	24	**44**	13	25	**51**
Worst	16	8	14	22	**24**	13	21	**25**	12	20	**26**	13	21	**30**
	32	12	15	27	**35**	14	26	**41**	13	25	**42**	14	26	**49**

Aggregate bandwidth for the case of pended reads with prefetch will depend upon the amount of queue space available in UTurn, the UTurn–internal latency on a prefetch hit, and the number of outstanding pended reads that modules present in a particular I/O system might generate concurrently. The maximum aggregate GSC+ pended DMA read bandwidth can be maintained through UTurn with 4 – 5 guests generating pended reads. Because of UTurn's bus traffic prioritization algorithm, which prioritizes read returns at the highest level always, read requests and read responses are not interleaved optimally. Steady–state, this results in a "bursty" bus behavior wherein all guests issue requests back–to–back, and after the first response is ready, all responses to issued requests are issued back–to–back. This results in sub–optimal bus cycle utilization for busses with only 2 or 3 requesting guests. The following table estimates the aggregate pended DMA read throughput (with prefetch enabled) as a function of the number of guests. A "typical" UTurn–internal delay cycle count is assumed; arbitration on GSC for read returns will likely only have to wait for a DMA read request in progress, and UTurn's expected arb delay in this case is just under 2 GSC cycles. In order to achieve the levels of aggregate performance indicated, guests must be able to own the bus for the minimum number of cycles in the request phase. One cycle after a grant is issued, the guest must drive an address cycle, and the guest must release its bus request during the address cycle as allowed by the GSC+ specification. If the guest waits until the cycle after the bus request to de–assert its bus request, or if the guest waits an additional cycle after being granted the bus to drive an address, the maximum throughput numbers shown will be reduced by up to 20%.

Each throughput value shown in the table is calculated based on an expected request burst / response burst combination. The total length of such a combination is calculated by starting with the number of UTurn–internal delay cycles and dividing this by the length of a pended DMA read request (assumed to be 3 cycles for this table – 2 required by the protocol, and 1 due to arbitration request–to–grant delay inside UTurn). The result of this division is the number of requests that can issued on GSC before the first read data is ready to be returned. If the number of guests is small, the total number of cycles in a request/response burst is limited by the number of guests, since each guest can only issue one pended read at a time. The total cycles in a burst takes into account the requests, any dead time to the first response, and all cycles

requires to complete responses for all outstanding requests. The throughput shown in the table will be shared equally among the active guests.

TABLE: Aggregate DMA read performance through UTurn –pended DMA reads with prefetch (single pended transaction per guest, connected reads by other guests disallowed)

Number of guests	Xfer size	Runway/GSC+ frequency combination											
		80 MHz / 33.3 MHz UTurn–internal delay: 13 Gclks			100 MHz / 33.3 MHz UTurn–internal delay: 12 Gclks			120 MHz / 33.3 MHz UTurn–internal delay: 11 Gclks			120 MHz / 40 MHz UTurn–internal delay: 12 Gclks		
		Total bytes read	Total cycles	**Thru-put (MB/s)**	Total bytes read	Total cycles	**Thru-put (MB/s)**	Total bytes read	Total cycles	**Thru-put (MB/s)**	Total bytes read	Total cycles	**Thru-put (MB/s)**
1	16	16	21	**25**	16	20	**26**	16	19	**28**	16	20	**32**
	32	32	25	**42**	32	24	**44**	32	23	**46**	32	24	**53**
2	16	32	27	**39**	32	26	**41**	32	25	**42**	32	26	**49**
	32	64	35	**60**	64	34	**62**	64	33	**64**	64	34	**75**
3	16	48	33	**48**	48	32	**50**	48	31	**51**	48	32	**60**
	32	96	45	**71**	96	44	**72**	96	43	**74**	96	44	**87**
4	16	64	39	**54**	64	38	**56**	64	37	**57**	64	38	**67**
	32	128	55	**77**	128	54	**79**	128	53	**80**	128	54	**94**
5	16	80	45	**59**	80	44	**60**	80	44	**60**	80	44	**72**
	32	160	65	**82**	160	64	**88**	160	64	**88**	160	64	**100**
6	16	96	53	**60**	80	44	**60**	80	44	**60**	80	44	**72**
	32	192	77	**83**	160	64	**88**	160	64	**88**	160	64	**100**

3.6. DIO writes

UTurn can master two different types of writes on GSC – fixed length writes and variable length writes. UTurn will master fixed length writes in one of two sizes: 1–4 bytes (single word write) and 8 bytes (2–word write). UTurn can master variable length writes with data payloads of 1 to 8 words. UTurn supports the fast DIO write timing documented in the GSC specification, and can thus support the maximum theoretical DIO write performance on the GSC bus. DIO write sequences can be arbitrarily long, supporting lengthy graphics write sequences. The table below documents DIO write performance for fixed–length writes of various sizes and ”modes”. Only the 4–byte (single word) ”fast” case is shown due to the special design considerations for fast READYL acknowledge and accounting for restrictions involving changes in the target guest for back–to–back writes. These concerns can be more easily covered in the double word write case.

TABLE: UTurn–mastered (DIO) write performance – Fixed length writes

Transfer type	Transfer size (bytes)	Transfer "mode"	GSC+ cycles	Maximum sustained bandwidth (MBytes/sec); GSC+ frequency =				
				24 MHz	32 MHz	33.3 MHz	40 MHz	50 MHz
Write	8	either	3	64	85	88	106	133
	4	fast	2	48	64	66	80	100
	4	reg.	3	32	42	44	53	66

UTurn can master variable–length writes (writeVs), and can transfer the data either one word per GSC cycle (1.5X mode) or 2 words per cycle (2X mode). Performance for variable–length writes at various GSC bus frequencies is shown in the following table. Note that this table assumes long streams of writes and READYL acknowledgement timing that will not limit performance. GSC bus arbitration overhead is not considered, nor is the effect of DMA traffic or other non–DIO write activity.

TABLE: UTurn–mastered (DIO) write performance – Variable length writes

Transfer mode	Transfer size (bytes)	GSC+ cycles	Maximum sustained bandwidth (MBytes/sec); GSC+ frequency =				
			24 MHz	32 MHz	33.3 MHz	40 MHz	50 MHz
2X	32	5	153	204	213	256	320
	28	5	134	179	186	224	280
	24	4	144	192	200	240	300
	20	4	120	160	166	200	250
	16	3	128	170	177	213	266
	12	3	96	128	133	160	200
	8	2 [1]	96	128	133	160	200
	4	2 [1]	48	64	66	80	100
1.5X	32	9	85	113	118	142	177
	28	8	84	112	116	140	175
	24	7	82	109	114	137	171
	20	6	80	106	111	133	166
	16	5	76	102	106	128	160
	12	4	72	96	100	120	150
	8	3	64	85	88	106	133
	4	2 [1]	48	64	66	80	100

[1] Assumes optimum READYL timing and consecutive transactions to same guest

3.7. DIO reads

Reads mastered by UTurn on GSC (DIO reads) can be performed in either a connected or a pended mode, depending on the capabilities of the guest and on the state of UTurn. UTurn can only master single word

and double word reads on GSC. The DIO read path is a low throughput path, and should not be used for bulk data transfer. As with DMA reads, performing a DIO read in a pended fashion has slightly poorer performance than an equivalent connected read, but pended DIO reads allow more bus cycles to be available for use by other devices when a guest is slow to supply data in response to a read request. Guest devices can vary widely in their read data access times. A GSC guest that is a bus converter to another bus (to EISA, say) could have to wait for a read to complete on a lower, slower bus before being able to respond to a read request. Reading a GSC guest internal register, however, may be a relatively fast operation.

In this performance analysis, the guest is assumed to respond to a read request with minimum internal delay. Specifically, the access time for the read data is assumed to occur during the bus turn–around cycle, meaning that the guest can drive data back to UTurn on the second cycle following the address cycle of the read. At a system level, the DIO read throughput provided to a given Runway–based processor by UTurn includes the Runway read_short transaction time, the UTurn–internal latency for the read transaction, the GSC read transaction time, and any arbitration delays encountered on GSC or Runway due to bus loading or design restrictions. The following table summarizes the bus cycles consumed in a DIO read initiated by a given processor on Runway and directed to a GSC guest.

For Runway arbitration delay values, use the same numbers as were used in the DMA read performance calculation (Best = 0 additional cycles; typical = 3 cycles; worst = 7 cycles). For GSC arbitration delay values, use an averaged version of the arb delay calculated for pended DMA read returns (averaged between transaction sizes within a given performance case, yielding best = 1, typical = 3, and worst = 7).

Delay component	Runway cycles	GSC+ cycles (1 word read)	GSC+ cycles (2 word read)
Runway read request arb delay	0 / 3 / 7		
Runway read request	1		
UTurn internal delay to receive and queue request	6		
Synchronize read request to GSC domain		1.5	1.5
De–queue and arb for GSC		1.5	1.5
GSC arbitration delay (best/typical/worst)		1 / 3 / 7	1 / 3 / 7
GSC read (single word)		3	4
UTurn internal delay to queue, sync, and dequeue read return	21	.5	.5
UTurn Runway arbitration delay (best/typical/worst)	0 / 3 / 7		
Read return transfer time	1		
Processor delay before next read request	2		
Totals	**31 / 37 / 45**	**7.5 / 9.5 / 13.5**	**8.5 / 10.5 / 14.5**

Using this data, a performance table can now be assembled for various GSC and Runway frequency combinations. Partial GSC cycle counts are rounded up before being used in the table. These numbers are both single guest and multiple guest values with respect to a single processor. Current and planned processors can only have one I/O read outstanding at a time.

TABLE: Connected DIO read throughputs – single guest and aggregate per processor (throughput between a given processor and one or more GSC guests)

Case	Transfer size	Fixed cycle counts	Arb delay cycles	Total cycles	Runway / GSC+ frequency (in MHz) =		
					100 / 33.3	120 / 33.3	120 / 40
					Thruput (MB/s)	**Thruput (MB/s)**	**Thruput (MB/s)**
Best	8	31R + 8G	1G	31R + 9G	**13**	**15**	**16**
	4	31R + 7G	1G	31R + 8G	**7**	**8**	**8**
Typical	8	31R + 8G	3R+3G	34R + 11G	**11**	**13**	**14**
	4	31R + 7G	3R+3G	34R + 10G	**6**	**6**	**7**
Worst	8	31R + 8G	7R+7G	38R + 15G	**9**	**10**	**11**
	4	31R + 7G	7R+7G	38R + 14G	**5**	**5**	**6**

Pended DIO reads are really intended for efficient GSC bus utilization. Performing a DIO read in a pended fashion adds a minimum of 3 GSC cycles to the completion time for a read; a more typical value will be 5 extra cycles. A performance table is generated below for the minimum delay pended DIO read case, but keep in mind that the real benefit of pended DIO reads is to free GSC bus cycles when a guest knows it will take a very long time to get the data for a read issued by UTurn. By pending the read, the bus cycles are made available for other GSC traffic.

TABLE: Pended DIO read throughputs – single guest and aggregate per processor (throughput between a given processor and one or more GSC guests)

Case	Transfer size	Total cycles from connected DIO read case	Extra cycles due to pend	Total cycles in pend case	Runway / GSC+ frequency (in MHz) =		
					100 / 33.3	120 / 33.3	120 / 40
					Thruput (MB/s)	**Thruput (MB/s)**	**Thruput (MB/s)**
Best	8	31R + 9G	3G	31R + 12G	**11**	**12**	**14**
	4	31R + 8G	3G	31R + 11G	**6**	**7**	**7**
Typical	8	34R + 11G	5G	34R + 16G	**9**	**10**	**11**
	4	34R + 10G	5G	34R + 15G	**5**	**5**	**6**
Worst	8	38R + 15G	5G	38R + 20G	**8**	**8**	**9**
	4	38R + 14G	5G	38R + 19G	**4**	**4**	**5**

3.8. Conclusions

Three DMA performace points can be made based on the preceding calculations. First, use of 32 byte transfers is the single most effective way to get more bus bandwidth for any traffic level. Second, guests should support pended DMA reads, allowing aggregate bus bandwidth to be maximized. This is particularly important when several guests with similar performance requirements are active simultaneously on

GSC+. Third, prefetch should be implemented for use with DMA read transactions to maximize individual guest read throughput. Prefetch support is particularly important for guests performing connected DMA reads; performance for that guest is optimized, and generally more bus cycles remain available for other guests.

Two DIO transaction performance conclusions should be highlighted. First, 2–word (8–byte) transactions should be used whenever possible to maximize performance. Second, don't expect much performance for DIO read transactions (connected or pended).

4. Synchronization

The UTurn chip permits completely asynchronous clock relationships between the Runway bus and the GSC bus. Parts of UTurn operate in each of these clock domains. Please refer to the UTurn block diagram in the introduction chapter of this ERS. The dashed line in this diagram denotes the boundary between the domains. As this diagram shows, the following blocks straddle the boundary between Runway and GSC clocking: The DMA Read Return Queue, the Outbound Command Queue, the Inbound Queue, the Prefetch and DMA Read Return RAM, and the Inbound RAM.

The last four structures are implemented as dual port RAMs. The control logic for these RAMs will be able to insure that the same address will never be applied to both ports of the RAMs simultaneously.

The DMA Read Return Queue is implemented as a two port register file using scannable standard cell flip–flops. The small size of this structure (8 x 11) minimizes the area penalty incurred by using flip–flops.

The three queues listed above require synchronization. A Pipelined Synchronizer FIFO is used here. (For detailed information, refer to ”Pipelined Synchronizer/FIFO Invention Disclosure” by Vince Cavanna and Pradip Shankar.) Variations of this synchronizer have been used multiple times in various HP ASICs. The design achieves synchronization with minimal holdoff. The fact that it utilizes a FIFO makes it a natural fit for this application.

Briefly described, the synchronizer uses a FIFO to compensate for the latency inherent in all synchronizers. As long as the input pointer is kept sufficiently ahead of the output pointer, then holdoff will be minimal. The pointers are maintained in gray code format. These gray coded input and output pointer values are the entities that actually span the synchronization boundary. A chain of cascaded synchronous registers is used for this purpose. The delayed version of the pointer from the opposite side is used to decide whether a given operation can occur.

There are several other single bit control signals which cross the clock domains in each direction. Standard synchronizer techniques are used to achieve an acceptably low probability of synchronizer failure, while keeping latency at a minimum.

5. Outbound (Runway–to–GSC+) Transaction/Data Flow

The outbound portion of the block diagram consists of two duplicate decode blocks which contain a number of registers and comparators, an outbound command queue where transaction information (such as address, operation, and size) is stored in order, an outbound data return queue where transaction information corresponding to data read returns resulting from a previously mastered Runway read is stored in order, an HPA register block, a cache buffer for private read returns, and the cache coherency queue. Data corresponding to write commands is stored in the command queue with the command. The data corresponding to data read returns is stored in an internal ”RAM”.

Because there are two Runway clocks for every single UTurn internal Runway clock, the outbound Runway block receives two cycles worth of Runway activity in a single internal state. Runway transactions require either a single external Runway cycle or multiple external Runway cycles to complete, as shown in the following table.

Transaction Type	**Cycle Count**
All Coherent Transactions (TTYPE[2] == 1)	One Cycle
Read_Short	One Cycle (guaranteed ”idle” cycle follows)
Dir_Error / Broad_Error	One Cycle
DMA_Sync / Cache_Sync	One Cycle
Write_Short	>= Two Cycles
Cache–to–Cache Copy	>= Five Cycles
Memory Read Return	>= Four Cycles

Thus one internal state can contain a single Write_Short, or part of a Cache–to–Cache Copy or Memory Read Return Transaction or it could encompass two different transactions. The guaranteed ”idle” cycle after a read short is not required to be a Runway idle cycle but it must NOT be a read short or write short to this IOA.

5.1. Timing

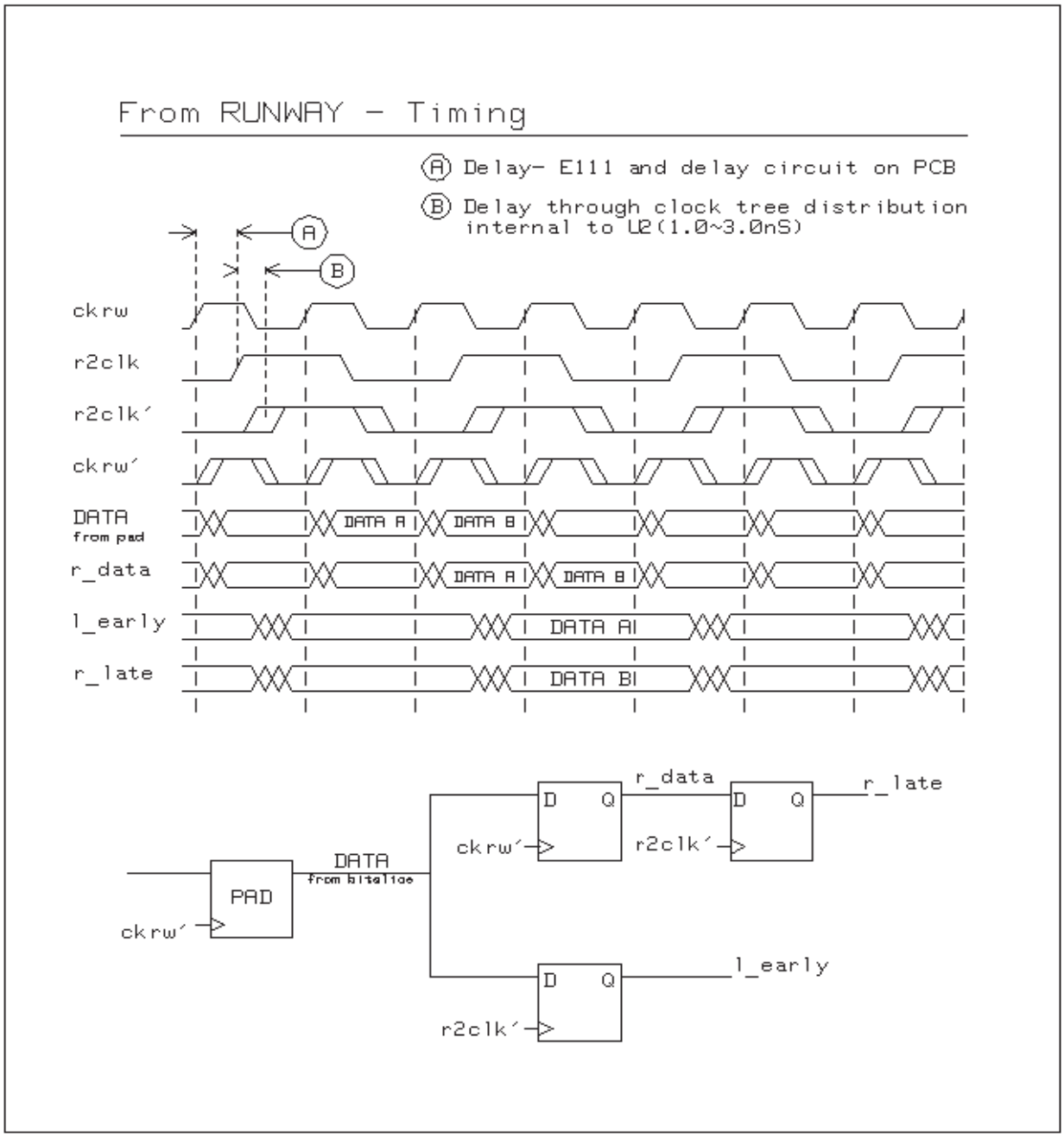
5.2. Blocks

The following sections describe the functional blocks required for outbound traffic, when UTurn is a slave to Runway.

5.2.1. Left and Right Side Decode Blocks

NOTE: The LEFT Runway cycle occurs BEFORE the RIGHT Runway cycle.

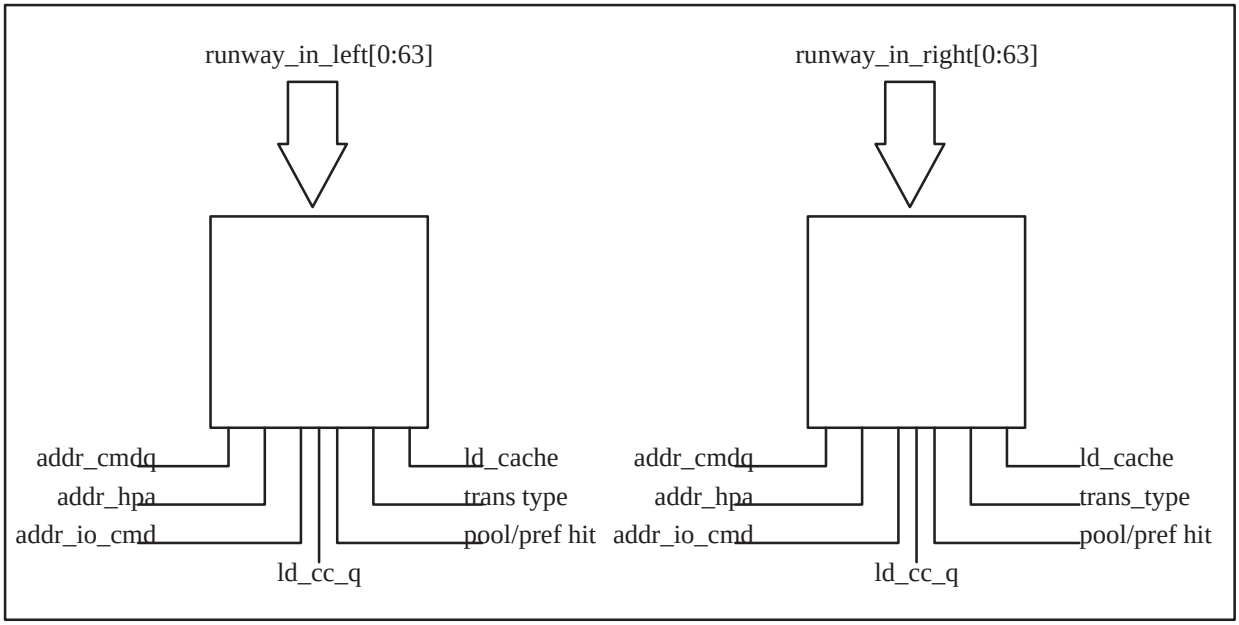
The left and right side decode blocks in the Runway receive block each receive one external Runway cycle worth of address and transaction information, corresponding to the two external Runway cycles within a single internal cycle, and determine the possible destination for the information. Possible destinations include the outbound data queue, the outbound command queue, RAM, a temporary queue



entry (for multi–cycle transfers), a temporary RAM buffer (for multi–cycle transfers), the HPA register block, the cache and the TLB. The results of the decode blocks go to the master control block which will steer the data/commands to the appropriate destination.

Also for every Runway cycle, a comparison to the pool and prefetch buffer Trans_ID registers is necessary to determine if this the first data cycle of a data read return destined for this I/O Adapter. This comparison will occur in the pool/prefetch block itself, independent of the left and right side decode blocks. The result of this comparison is additional input to the master control block to determine the appropriate destination for the data.

By processing the left and right side of an internal cycle in parallel, the receive logic can keep up with Runway bandwidth, as long as the destination of the left side and right side blocks are unique. The following diagram illustrates the left and right side decode blocks with appropriate destinations.



Internally, each decode block will be identical containing the following:

- Address Decode Logic
- Cache Address Comparator
- IO_Flex Comparator
- Address Bounds (IO_IO_Low and IO_IO_High) Comparator (two sets)
- Parity Checker
- Master_Id Comparator
- Transaction_Type Decode

5.2.2. Pool Buffers

There are eight pool buffers, the maximum number of outstanding requests is six (one for each GSC+ guest) and the maximum number of outstanding prefetch requests is six prefetch buffers (one for each GSC+ guest). There are only 8 pool buffers because each IOA is limited to eight outstanding transactions due to the 8 transaction IDs available. These buffers are stored by the Inbound Runway block and dumped to the Outbound Runway block. The Runway Trans_ID is used as an address to select the pool entry. . When a Runway cache–to–cache copy or memory read return is observed, the I/O Adapter compares the Master_ID and checks the valid bit of the pool entry indicated by the Trans_ID. If the Master_ID matches this I/O Adapter and the pool valid bit is set then the read return is destined for us.

Each entry is 45 bits wide and is organized as follows(The first entry is a normal pool entry, the second is a prefetch entry):

30k		2	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0 26 Not Used	27. .29 GSC+ Word IN Cache Line [4:2] =GSC+ ADDR [4:2]	30 31 GSC+ Trans Type [2:3]	32 34 GSC+ Guest ID [0:2]	35 DIAG-NOSTIC	36 NOTUSED	37 NOTUSED	38 CACHE	39 TLB	40 CON-NECT	41 TIME-OUT	42 TIMED_OUT	43 TIMED_ENAB	44 NO-TUSED	45 NOTUSED	46 RAMADDR	47 POOL=0	48 INUSE
0 29 30 Bit GSC+ ADDR		30 31 GSC+ Trans Type [2:3]	32 35 GSC+ Guest ID [0:2]	35 DIAG-NOSTIC	36 RE-TURNED	37 REQBYGSC+	38 NOTUSED	39 NOTUSED	40 CON-NECTED	41 TIME-OUT	42 TIMED_OUT	43 TIMED_ENAB	44 ERR_DUR_PRE	45 DISCARD	46 RAMADDR	47 PRE-FETCH=1	48 INUSE

Where:

GSC+ Word IN Cache Line – indicates which word in the cache line is the desired word – these bits are simply read and put in the command queue. The GSC+ block uses these bits. These bits are equivalent to the GSC+ bus address bits of [4:2].

GSC+ Trans Type – indicates the transaction type on GSC+ . These bits are put into the command queue.

GSC+ Guest ID[0:2] – this is the GSC+ Guest ID of the original transaction request.

DIAGNOSTIC – This is a diagnostic pool entry, this bit will be passed along into the RRQ entry so that the GSC block will know that this read return is not REALLY destined for a GSC guest.

CACHE – indicates data return destination is the cache.

TLB – indicates data return destination is the TLB.

CONNECTED – indicates the data return will be indicated by the Connected flag to GSC+. Data will be returned immediately – bypasses read return queue.

NOTE: CACHE and CONNECTED can both be set – this is used for semaphore data and indicates that the data should be put in both cache and the Read Return RAM to be returned to the GSC+ guest. This is the ONLY time that multiple bits of (CACHE, TLB, CONNECTED) can be set.

TIMEOUT – is used to determine if a timeout has occurred for this transaction.

TIMED_OUT – indicates a timeout has occurred for this transaction.

TIMED_ENAB – indicates that the read request has been seen on Runway and that the timeout process can begin.

ERR_DUR_PRE – indicates that there was a error during the data return for prefetch read. This error will get returned to the GSC guest if the guest tries to request the prefetched data.

DISCARD – indicates to outbound block to discard data when it returns because it is no longer needed. Only used for discarding prefetched data. See section on prefetching on inbound chapter. Outbound block will simply discard data and clear the INUSE bit.

RAMADDR – This is the LSB of the RAM address. The GSC+ Guest ID makes up the rest of the address.

POOL(0)/PREFETCH(1) – this bit indicates what this entry is being used for – either a POOL entry or a PREFETCH entry.

INUSE – This bit indicates that this entry is currently being used and the contents are valid. If data is received and the trans_ID matches a pool entry that is NOT is use, the data will be discarded and a unexpected data return error will be logged.

30 Bit GSC+ ADDR – this is the prefetch address. It is compared against the next request from that guest to verify that the guest did in fact request this address location.

RETURNED – indicates that the prefetch data has been returned from Runway.

REQUESTED ON GSC+ – indicates that when the data is returned the request should be moved from the Pool to the command queue.

Each IOA only has eight trans_IDs available. The trans_IDs have a one–to–one mapping to the pool entries. When data is returned the trans_ID is used to address the pool entries. If the pool entry is ”IN-USE” then the data must be stored in the appropriate place. The following table shows where the data is stored:

Pool(0) / Prefetch(1)	CACHE	TLB	CON-NECTED	Data re-turned	Requested ON GSC+_	DATA DEST.	Outb RRQ entry
0	0	0	0	X	X	OUTBND RAM [{GSC+GUEST_ID,RAMADDR}]	YES
0	0	0	1	X	X	OUTBND RAM [{GSC+GUEST ID, RAMADDR}] Will fire Connected bit	NO
0	0	1	0	X	X	TLB	NO
0	1	0	0	X	X	CACHE	NO
0	1	0	1	X	X	OUTBND RAM and CACHE [(GSC+GUEST_ID,RAMADDR)]	NO
1	0	0	0	set data returned	0	OUTBND RAM [{GSC+GUEST_ID,RAMADDR}]	NO
1	0	0	0	X	1	OUTBND RAM [{GSC+GUEST_ID,RAMADDR}]	YES

Most responses will be queued in the Outbound Read Return Queue(RRQ), and the corresponding data will be stored in the Outbound RAM. However, responses intended for the TLB, cache, or connected

GSC+ responses, will bypass the Data Return Queue and the Outbound RAM and go directly to the TLB, cache, or GSC+, respectively. This is done to prevent deadlock.

Corresponding to each prefetch buffer are bits which indicate if the data has already been returned and is in the read return RAM, if the GSC+ transaction has been formally issued (requested), and finally if the buffer contents are valid (INUSE). An Outbound Read Return Queue entry will not be formed until the requested bit is set (indicating that the GSC+ request has been issue) and the outstanding bit is clear (indicating that the return data is valid in the Outbound RAM). It is the responsibility of the outbound control block to change the returned bit from false to true, once the data is valid in the RAM. The outbound control block must also change the inuse bit from true to false, once an entry corresponding to this prefetch is stored in the Read Return Queue. This is an indication to the inbound control block that the buffer (and its corresponding Trand_ID) is now available to reuse for a subsequent transaction.

5.2.3. Outbound Command Queue

The Outbound Command Queue is a synchronizer/fifo, which is loaded in the internal Runway domain and dumped in the GSC+ domain. All Runway transactions for which this I/O Adapter is a slave will be loaded in this queue in the order they are received from the Runway Bus. The queue is 26 entries deep, sized to accommodate a critical 3D graphics transaction size. Once the queue has 19 valid entries, UTurn will indicate STOP_IO. As it may require 3 cycles for UTurn to drive the STOP_IO line plus up to 4 cycles for this flow control to take effect throughout the system, 7 additional buffers ensure that the queue will not be overrun. It is only possible to completely fill the OUTQ if write shorts(or read shorts) are arriving as fast as possible(address cycle every other Runway cycle) and if the address cycle arrives on the ”right” side of the decode block. If the address cycle is on the left cycle the OutQ can only be filled with 25 entries.

There are three types of OutQ entries, they are, listed in the order they appear in the following table:

1) GSC DIO, PDC or GSC HPA register access
2) Runway HPA read, TLB command (U–turn)
3) Test entry – used for internal testing by PDC

Each entry is 120 bits wide and is organized as follows:

1	6	3	1	1	1	1	1	1	8				1	1	1	1	1	1	1	1	32	64
0=0(GSC+)	1 6 Runway TransID	7 9 Runway MstrID	10 PDC	11 IO_BROAD	12 READ/NWR	13 8BY-TES	14 TEST=0	15 NO-TUSED	16 23 Runway Byte Enable				24 RFE	25 RHE	26 NO-TUSED	27 MAYBEDIO	28 CO-ALESCE	29 ALL_BES	30 ADDR15X	31 ADDR2X	32 63 32 bit address	64 127 64 bit data
1=INBND	1 6 Runway TransID	7 9 Runway MstrID	10 NOTUSED	11 NOTUSED	12 NOTUSED	13 NOTUSED	14 TEST=0	15 NO-TUSED	16 HPAREAD	17 TLB_CMD	18 19 NOTUSED	20 23 CMD TYPE	24 NO-TUSED	25 NO-TUSED	26 ERR_RTN0	27 NO-TUSED	28 NOTUSED	29 NOTUSED	30 NOTUSED	31 NOTUSED	32 63 32 bit address	64 127 Not Used
1=INBND	1 –3 Not Used 4 Conn 5 Lock 6 Prefetch	7 9 GSC Guest ID	10 NOTUSED	11 NOTUSED	12 NOTUSED	13 NOTUSED	14 TEST=1	15 NO-TUSED	16 19 GSC Byte Enable			20 23 GSC TYPE	24 NO-TUSED	25 NO-TUSED	26 NO-TUSED	27 NO-TUSED	28 NOTUSED	29 NOTUSED	30 NOTUSED	31 NOTUSED	32 63 32 bit address	64 127 Not Used

Where:

Bit 0: (1/0) – indicates destination of this entry (INBND block/GSC+) – also called the “u–turn” bit

Runway TransID indicates the transaction ID (in the case of a Read_Short or Read_Burst, where the Trans_ID must be redriven on the corresponding read responses from the I/O Adapter)

Runway MstrID indicates the transaction master ID (in the case of a Read_Short, where the Master_ID must be redriven on the corresponding read responses from the I/O Adapter)

Runway Byte Enable specifies which of the 8 bytes within the addressed double word are being read or written (in the case of a Read_Short or Write_Short)

PDC indicates that this is an access to PDC address space

IO–BROAD indicates to the GSC+ block that this is an access to broadcast IO address space (address falls between IO_IO_LOW[_HV] and IO_IO_HIGH[_HV] and IOA is in PEEK mode).

READ/NWR indicates to the GSC+ block if this is a read short or a write short transaction

8BYTES is set whenever the Runway Byte Enable field indicates that this transaction is a double word (8 Byte) transaction.

TEST indicates this is a DMA test entry – this will be used to form a ”GSC” inbound DMA request

RHE indicates that the Runway side is in hard error mode.

RFE indicates that the Runway side is in fatal error mode.

ERR_ RTN0 is used to force a read of HPA registers to return all 0’s. This was originally intended to be for reading unimplemented registers but is now only used to return 0s on a read of the IO_STATUS register if the read is done while a command reset is in progress.

MAYBEDIO – This is a predecode for the GSC master state machine, indicating that the entry is not a “u–turn”, not a PDC access, not a broadcast and we are not in hard or fatal error mode.

COALESCE is set if an entry is a DIO write and its address is sequential with the address of the preceding OutQ entry.

ALL_BES is set if an entry is a DIO write and one or two full words of data are valid.

ADDR15X is set if an entry is a DIO write and its address falls into an I/O address space enabled for GSC 1.5X mode writes (variable length writes).

ADDR2X is set if an entry is a DIO write and its address falls into an I/O address space enabled for GSC 2X mode writes (variable length writes with data at twice the normal GSC rate).

32 bit address is the least significant 32 bits of the Runway 40 bit address.

64 bit data is used for Write Shorts, and contains the data being written

FOR UTURN =1 and TEST =0:

HPAREAD , TLBCMD, and TYPE are used as a 6 bit field to inform the Runway inbound block what to xtion to execute. These bits are encoded as follows:

HPAREAD	TLBCMD	TYPE	
1	0	1xxx	HPA normal read – address field indicates which register
1	1	1000	TLB Read – entry = OutQ[32:51]
1	1	1001	TLB TAG Read – entry = OutQ[32:51]
0	1	1100	TLB Purge Cmd – entry = OutQ[64:83]
0	1	1010	TLB Insert Cmd – entry = OutQ[64:83]
0	1	1001	TLB Direct Write – entry = OutQ[64:83]

FOR UTURN =1 and TEST =1:

The Connected, Lock, Prefetch, GSC guest ID, Byte Enable, GSC Type, and Addr fields are used to form a ”GSC DMA” entry in the inbound queue. These bits represent the Connect, LOCK, PREFETCH, GSC guestId, GSC byte enables, GSC trans type, and the GSC address (respectively) in the inbound queue.

5.2.4. Outbound Read Return Queue

The Outbound Read Return Queue is also a synchronizer/fifo, which is loaded in the internal Runway domain and dumped in the GSC+ domain. This queue stores all Cache–to–Cache Copies and Memory Read Returns information corresponding to reads mastered by this I/O Adapter. The queue is 6 entries deep which will accommodate one outstanding transactions from each of the six guests permitted per GSC+. This queue should never be in danger of overflowing, as each GSC+ guest can only have one

outstanding request at a time and this entry is not loaded until the guest makes the request. Therefore a prefetch request will remain in the pool/prefetch block until the GSC+ guest has actually made the request for the data..

Each entry is 11 bits wide and is organized as follows:

0 2	3	4 5	6 8	9	10
GSC+ Guest ID	RAM Address	GSC Type[2:3]	GSC+ Word in Line	Error	Diagnostic

Where

GSC+ Guest ID indicates which GSC+ guest this read response is destined for

RAM Address points to the internal RAM location where the return data is stored; this bit is combined with the GSC+ GuestID to form the address of the RAM location (cache line).

GSC Type[2:3] reflect the address cycle value of the least significant two bits of the GSC+ Transaction Type field and indicate the amount of data to be returned (one, two, four, or eight words)

GSC+ Word in Line indicates the first word that was actually requested within the cache line.

Error – indicates that an error occured while preforming the read of the requested data.

Diagnostic – used for returning data to the ”GSC guest” during a PDH induced diagnostic DMA read. This bit will tell the GSC state machine to ”return” the data and also write the data into the inbound RAM.

5.2.5. Connected Read Returns

Since many GSC+ guest mastered reads are connected (meaning that the GSC+ bus is occupied by the mastering guest until the return data is issued by the I/O Adapter), a mechanism is required to bypass the Outbound Data Queue for these read returns. On a connected read return, data will be stored in a location based on the requesting GSC guest and an internal RAM address bit, and a signal will fire to indicate that the data is valid on chip. The “data valid” signal must be synchronized from the internal Runway clock doamin to the GSC+ domain.

5.2.6. Outbound and Prefetch RAM

There are 12 RAM locations associated with returning data, two for each GSC+ guest. Each RAM location is 256 bits wide but in fact is organized 128 bits wide so the RAM can be thought of as 26x128. The RAM is stored 128 bits (1/2 cache line) at a time. It is the responsibility of the multi–cycle state machine to coordinate the left side and right side data cycles to form a full 128 bit unit to store to the RAM. Data can be stored temporarily in 64 bit units in the temporary RAM buffer, and its corresponding entry information is stored in temporary queue entry until all data is valid on chip.

Since each GSC+ guest ”owns” two locations in the RAM, the GSC+ guest ID forms the MSBs of the RAM address. The LSB of the RAM is controlled by the Pool/Prefetch block. The LSB address will simply alternate for each data return for a given GSC+ guest.

5.2.7. HPA Register Block

The HPA registers will be distributed among three blocks, Runway outbound, GSC+, and Runway inbound. The HPA registers in the Runway outbound block include the address range registers

(IO_IO_LOW and IO_IO_HIGH), the IO_FLEX register, the IO_FLEX_MAP register, and the local and global IO_COMMAND registers. (Reference the architectural chapter for a detailed explanation on the functionality of these registers.

The Runway HPA registers are loaded by the outbound block and dumped by the inbound block. Because the outbound block must decode both reads and writes of these registers, the writes will occur immediately but the read data will be queued in the outbound command queue. The GSC+ block will forward these queue entries to the inbound queue, where the inbound block can drive the response onto Runway. When the outbound block receives a register access for a register in the GSC+ or inbound block, the transaction is forwarded as a queue entry in the outbound command queue, the Bit location HPA Reg indicates to the GSC+ block that this command queue entry is an access to the HPA registers.

The outbound control block will detect errors and fire error lines indicating the error type. It is the responsibility of the Error block to coalesce and decode the error lines from the inbound block and the outbound block. The error block will create the entries for IO_STATUS, and IO_ERROR registers. The inbound block will also use this information to drive ERROR transactions onto Runway. Additional information such as master_id and transaction_id must be logged if the inbound block is to drive a DIRECTED_ERROR transaction.

5.2.8. Cache (Private Read Returns)

Because the KittyHawk Memory System only stores in cache line quantities, on partial cache line writes, the I/O Adapter is required to perform a full cache line private read. The data return resulting from this read is stored in the cache buffer inside the I/O Adapter. Once the cached data is valid on chip, the corresponding partial cache line write can proceed, followed by a Runway WRITE_BACK to cast out the line.

It should be noted that this mechanism is also required for GSC+ guest mastered semaphores. However, on a semaphore, the return data must be stored in both cache and the predesignated connected read return buffer. Once the return is issued to the GSC+ guest, the guest will master a write to the cache line address, and ultimately a Runway WRITE_BACK will follow.

5.2.9. Cache Coherency Queue

The I/O Adapter is required to eavesdrop Runway coherent transactions and respond with status. Because two cycles of Runway information is received on every one internal Runway state, two cache line comparisons occur simultaneously (in the left and right side decode blocks), and the results are either driven out to the UTurn CC block or stored in the queue. A comparison consists of determining if the cache address is equal to the address of the received transaction, checking if bit 2 of the received transaction type is true, checking if the cache line buffer on chip contains a valid entry, and seeing that there are no errors on the received transaction. In most cases the cache will not contain private data and a COH_OK response can be given immediately. However, in the event of a cache line hit, UTurn will delay giving a COH_OK response until the cache line is no longer held private.

UTurn WILL ALWAYS respond COH_OK it is just a matter of when. Delaying the COH_OK response when there is cache hit will not impact performance because UTurn will own lines private for a very short time (sub–cache line writes). The only exception to this is when LSL is indicated on GSC+, UTurn will interpret this as a semaphore access, in this case UTurn can own a cache line private for a long time – however the system is locked up anyway due to the LSL bit.

Because cache tag compares occur on the fly, there will be no need to queue outstanding coherent checks due to backlog. However queueing is required when the I/O Adapter has a hit on line held private. Count-

ers are used to count how many COH_OK responses are queued up. This COH_OK responses are sent out as soon as the cache line is no longer private. There are 2 counters needed – one counts the responses before a delay is required (to wait for not private) and the other counts the responses after the delay. Each of the counters is larger than the minimum requirement.

There are a couple of corner cases involved with the cache comparisons. First, it is possible that the left side transaction contains a read private from this I/O Adapter, and the right side transaction contains a coherent transaction with the very same address as this read private. Therefore, whenever a read private on Runway is received which matches this I/O Adapter's master ID, the comparison registers in the left side and right side decode blocks must be loaded immediately to affect all subsequent accesses. Additionally, the left side and right side addresses must be compared to one another. When CCC responses are delayed until not private, UTurn must wait until the last data cycle of the write back has been put on the Runway bus before allowing the COH_OK responses to be put on the bus.

5.3. Transaction and Data Flow

The following sections describe the state machine control required for transactions and data flow in the outbnd block, when UTurn is a slave to Runway.

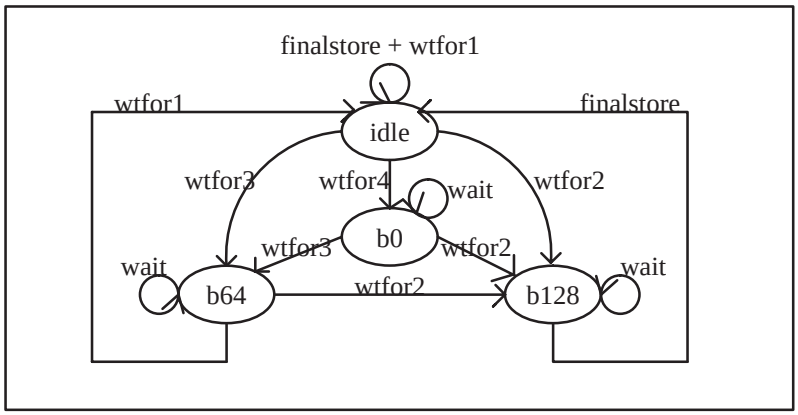
5.3.1. Multi–Cycle Transactions

Write_Shorts Cache–to–Cache Copies, and Memory Read Returns require more than one Runway external cycle, and in some cases more than one internal cycle(which is = to 2 external Runway cycles). As an example, the Write_Short transaction will span at least two external Runway cycles, with no inserted idle states. Internally, the transaction may be contained within one internal cycle or it may cross an internal cycle boundary, with the Address phase occurring during the latter half of the first internal cycle and the Data phase occurring during the first half of the second internal cycle.

Within a single transaction, the transaction / address information will be held in a temporary queue register until the data is valid on chip. Write_Short and Read_Short transactions will result in a command queue entry being formed. For Write shorts the queue entry contains the data coalesced with the command to be written into the command queue in one 128 bit quantity.

For read returns (and c2c_writes) data will be coalesced into writeable size blocks of 128 bits, corresponding to two external Runway cycles which could span a single internal Runway cycle or be distributed across the right and left halves of subsequent internal Runway cycles. As the final data block is stored in internal RAM, the temporary queue register is loaded into the tail of the read return queue.

The following state machine illustrates the progression through a multi–cycle transaction:



where finalstore means that transaction address and data are stored in their destination (whether it be the read return queue and RAM, the outbound command queue, or HPA), wtfor1 means that one data cycle is pending in which case the machine must return to idle where it can receive the remaining data from this transaction as well as any data or address of a subsequent transaction, wtfor2 means that two data cycles are pending, wtfor3 means that three data cycles are pending, and wtfor4 means that four data cycles are pending.

NOTE: Some of the transitions (like b64 to b128 can only occur if there are idle cycles within transactions) This is an issue still being discussed.

5.3.2. Single Cycle Transactions

The idle state in the previous state diagram is also the state where single cycle transactions may be handled entirely. The following table illustrates all possible combinations of two different transactions within a single internal Runway cycle. Both of these cycles are decoded in parallel by the left side and right side decode blocks, to determine the destination of the address and data. As long as the destination of the left side and right side cycles are unique, there will be no conflict and the I/O Adapter can process two transactions at one half of the external Runway frequency and keep up with the external bandwidth.

Left Side Cycle	**Right Side Cycle**	**Resulting State (if not idle)**
Nop / Idle / Don't Care	Nop / Idle / Don't Care	
Nop / Idle / Don't Care	Coherency Check	
Nop / Idle / Don't Care	Read_Short	
Nop / Idle / Don't Care	Dir_Error / Broad_Error	
Nop / Idle / Don't Care	DMA_Sync / Cache_Sync	
Nop / Idle / Don't Care	Write_Short (First Cycle)	wait–for–1
Nop / Idle / Don't Care	Cache–to–Cache Copy (First Cycle)	wait–for–4
Nop / Idle / Don't Care	Memory Read Return (First Cycle)	wait–for–3
Coherency Check	Nop / Idle / Don't Care	
Coherency Check	Coherency Check	
Coherency Check	Read_Short	
Coherency Check	Dir_Error / Broad_Error	
Coherency Check	DMA_Sync / Cache_Sync	
Coherency Check	Write_Short (First Cycle)	wait–for–1
Coherency Check	Cache–to–Cache Copy (First Cycle)	wait–for–4
Coherency Check	Memory Read Return (First Cycle)	wait–for–3
Read_Short / Read_Burst	Idle (guaranteed)	

Left Side Cycle	Right Side Cycle	Resulting State (if not idle)
Dir_Error / Broad_Error	Nop / Idle / Don't Care	
Dir_Error / Broad_Error	Coherency Check	
Dir_Error / Broad_Error	Read_Short	
Dir_Error / Broad_Error	Dir_Error / Broad_Error	
Dir_Error / Broad_Error	DMA_Sync / Cache_Sync	
Dir_Error / Broad_Error	Write_Short (First Cycle)	wait–for–1
Dir_Error / Broad_Error	Cache–to–Cache Copy (First Cycle)	wait–for–4
Dir_Error / Broad_Error	Memory Read Return (First Cycle)	wait–for–3
DMA_Sync / Cache_Sync	Nop / Idle / Don't Care	
DMA_Sync / Cache_Sync	Coherency Check	
DMA_Sync / Cache_Sync	Read_Short	
DMA_Sync / Cache_Sync	Dir_Error / Broad_Error	
DMA_Sync / Cache_Sync	DMA_Sync / Cache_Sync	
DMA_Sync / Cache_Sync	Write_Short (First Cycle)	wait–for–1
DMA_Sync / Cache_Sync	Cache–to–Cache Copy (First Cycle)	wait–for–4
DMA_Sync / Cache_Sync	Memory Read Return (First Cycle)	wait–for–3
Write_Short (Last Cycle)	Nop / Idle / Don't Care	
Write_Short (Last Cycle)	Coherency Check	
Write_Short (Last Cycle)	Read_Short	
Write_Short (Last Cycle)	Dir_Error / Broad_Error	
Write_Short (Last Cycle)	DMA_Sync / Cache_Sync	
Write_Short (Last Cycle)	Write_Short (First Cycle)	wait–for–1
Write_Short (Last Cycle)	Cache–to–Cache Copy (First Cycle)	wait–for–4
Write_Short (Last Cycle)	Memory Read Return (First Cycle)	wait–for–3
Cache–to–Cache Copy (Last Cycle)	Nop / Idle / Don't Care	
Cache–to–Cache Copy (Last Cycle)	Coherency Check	
Cache–to–Cache Copy (Last Cycle)	Read_Short	
Cache–to–Cache Copy (Last Cycle)	Dir_Error / Broad_Error	

Left Side Cycle	Right Side Cycle	Resulting State (if not idle)
Cache–to–Cache Copy (Last Cycle)	DMA_Sync / Cache_Sync	
Cache–to–Cache Copy (Last Cycle)	Write_Short (First Cycle)	wait–for–1
Cache–to–Cache Copy (Last Cycle)	Cache–to–Cache Copy (First Cycle)	wait–for–4
Cache–to–Cache Copy (Last Cycle)	Memory Read Return (First Cycle)	wait–for–3
Memory Read Return (Last Cycle)	Nop / Idle / Don't Care	
Memory Read Return (Last Cycle)	Coherency Check	
Memory Read Return (Last Cycle)	Read_Short	
Memory Read Return (Last Cycle)	Dir_Error / Broad_Error	
Memory Read Return (Last Cycle)	DMA_Sync / Cache_Sync	
Memory Read Return (Last Cycle)	Write_Short (First Cycle)	wait–for–1
Memory Read Return (Last Cycle)	Cache–to–Cache Copy (First Cycle)	wait–for–4
Memory Read Return (Last Cycle)	Memory Read Return (First Cycle)	wait–for–3

In the case where one half of the transaction is a Nop, Idle cycle, or a Don't Care (meaning that the transaction was destined for another Runway module), then the second half of the internal cycle is guaranteed not to conflict with the first. If either half of the transaction is a coherency check, then again there should not be a conflict, as coherency checks are performed in parallel, and if queued, can be stored two at a time. Error transactions will be logged in a separate block, so conflicts will only occur if two errors occur within one internal cycle. In this case, the highest priority error will be logged. Sync transactions are similar to coherency checks, in that they can be processed in parallel and responded to immediately or queued two at a time. Because data corresponding to writes, cache–to–cache copies, and memory read returns will be stored in temporary RAM registers during data cycles, they could only conflict with another data cycle from a different transaction within the same internal cycle, which does not contain an address header, such as a memory read return. To prevent this conflict, two temporary RAM registers will reside per left and right side block. This will provide enough storage for back to back data cycles of different transactions, or back to back 16 byte data cycles within the same transaction, as the RAM will fill 128 bits at a time, requiring two internal cycles.

5.3.3. Prefetch Returns

If a prefetch return is no longer outstanding (meaning the data is in RAM) and the GSC+ guest just formally requested it, then the contents of the prefetch buffer need to be added to the outbound data read return queue. Because it is a GSC+ request which triggers the queueing, as opposed to a Runway request, this case introduces some specific timing concerns.

While the I/O Adapter is busy receiving valid addresses or data every Runway cycle, it will be unable to process the prefetch return. As soon as the path to the outbound data return queue is available, the outbound block must queue the prefetch return and mark the prefetch buffer invalid. The inbound block is not able to fill the buffer with a subsequent prefetch until it observes the invalid bit.

6. Inbound (GSC+ to Runway) Transaction Flow

6.1. Basic Description of Inbound Operation

The operation of the IOA processing transactions from GSC+ to Runway (inbound) is described below. All guest mastered GSC+ transactions are processed by the GSC+ block's slave state machine. The data for the transaction (if any) is stored in the Inbound RAM (InRAM), while the transaction information (GSC+ address, GSC+ transaction type, GSC+ guest id, GSC+ byte enables, GSC+ LSL line, connected read indication, prefetch indication) is stored in the Inbound Queue (InQ) for synchronization into the Runway clock domain (see the synchronization chapter of this ERS for more details). Since transactions are placed into the InQ in the order that they were issued on GSC+, and the queue maintains FIFO ordering, the ordering of transactions is preserved.

There are two possible types of transactions in the InQ. These two types of transactions have two different uses of the inbound queue fields differentiated by an ENTRY TYPE bit in the queue. The first is for transactions mastered from a GSC+ guest (memory read, memory write, semaphore, ...). The ENTRY bit is a zero for these GSC+ mastered transactions. The second is for data returns of reads of IOA internal HPA registers (IO_PDIR_BASE, IO_CONTROL,....), or data returns by a GSC+ guest in response to a processor generated IO read or PDC read. Transactions used to manipulate the IO TLB (TLB purge, insert, direct write) also use the 'data return' format. For data returns and TLB manipulation transactions the ENTRY bit is a one (1). Both types of transactions are placed in the InQ by the GSC+ receiver. GSC+ guest mastered transactions originate on GSC+, whereas HPA register reads, PDC reads, and TLB transactions originate on Runway and arrive via the Outbound Command Queue (OutQ). These two different types of transactions result in different types of entries in the InQ. See the section on hardware blocks for more detail.

Finally, there are also transactions fabricated by the IOA as the result of an external event (transfer of control, power fail warning), or an IOA detected error (Runway, GSC+, or IOA internal). These transactions are generated by assertion of a signal, and do not have a corresponding transaction in the InQ.

Data returns in the InQ come in two flavors. Both types have transaction information (Runway MASTER_ID, TRANS_ID, data return type) in the InQ. Data returns from IO reads (i.e. from GSC+ guests) have the data in the InRAM location assigned to InQ location. However, data returns from Runway side IOA internal HPA register reads are contained in the HPA registers, not in the InRAM. The appropriate register is read, and the data transferred directly to the Runway driver register when the data return transaction reaches the top of the InQ.

Once a guest mastered GSC+ transaction has reached the top of the InQ, a Runway address must be generated. Notice the address translation is necessary for two reasons. First, to provide the address bits necessary to extend a 32 bit IO virtual address (GSC+ address) to a 40 bit memory physical address (Runway address). Second, to provide the virtual index address bits necessary for processors to snoop their caches and maintain cache coherency. There are two ways in which the 40 bit physical address can be generated, either by '0' or 'F' extending the GSC+ address, or by looking up the address translation in the TLB (see the section below on address translation mechanisms).

For coherent transactions to be issued on Runway (so processors can check their caches), a virtual index must be read from the TLB, or generated when the system is running in REAL addressing mode. REAL addressing mode is only used for system boot up before the IO PDIR (IO Page DIRectory, the table in memory which has all the mappings of IO addresses to physical memory addresses) exists, and for acces-

sing the IO PDIR. In REAL mode, a virtual index is generated, but requires that the page be equivalently mapped (eight bits of the GSC+ address ARE the virtual index). Note that since the IO PDIR is used to service TLB misses, and since there is no dedicated TLB entry for the IO PDIR, the entire IO PDIR is required to be be placed in an equivalently mapped area in memory AT ALL TIMES and FOR ALL MODES.

In the normal address translation mode, once the transaction reaches the head of the InQ, operation proceeds as follows. The TLB is accessed using the IO virtual page number portion of the GSC+ address (part as address into the TLB, part as tag to determine if we actually got a TLB hit). If the TLB entry is present, then the Runway physical page number, the virtual index bits, the page type bits, and a TLB entry valid bit are read from the TLB. The transaction to be used on Runway is determined by the combination of the transaction type at the top of InQ, the GSC+ LSL line (LOCK internal to UTurn), and the page type bits. See the section below on Transaction map (GSC+–>Runway). This transaction is loaded into the Runway DRIVE REGISTER along with the Runway address, the virtual index bits, and any data associated with the transaction (WRITE transaction for example). Once the Runway DRIVE REGISTERS are loaded, the transaction is ready to go out onto Runway. Driving the transaction out is handled by the full speed Runway interface. See the section below on the 2 TO 1 INTERFACE.

If the transaction issued on Runway will have a response (i.e. a READ transaction where a data is expected), then the transaction information is stored in the Pool, so that the return data can be matched with the original transaction, and returned to the correct guest on GSC+. The TIMEOUT bit is cleared when the transaction is placed into the Pool. The TIMEOUT bit is set once the timeout counter produces a carry, and a transaction times out on the next carry from the timeout counter. This guarantees at least one timeout period has passed (possibly as much as two), and we consider that transaction to have timed out, and an error is logged. The timeout period is determined by the value loaded into the Runway_TIMEOUT HPA register.

If the TLB entry is not present, and if the IO_CONTROL_HV_MODE register has IO PDIR read–on–TLB–miss enabled (NORMAL mode), the IO PDIR is read to get the TLB entry. This is accomplished by adding the IO_PDIR_BASE register to the GSC+ address bits [31:12], and reading the IO PDIR entry at that location. Note that the page in memory containing the IO PDIR must be equivalently mapped since we would be unable to generate a virtual index otherwise. The IO PDIR read is issued onto Runway, and the transaction information is stored in the Pool, along with a special flag indicating that this read is to satisfy a TLB miss. Once the data for the IO PDIR read comes back, it will be written immediately to the TLB. The return data will NOT go through the OutQ, nor through the InQ, but rather sidetracked directly to the TLB. This is necessary as all transactions in the InQ are being held up due to the lack of a TLB entry for the transaction at the head of the queue. Now the transaction at the head of queue can be generated, and driven out as described above.

GSC has a prefetch hint to enable better performance (reduced latency) for DMA reads. The prefetch is requested by assertion of the XQL line on GSC+. This is a deterministic prefetch, which means that if a GSC+ guest asserts XQL, it will at some time in the near future come back and request that address. The IOA accommodates prefetching by allowing eight total outstanding transactions. This is limited by the number of transaction IDs on Runway implemented by the Tornado processor. Runway defines six bits of transaction ID, but Tornado only implements three TRANS ID bits. Note, the limit on outstanding transactions is only relevant to transactions expecting responses (i.e. reads), and does not limit the number of outstanding write transactions. Because of the limit on transaction IDs, the IOA may have eight outstanding read transactions at any one time, hence our Pool is only eight entries deep. This limits a GSC+ bus with six guest to six normal reads, and two prefetch reads. Similarly, a GSC+ bus with four guests could have four normal reads, and four prefetch reads outstanding at one time. Note these exam-

ples assume GSC+ guests because to have ANY outstanding read transactions requires that they be split reads (only possible by GSC+ guests).

Due to the GSC+ limitation of only one prefetch outstanding per guest, bus converters below GSC+ will not get performance benefits from prefetching unless only ONE DMA stream is active at a time. Two simultaneous DMA streams will cause thrashing of the single outstanding prefetch per guest limitation. See the section on prefetch for more details on obtaining maximum benefit from prefetching and the differences between deterministic and speculative prefetch.

If a GSC guest asserts XQL along with a DMA read transaction address cycle, the InQ entry created to forward the read request to Runway will have its NEXT bit set. When a read request with NEXT asserted reaches the top of the InQ, the read is issued on Runway as described above. If the transaction size is 16 bytes or 32 bytes, and the sequential bit read from the TLB allows prefetching, then the address is incremented, the prefetch read is issued and the transaction information stored in the Pool. Some additional information is stored in the Pool for a prefetch read. These bits indicate the GSC prefetch address, the GSC+ guest ID, and the state of the prefetch (prefetch issued on Runway, prefetch data returned from Runway, read corresponding to prefetch requested on GSC+, or discard prefetch after data returned). The GSC address stored in the Pool is compared against the read address, and the guest ID from the Pool is compared with guest ID to determine whether the prefetch has been formally requested (i.e. now a normal read) or if the prefetch should be discarded. NOTE: prefetching will only be performed for reads of 16 or 32 bytes with XQL asserted on pages with the TLB sequential bit set. See the section on prefetching for more details.

Note that each UTurn is allocated two master IDs, so each IOA has its own master ID to uniquely identify which IOA to return the data to. Recall, Runway defines six bits of transaction ID, but the TORNADO processor only intends to implement three of these ID bits. In order to be compatible with both TORNADO and any future processor upgrades (i.e. PCX–U), the IOA does the following with transaction IDs. When the IOA sources transactions, we only use the lower three transaction ID bits; the upper three transaction ID lines are always driven to zeros (0). When the IOA is responding to a read sourced on Runway (either a read of an IOA internal register, or read of a GSC guest) we will return all six bits of the transaction ID sourced on Runway. This allows the IOA to operate with the trans–ID–limited–TORNADO, and the full–trans–id–PCX–U.

6.2. Coherent IO

The IOA implements coherent IO. Coherent IO provides many advantages. First, it allows outbound DMA to get the most current data, possibly extracting it from a processor cache, without requiring the processor to flush all DMA data from its cache prior to starting the DMA. Similarly for inbound DMA, the IOA is able to put data into memory, forcing stale data out of the processors without requiring processors to purge their caches. These two features allow much less restrictive speculative prefetch rules for processors wanting to do speculative execution (PCX–U).

Additionally, for partial cache line writes from the IO world, the IOA is able to acquire a private copy of the cache line, modify the appropriate part of the cache line, and then write it back. This prevents the memory controller from having to implement read–modify–write. Similarly, for semaphore operations from the IO world, the IOA performs a read_priv, return data to GSC guest, modify in cache, and write back sequence to guarantee atomicity. Note this only applies to semaphores (or atomic read/write combinations) which are contained within one and only one cache line.

Note in the first case of coherent IO above, the IOA just uses coherent Runway transactions to get the DMA data to/from all necessary locations (main memory, processor caches). In the second case, the IOA

actually has a cache (albeit very small – just one location) and uses coherent Runway transactions to accomplish read–modify–write and semaphore operations within that one cache line. The point here is that coherent IO does NOT require a cache, just use of coherent transactions. The IOA does both: using coherent transactions to read/write the most current data values, and using a cache to provide atomic access of memory for read–modify–writes and semaphores.

6.3. Cache location and control

Each IOA has one cache location used to perform read–modify–writes and semaphores. When a short write to memory (all 1, 2, 3, 4, 8, and some 16 byte writes) or the read part of a read/write semaphore, or a clear transaction is issued on GSC+, the inbound side issues a read private on Runway and sets the CACHE bit in the Pool. This indicates that the returned data should be placed into the cache once received by the outbound side. Note that for reads and clears, the data is also placed into the outbound read return queue (since the GSC+ guest is expecting return data).

With the cache line of data held privately, the inbound side is then able to modify the part of cache line corresponding to the write or clear transaction issued by the GSC+ guest. This is supported by extensive byte steering logic around the cache location. This byte steering logic allows the inbound side to load any 1, 2, 3 or 4 bytes within a word, or any word, or any aligned double word, or any aligned four words in the cache line. Recall that writes are REQUIRED to be quantity aligned in the PA world.

The cache line is then returned to memory via a write back Runway transaction. Note that UTurn NEVER issues a cache to cache copy. For simplicity, UTurn will delay cache coherent check responses that HIT on the cache location until AFTER the write back, then respond with COH_OK. This keeps UTurn coherently 'legal', but allows us the simplicity of never having to do cache to cache copies. This has minimum impact on the system performance since UTurn rarely owns cache lines (privately), and once owned privately the cache line is quickly modified and returned. See the section on cache coherency in the outbound chapter for more details.

In addition to the storage for the cache location, and the steering logic for loading the cache location, there is also a small state machine for remembering the state of the cache line. The UTurn cache can be in any one of four states. The cache location starts out in the CACHE_INVALID state (i.e. nothing in cache). Once a read private is requested by the inbound side, the cache state advances to the WAIT_ON_DATA state (i.e. read issued on Runway, UTurn owns the cache line, but data has not yet been returned to place in the cache). When the data is returned on Runway, and loaded into the cache, the cache state advances to the CACHE_VALID state (i.e. UTurn owns cache line, data is valid in the cache). The inbound side is now able to modify the data (or some bytes of the cache line), and write the cache line back to memory. Once written back, the cache state returns to CACHE_INVALID.

There is one additional cache state which is used when a GSC read 'looks' like the first half of a semaphore (i.e. read, connected with LSL asserted), but the next transaction is NOT the second half of the semaphore (i.e. write with LSL asserted but to some OTHER address). In this case UTurn issues a read private, and while waiting for the data to return from memory, the GSC+ guest releases the LSL line. This indicates that this was NOT a semaphore, and the cache state advances to the INVALID_WAIT state. This is necessary since the inbound side can NOT issue a second read private until the data from the first one has been returned (this would confuse cache coherency checking and Pool entries). The INVALID_WAIT allows the data to return, then the data is discarded and UTurn no longer owns the cache line. Note this is a strange coherency case where UTurn 'owns' a cache line, then no longer 'owns' it without a write back. This is allowable since UTurn never dirtied the cache line, but this presents a problem if a directory based cache coherency scheme were ever implemented on Runway. The cache location is considered to be held private while in either the WAIT_ON_DATA or the CACHE_VALID state.

6.4. Two to One Runway Interface

The target frequency for Runway is 120 MHz, whereas the majority of logic in a synthesized standard cell IOA design is able to run at only about half this frequency. This requires that the interface between the Runway bus and the Runway side of IOA handle the 2 to 1 frequency difference.

On the inbound side of the IOA this is a relatively simple problem. The transaction from the top of the InQ is loaded into a set of temporary registers along with the address generated from the TLB, and any data associated with the transaction. This is all done at the half Runway frequency. Once all the information for the transaction is loaded into these temporary registers, the R_start_arb signal is asserted and the full frequency Runway side takes over.

The full frequency portion arbitrates for the Runway bus, and once it has won arbitration drives the transaction out onto Runway by dumping the values in the temporary registers in order. Notice that the full frequency side need only know how to arbitrate, and then just dumps the transaction onto the bus without knowing anything about the transaction except its length. This works because Runway has no mechanism for the transaction destination to pace the deliver of data. We just put it out there, and the destination MUST accept it. Once the transaction has been completed, the full frequency side asserts an interface_ready signal indicating that the register are ready for the next transaction.

6.5. TLB Operation

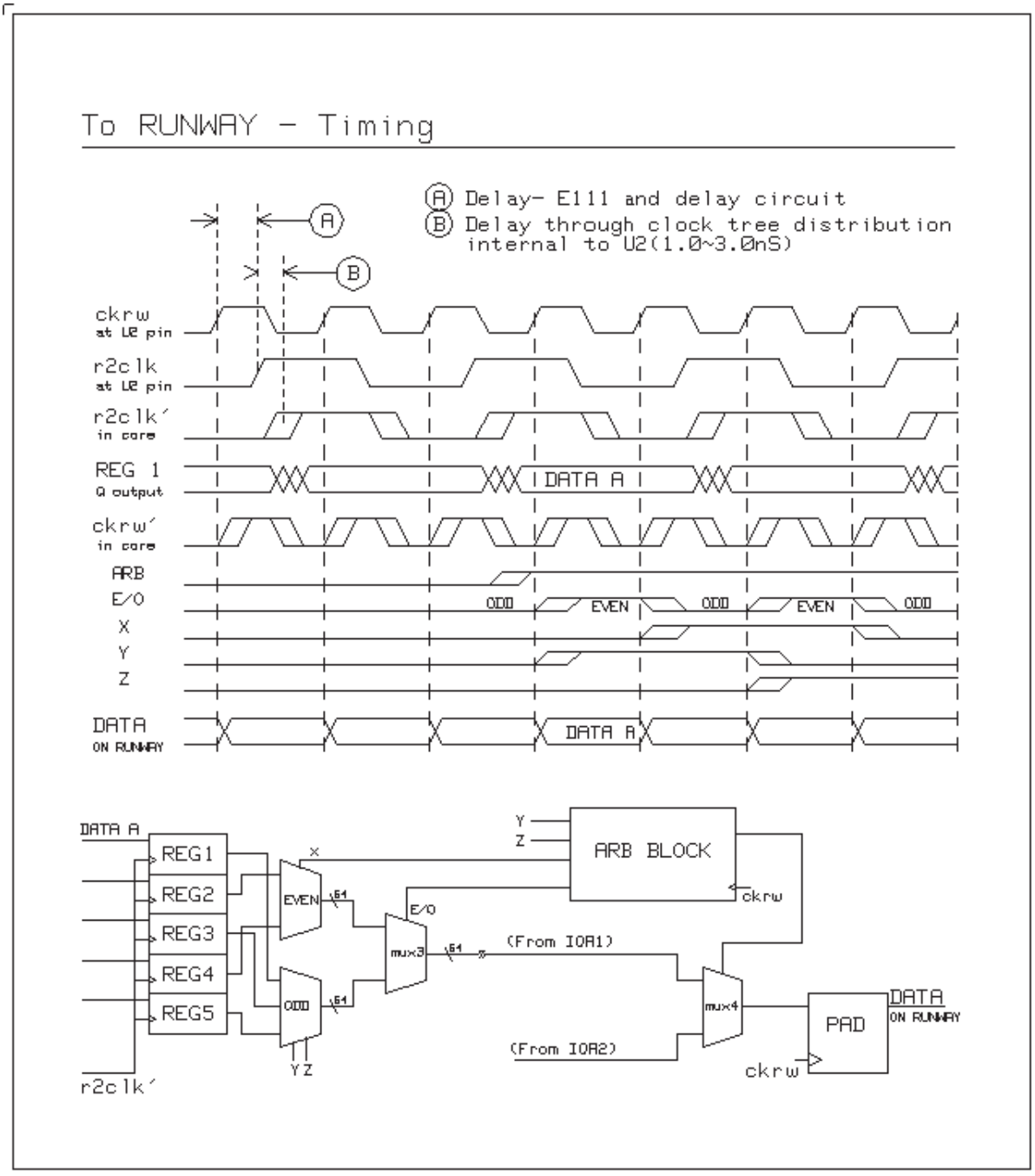
The TLB in the IOA provides three important functions. First, it provides the address translation mechanism necessary for taking a 32 bit GSC+ address and generating a 40 bit Runway address. Second, it provides the virtual index necessary for processors to snoop their caches to do coherency checking for the coherent transactions used by the IOA. Finally, it provides the page type bits used to map GSC transactions to Runway transactions (FAST DMA page, SAFE DMA page), used to enable/disable prefetching, and used to enable/disable assertion of LSL from causing assertion of STOP_MOST (STOP_MOST enabled page, or STOP_MOST disable page),

Entries in the TLB can be written in two different ways. Operating system software can preload the TLB entry to prevent the additional latency required when a TLB miss occurs. Alternately, the IOA can read from the IO PDIR when a miss occurs, and then write the missing entry into the TLB. These two mechanisms provide both the minimum latency (preloaded TLB entry) and the robust solution (IOA can service TLB misses on its own). Note that the IO PDIR table MUST be placed in an equivalently mapped region in memory since there is no TLB entry for the reads of IO PDIR on a TLB miss.

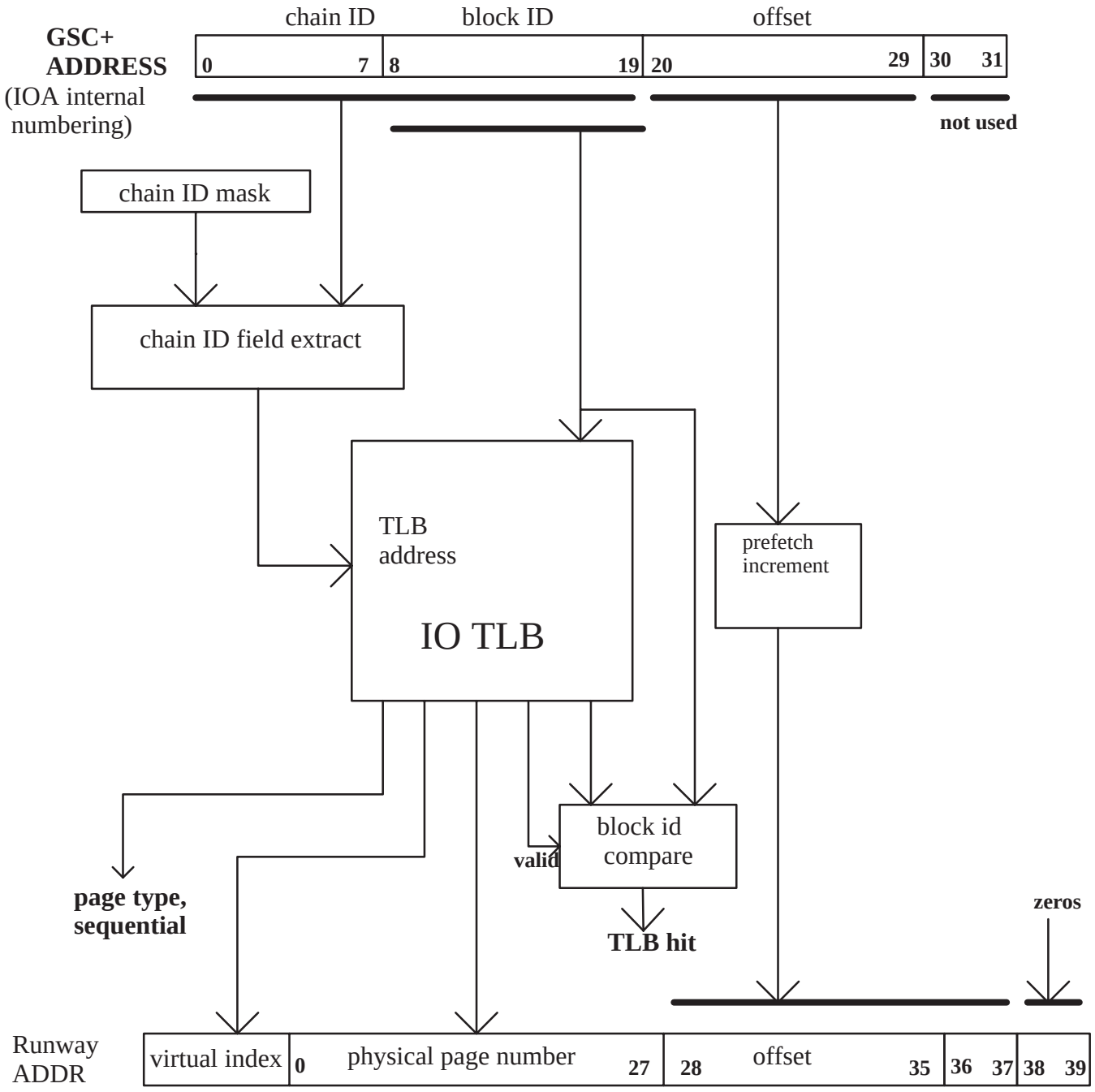
The TLB address translation method is determined by the value in the IO_CONTROL_HV_MODE register. The ability to preload TLB entries is accomplished using the IO_TLB_ENTRY_M,L registers for the data, and then invoked by sending the appropriate command to the IO_COMMAND register. See the section on Initializing the TLB and Manipulating TLB Entries (below) in this chapter.

6.5.1. TLB Address Translation Modes

Three different TLB address translation modes are supported. REAL mode (the default mode of operation) is for system initialization before the TLB is initialized and turned on. In REAL mode all coherent accesses use a virtual index that assumes equivalently mapped pages. REAL mode has some special requirements, see the section on Real mode, below. ERROR mode is available for small systems where



the TLB contains valid address translations, but there is no IO PDIR in memory (since the memory consumption of the IO PDIR could be significant in a small system). Since there is no IO PDIR, all active address translations MUST be valid in the IOA's TLB for ERROR mode. Finally, NORMAL mode where if an active address translation is not in the IOA's TLB, the entry will be fetched from the IO PDIR. NORMAL mode requires that all active address translations be present in the IO PDIR, and the IO PDIR must be memory resident in an equivalently mapped region. The address translation modes are selected


chain ID
block ID
offset
GSC+
ADDRESS
(IOA internal
numbering)
0
7
8
19
20
29
30
31
not used
chain ID mask
chain ID field extract
TLB
address
IO TLB
prefetch
increment
block id
compare
valid
page type,
sequential
TLB hit
zeros
Runway
ADDR
virtual index
0
physical page number
27
28
offset
35
36
37
38
39

through the IO_CONTROL_HV_MODE HPA register. See the section on architectural register for more information.

6.5.1.1. Translation of GSC IO space addresses

Note that regardless of which of the above address translation modes is selected, a GSC address with bits 0 through 3 set (the GSC IO address space, address Fxxx_xxxx) is 'F' extended and passed to Runway without ever accessing the TLB. This allows writes from an GSC guest to an address in the IO address space to remain in the IO address space on Runway (writes to the processor IO_EIR for example). Recall that writes to an IO address on Runway (using the write short transaction) do NOT require a virtual index since no coherency check is made for transactions to the IO address space.

Note that the only transaction allowed on GSC to an IO address space is a WRITE1 transaction. All other GSC guest sourced transactions to IO space are illegal and will result in the logging of a Runway side hard error (GSC improper access).

Because GSC IO space addresses do NOT use the TLB, some TLB entries are NOT available depending on the value of the IO_CHAIN_ID_MASK. See the section below on IO_CHAIN_ID_MASK values and usable GSC addresses.

6.5.1.2. Real Mode

In REAL mode coherent transactions are used, and addresses are '0' or 'F' extended as necessary. With 'F' extension, IO space addresses on GSC+ are mapped to IO space addresses on Runway, and non–IO addresses are mapped to the equivalently mapped addresses on Runway. The equivalently mapped virtual index necessary for cache snooping is taken directly from GSC+ address bits [19:12].

Note that in REAL mode there are no page type bits to assist in mapping GSC transactions to Runway transactions since there is no IO PDIR, nor are there entries in the TLB. The IOA defaults to the SAFE DMA with STOP_MOST enabled page type, with prefetching disabled for REAL mode. This allows the use of DMA, though with reduced performance if writes of half cache lines (4 word or 16 bytes) are used. The SAFE DMA with STOP_MOST enabled page allow the use of semaphores and EISA locked transaction sequences if requested on GSC. See the section on transaction mapping GSC+–> Runway for more details.

Note that if it is desirable to operate in REAL mode, but it is also necessary to control the page type bits, a 'fake' REAL mode can be employed. With this scheme an IO PDIR is created where ALL the entries are equivalently mapped (the same address mapping that would result if the TLB were operated in REAL mode), but now the page type and prefetch bits can be programmed. This may be desirable if STOP_MOST needs to be disabled for performance reasons, or if prefetching wants to be turned on. However, there is a downside to doing this. The IO PDIR will have to exist unless software wants to CAREFULLY manage the TLB in ERROR mode (see below). The size of the IO PDIR determines the size of the address space in physical memory which is accessible by the IO subsystem at any one time. See the section on Normal mode below for more details on the interaction between IO_PDIR size, IO_CHAIN_ID_MASK values, and IO addressable memory.

6.5.1.3. Error Mode

The second, ERROR mode is for system operation where there is no IO PDIR. This mode is provided for small systems where 256 active translations are sufficient, and the memory consumed by the IO PDIR is unnecessary. Here addresses are translated using the TLB (except IO addresses which require 'F' ex-

tension), and coherent transactions are used, but if there is a TLB miss, the IOA goes into hard_error mode. In ERROR mode, it is expected that all active address translations have been written to the TLB BEFORE any DMA using that address translation is initiated. The only mechanism for placing address translations into the TLB for ERROR mode is via the TLB direct write command. TLB entries can be removed by overwriting the entry (TLB direct write), or by purging the TLB entry (TLB purge).

6.5.1.4. Normal Mode

Finally, NORMAL mode where addresses are translated using the TLB (except IO addresses which are again 'F' extended), coherent transactions are used, and on a TLB miss the IOA reads the entry from the IO PDIR. Note that in NORMAL mode the IO PDIR MUST contain a valid, and correct TLB entry before any DMA is initiated. Otherwise a TLB miss will read the IO PDIR for the entry, place the entry into the TLB, and then attempt the transaction again. If the TLB entry just placed into the TLB does NOT have the VALID bit set, then when the TLB is accessed, the IOA will detect ANOTHER TLB miss, recognize this as a hard error, and log it. Note that in NORMAL mode, address translations can be pre–loaded into the TLB by either TLB direct writes, or with a TLB insert. Alternately, in NORMAL mode, the IOA TLB will fetch the desired address translation from the IO PDIR automatically when the first DMA transaction causes a TLB miss. TLB entries can be removed by overwriting the entry (TLB direct write, or TLB insert), or by purging the TLB entry (TLB purge). Recall that the IO PDIR must be placed in a equivalently mapped portion of resident memory. This is required as there is no TLB entry for the IO PDIR.

Note that the IO PDIR will have to be sized depending on the desired IO addressable memory. For example, if IO_CHAIN_ID_MASK = FF00_0000 (the maximum), then the IO addressable memory is a 32 bit address space (the maximum). If MASK = 0FF0_0000, then the IO addressable memory is reduced to a 28 bit address space, and the IO_PDIR size is reduced by a factor of 16. Similarly, if this IO PDIR is too large to tolerate, then MASK = 00FF_0000 could be used (just as a contrived example), then the IO PDIR size is reduced by another factor of 16, but the IO addressable memory would be reduced further to a 24 bit address space. Note that due to the ability to program the TLB, the destination in memory for DMA can be ANYWHERE in physical memory, but ONLY a region the size of the IO addressable space is 'touchable'. The IO addressable space in physical memory can be scattered anywhere in physical memory (mapped in 4K byte blocks), but the TOTAL AT ANY ONE TIME cannot exceed the IO addressable size

6.5.2. Initializing the TLB and Manipulating TLB Entries

Before the TLB can be used, the registers affecting TLB operation, and the actual TLB RAM entries must be initialized. Initialization depends on the address translation mode selected, discussed above. The details of initialization for each address translation mode are discussed below, but first lets explain the TLB manipulation commands used both to initialize the TLB (at system boot–up) and to place or delete entries in the TLB. The three TLB commands are accessed via the HPA_IO_COMMAND register discussed in the Architectural Requirements chapter. ALL TLB operations (including the following TLB commands) require that the IO_CHAIN_ID_MASK HPA register be written BEFORE any of these commands are executed (see the section below on TLB initialization for more details). The three commands are: TLB_Purge, TLB_Insert, and TLB_Direct_Write.

- TLB_Purge removes the TLB entry specified by the 20 bit page_num field in the HPA_IO_COMMAND register. The TLB entry is only purged if the TLB entry in RAM is valid and matches the tag portion of the page_num field (i.e. a full match is required for the 20 bit page_num field).
- TLB_Insert causes the TLB to fetch the TLB entry specified by the 20 bit page_num field in the HPA_IO_COMMAND register from the IO_PDIR table in memory.

Note TLB_Insert requires that the IO_PDIR table in memory is valid, and the IO_PDIR_BASE HPA register in the IOA has been initialized to point to the IO_PDIR table. Note that part of the 20 bit page_num field specified with the TLB_Insert command is used to point to the TLB RAM location, and part is used as the tag for the TLB entry, and all of the page_num field is added to the IO_PDIR_BASE register to find the TLB entry in the IO_PDIR table in memory.

- TLB _Direct_Write causes the TLB entry loaded in IO_TLB_ENTRY_M,L HPA registers to be written to the TLB location specified by the 20 bit page_num field in IO_COMMAND. The TAG portion of the TLB entry is part of the page_num field. TLB_Direct_Write is the ONLY method for initializing the TLB RAM at power on (see below). TLB_Direct_Write can be used to place entries into the TLB for NORMAL mode, and is the only method for placing entries into the TLB for ERROR mode. The sequence of transactions required to perform a TLB direct write are: 1) write the desired TLB entry into IO_TLB_ENTRY_M,L 2) write the TLB direct write command to the IO_COMMAND register along with the 20 bit page number field 3) read the Runway IO_STATUS register and check that status indicates ready (i.e. TLB write has completed). Only AFTER the read of IO_STATUS indicates 'ready' may IO_TLB_ENTRY_M,L registers be re–loaded. If they are re–loaded too early the TLB entry written may be the old value, the new value, or some mix of the two. Obviously this should be avoided.

6.5.2.1. Initializing the TLB for NORMAL mode

Minimum initialization for NORMAL mode requires that software read the size of the IO TLB, then write to the IO_CHAIN_ID_MASK register, then write the IO_PDIR_BASE register, and then write the IO_CONTROL_HV_MODE register selecting NORMAL address translation. The TLB_ENTRY_M,L registers should then be written with a tlb entry that has the VALID bit NOT set (writing all zeros to TLB_ENTRY_M,L is a nice simple way to do this). A sequence of TLB direct writes should then be done to ALL 256 of the TLB entries to initialize the TLB RAM. Failure to initialize the TLB RAM will cause false (random) TLB entries to be read from the TLB. These random entries will probably not match with actual transactions; BUT THEY MIGHT, and they would write and read from random areas in memory. The safe initialization described above resets all valid bits, preventing this small probability problem from ever occurring.

IO TLB size indicates the size of the TLB in the IOA, and is emulated by IODC. The IO_CHAIN_ID_MASK determines where the chain ID field is extracted from the GSC+ address (and indirectly the size of the IO PDIR in main memory). The IO_PDIR_BASE register is the pointer to the base of the IO PDIR in main memory, and is used to fetch a TLB entry when a miss occurs. The IO PDIR must be memory resident in an equivalently mapped region of memory.

For NORMAL mode there are several ways the valid TLB entry can be loaded into the TLB, though NORMAL mode requires that the address translation be present in the IO PDIR regardless of how it first gets into the IOA TLB. First, wait until a GSC transaction causes a TLB miss, the IOA will automatically fetch the TLB entry from the IO PDIR table (in an equivalently mapped area of main memory). Second, cause the entry to be preloaded by doing a TLB_Insert, which will cause the IOA to go fetch the entry from the IO PDIR table in main memory. This is similar to fetch–on–miss described above, except that the insert can be done in parallel with the setup of the DMA. This prevents the miss from adding additional latency to the DMA transfer, since the TLB entry is read from the IO PDIR table ahead of the first GSC DMA transaction. Finally, the TLB entry can be loaded using TLB_Direct_Write command with

IO_TLB_ENTRY_M,L registers. This is the same method used for putting entries into the TLB for ERROR mode. Note the sequence of transactions required for proper operation of a TLB direct write described above.

Recall that in NORMAL mode when a TLB miss occurs the TLB entry will be read from the IO PDIR table in main memory. If a valid TLB entry is NOT in the IO PDIR when the TLB miss read is done, the IOA will detect this as an error. To avoid this error, there must be a valid entry in the IO PDIR table BEFORE any IO transaction expecting to use that entry is seen by the IOA. This TLB fault generates and logs a HARD error, which prevents further DMA activity until cleared with a command RESET. Avoid this, it's bad.

6.5.2.2. Initializing the TLB for ERROR mode

Alternately, the TLB can be used in ERROR mode, where initialization consists of: read the size of the IO TLB, write the IO_CHAIN_ID_MASK register, write IO_CONTROL_HV_MODE to select ERROR translation mode, and initialize all TLB RAM locations (described below), then write the desired entry (or entries) into the TLB by writing to IO_TLB_ENTRY_M,L, and force the entry to be written by writing a TLB_Direct_Write command and page_num indicating the TLB RAM location and tag to the IO_COMMAND register. Finally read the Runway IO_STATUS register and check that status indicates READY before attempting to re–load IO_TLB_ENTRY_M,L (otherwise the values in IO_TLB_ENTRY_M,L might change BEFORE the TLB direct write actually happens — NOT GOOD).

ERROR mode also requires that all TLB RAM entries be initialized before use. The minimum TLB RAM initialization requires that only the TLB RAM locations to be used be initialized. However, to avoid the possibility of accidentally using a random TLB RAM value, it is safer to initialize the entire TLB RAM with non–valid entries. This is accomplished by writing an initialization entry with the valid bit NOT set to IO_TLB_ENTRY_M,L (all zeros to IO_TLB_ENTRY_M,L works fine), and writing this initialization entry to ALL 256 TLB RAM locations using TLB direct writes. Once this is done, the actual TLB entries to be used can be written using the TLB_Direct_Write command described above.

Recall that in ERROR mode there should NEVER be a TLB miss. Software MUST guarantee that there is a valid address translation in the TLB for every active DMA BEFORE that DMA begins (otherwise an error is logged, hence the name). Another side effect of ERROR mode is that two active DMAs cannot use the same TLB RAM location. Software MUST guarantee that all concurrently active DMAs use GSC addresses that do not collide in the TLB RAM.

6.5.2.3. TLB initializations MUSTS

For proper TLB operation, the TLB MUST be initialized. The initialization sequence is described below. The initialization should be done in the sequence specified below.

- Read the TLB size by reading IO_DC_DATA. The UTurn IO TLB size will be contained in the shift field of the iodc_spa byte of IO_DC_DATA. For UTurn, with a TLB size of 256, the shift field will be equal to 8 (2^8 = 256). Note that the actual 8 bytes of io_dc_data on UTurn does not contain the IO TLB size. PDC will add this information in the PDC_IODC call before the io_dc_data is returned to the O.S. This requires that the OS always access UTurn's IO_DC_DATA via the PDC_IODC "get entry point" option.
- Write the IO_CHAIN_ID_MASK registers with the consecutive number of bits set equal to log2(TLB size). For UTurn, the TLB size is 256, so the IO_CHAIN_ID_MASK value should have AT MOST eight bits set. These eight bits

indicate where to extract the TLB address from. These eight bits MUST be consecutive in the register. More than eight bits set will cause unpredictable collisions in the UTurn TLB. Fewer than eight bits set will reduce the usable size of the IOA TLB (i.e. seven bits set would make the TLB size decrease from 256 entries to 128 entries). If there are any LEADING ZEROs in the IO_CHAIN_ID_MASK register, then software MUST guarantee that IO virtual address (GSC+ DMA addresses), and PAGE_NUMs used for TLB direct write, purge and insert commands will NEVER have any of these bits set (with the exception of GSC addresses in the IO address space). Recall that IO addresses (address = Fxxx_xxxx to FFFF_FFFF) do NOT use the TLB, but are 'F' extended. Restated, if IO_CHAIN_ID_MASK value has leading zeros (i.e. 0FF0_0000), then GSC+ DMA addresses must not have any address bits set in the leading zero area (i.e. if MASK = 0FF0_0000, illegal GSC addresses are: 1xxx_xxxx thru Exxx_xxxx, where Fxxx_xxxx is OK because it is in the IO address space and will not use the TLB). Similarly, if CHAIN_ID_MASK = 0FF0_0000, then illegal PAGE_NUM values for TLB purge, insert and direct write are: 1xxx_xxxx thru Fxxx_xxxx. Failure to guarantee this may result in FALSE hits in the UTurn TLB, giving random address translation and the potential for data corruption in memory, or false TLB access for TLB commands (purge, insert, direct write). See the section below on IO_CHAIN_ID_MASK Values and Usable GSC addresses for more detail. The IO_CHAIN_ID_MASK register should NOT be re–written while any IO activity is in progress. In general the IO_CHAIN_ID_MASK register should only be written once on system boot–up initialization. Changing the IO_CHAIN_ID_MASK register scrambles the location of entries in the TLB RAM, and requires complete re–initialization of the IO TLB in UTurn.

- All TLB RAM entries should be initialized to a non–valid entry (for both NORMAL and ERROR mode). This is done by writing a non–valid entry into TLB_ENTRY_L,M (a value of all zeros works fine). This non–valid TLB entry should then be written to all TLB RAM locations by doing 256 TLB direct writes, one to each of the TLB RAM locations. The TLB direct write requires a 20 bit PAGE_NUM value to determine where to write the entry. For initialization, only 256 direct writes need to be done, but the PAGE_NUM values are dependent on the IO_CHAIN_ID_MASK value. Thus if CHAIN_ID_MASK = FF00_0000, then initialization direct writes should be done to PAGE_NUMs: 0000_0xxx, 0100_0xxx, 0200_0xxx, 0300_0xxx, ... FF00_0xxx. Similarly, if CHAIN_ID_MASK = 3FC0_0000, then initialization direct writes should be done to PAGE_NUMs: 0040_0xxx, 0080_0xxx, 00C0_0xxx, 0100_0xxx, 0140_0xxx, 01C0_0xxx, ... 3FC0_0xxx. Notice that the count from 0 to 255 occurs in the SAME bit positions as the CHAIN_ID_MASK value. TLB RAM initialization could also be done with TLB purges, however since TLB tag hit is checked before the purge is done, to initialize all TLB RAM locations 2^20 TLB purges would have to issued in order to 'hit' all possible values for the tag to guarantee that all RAM locations are initialized. This is WAY TOO MANY to be practical (1,048,576 TLB purges), although it does work. TLB RAM initialization prevents random (power on) values from being accidentally accessed by DMA traffic. Although unlikely that an IO virtual page number would match a random power–on value in the TLB, it could happen. TLB initialization prevents this from occurring, and only needs to be done on power up, or if the IO_CHAIN_ID_MASK value is changed.

- If the TLB is to be used in NORMAL mode, the IO_PDIR_BASE register should be written with the address of the IO_PDIR. See the section below on how the IO_PDIR_BASE is used along with the address of the TLB miss to fetch a TLB entry from the IO PDIR. The IO_PDIR_BASE register (used for NORMAL mode and TLB insert command) should NOT be re–written while any IO activity is in progress. In general the IO_PDIR_BASE register should only be written once when the IO PDIR in main memory is built. Changing the base register causes TLB entries to be fetched from the 'new' location of the IO PDIR (O.K. if a 'new' IO PDIR is set up BEFORE changing the base register).
- For NORMAL mode (where TLB entries are automatically fetched from the IO PDIR), the entry in the IO PDIR MUST be valid and correct BEFORE a DMA is initiated that uses that entry. Specifically, the IO PDIR entry must be correct and have the VALID bit set. Failure to have the valid bit set in the IO_PDIR will result in a hard error. The virtual index, physical page number, and page types should also be correct prior to initiating a DMA. This is not a detectable error, but an incorrect virtual index or physical page number will result in memory corruption or cache incoherence. Additionally, the IO PDIR table must be resident in an equivalently mapped region of memory.
- For ERROR mode (where TLB entries are directly written into the UTurn TLB), the TLB entry in UTurn MUST valid and correspond to the DMA, BEFORE the DMA is initiated. Similarly, no two active DMAs can use the same TLB RAM entry. Failure to do either of these will result in an TLB miss, and a hard error.
- All TLB RAM entries should be initialized first to a non–valid entry (for both NORMAL and ERROR mode) and then to the actual desired TLB entry (for ERROR mode) before starting any IO activity.

6.5.2.4. Timing of TLB commands and control register writes with respect to DMA activity

The timing of commands affecting TLB entries (TLB_Purge, TLB_Insert, TLB_Direct_Write), and writes to HPA registers affecting TLB operation (IO_CONTROL_HV_MODE, IO_CHAIN_ID_MASK, IO_PDIR_BASE, IO_TLB_ENTRY_M,L) must be guaranteed with respect to DMA activity. This requires that operating system software FULLY initialize the TLB BEFORE any DMA activity on GSC. Similarly,
operating system software must guarantee for ERROR mode that the TLB_Direct_Write arrive BEFORE any DMA activity using that entry. Refer to the timing information below.

- HPA register writes happen IMMEDIATELY. Thus writes to IO_PDIR_BASE, IO_CHAIN_ID_MASK, IO_CONTROL_HV_MODE have immediate side effects. In general there should be NO DMA activity if any of these registers are being altered.
- TLB commands are first passed through the outbound queue, and then through the inbound queue. Thus TLB_Purge, TLB_Insert, and TLB_Direct_Write may take many cycles before causing the change in the TLB entry. The delay depends on the number of transactions in both queues. In general, the TLB manipulation commands should be issued on Runway BEFORE any write on Runway which initiates DMA activity.
- A side effect of the above two bullets (HPA writes happen immediately, and TLB commands must go through both IOA internal queues) is that the following sequence

MUST be adhered for correct operation of TLB direct writes: 1) write TLB entry to IO_TLB_ENTRY_M,L 2) send a TLB direct write 3) read Runway side (UBC) IO_STATUS and check that the ready bit is set. THEN and ONLY THEN can the IO_TLB_ENTRY_M,L registers be re–written. If you don't wait for the data return from the read of IO_STATUS before changing IO_TLB_ENTRY_M,L, you will get the WRONG (or even a mixed) TLB entry written to the TLB.

6.5.2.5. Effects of TLB initialization

The initialization sequence described above has the following effects. When software reads the TLB size it know how many bits must be extracted from the IO address to be used as the address into the TLB. Software allocates space for the IO PDIR table in an equivalently mapped area in memory. The size of the IO PDIR table is equal to: size of a coherent IO TLB entry * 2^(CHAIN_ID + BLOCK_ID). Where the size of the coherent IO entry is 64 bits long, and the size of the CHAIN_ID field is determined by the number of bits necessary to access the TLB (256 entries in the TLB RAM, requiring 8 bits of CHAIN_ID address). This leaves the size of the BLOCK_ID field adjustable. The tradeoff in determining the size of the BLOCK_ID field is: size of the IO PDIR table in main memory (always present in memory, equivalently mapped) vs the size of the IO address space available for DMA. As the BLOCK_ID field gets smaller, the IO PDIR table in main memory also gets smaller, as does the IO address space that is usable on GSC+. Note that the CHAIN_ID field remains the same size, but gets extracted from a different part of the IO address so that the TLB always receives the same number of bits as address. This is the function of the CHAIN ID FIELD EXTRACT block shown in the block diagram of the TLB.

6.5.2.6. IO_CHAIN_ID_MASK Values and Usable GSC addresses

Recall that the IO_CHAIN_ID_MASK register is used to extract a portion of the GSC address or a portion of the TLB command PAGE_NUM field to access the TLB. A side effect of this functionality is that some GSC address are NOT usable depending on the value in the IO_CHAIN_ID_MASK register. Similarly, some PAGE_NUM values are NOT legal depending on the value of CHAIN_ID_MASK. This is essentially an aliasing problem due to bits being ignored depending on the CHAIN_ID_MASK value. The rules are outlined in the section above called TLB initialization MUSTS. The side effects are explained here.

For example if IO_CHAIN_ID_MASK = FF00_0000, then GSC address 00xx_xxxx uses TLB entry 00, and GSC address 01xx_xxxx uses TLB entry 01, but GSC address F0xx_xxxx which would use TLB entry F0 EXCEPT that it is in the IO address space, and thus does NOT use the TLB. All GSC address from F0xx_xxxx through FFxx_xxxx are in the IO address space, hence do NOT use the TLB, and thus for this example TLB entries F0 through FF are unavailable. So for IO_CHAIN_ID_MASK = FF00_0000, sixteen TLB entries of the total 256 are NOT usable, but all GSC address are legal, and all PAGE_NUM values are legal. However, recall that GSC addresses F0xx_xxxx through FFFF_FFFF are in the IO address space, and are NOT usable for DMA.

USABLE GSC address for example IO_CHAIN_ID_MASK = FF00_0000

- GSC addresses 00xx_xxxx uses TLB entry 00 and are legal
- GSC addresses EFxx_xxxx uses TLB entry EF and are legal
- GSC addresses F0xx_xxxx through FFFF_FFFF are in the IO address space, and do NOT use a TLB entry, but are legal addresses

- TLB entries F0 through FF are NOT available in this example (16 out of 256 TLB entries are lost due to collisions with the IO address space)
- PAGE_NUM values 0000_0xxx thru FFFF_Fxxx (all values) are legal for TLB commands (TLB purge, insert, direct write).

Let's try another example to drive this point home. If, for example, IO_CHAIN_ID_MASK = 7F80_0000, then GSC address 008x_xxxx uses TLB entry 80, and GSC address 7F8x_xxxx uses TLB entry FF (see the section below on TLB RAM address generation if you want details on how the CHAIN_ID_MASK is used to generate the TLB RAM address). GSC addresses 8xxx_xxxx through EFFF_FFFF violate the "no bits set left of IO_CHAIN_ID_MASK value" rule, and are NOT usable. GSC addresses F0xx_xxxx though FFxx_xxxx are in the IO address space, do NOT use TLB entries, but are legal. Similarly, PAGE_NUM values of 0000_0xxx thru 7FFF_Fxxx are legal.

USABLE GSC addresses for example IO_CHAIN_ID_MASK = 7F80_0000

- GSC addresses 000x_xxxx through 007x_xxx uses TLB entry 00 and are legal
- GSC addresses 7F8x_xxxx uses TLB entry FF and are legal
- GSC addresses 8xxx_xxxx though EFFF_FFFF violate the aforementioned rule (no bits set left of most significant bit set in the CHAIN_ID_MASK) and are NOT legal for this example
- GSC addresses F0xx_xxxx through FFFF_FFFF are in the IO address space, and do NOT use a TLB entry, but are legal addresses
- PAGE_NUM values 0000_0xxx thru 7FFF_Fxxx are legal for TLB commands (TLB purge, insert, direct write)

6.5.3. TLB RAM address generation

This section explains how the TLB RAM address is generated. Specifically, how the GSC virtual page number (or TLB command PAGE_NUM) is used along with the IO_CHAIN_ID_MASK field to generate an eight bit TLB RAM address. This section is really only relevant to those wanting to understand the source of the IO_CHAIN_ID_MASK rules.

6.5.3.1. IO_CHAIN_ID_MASK rules

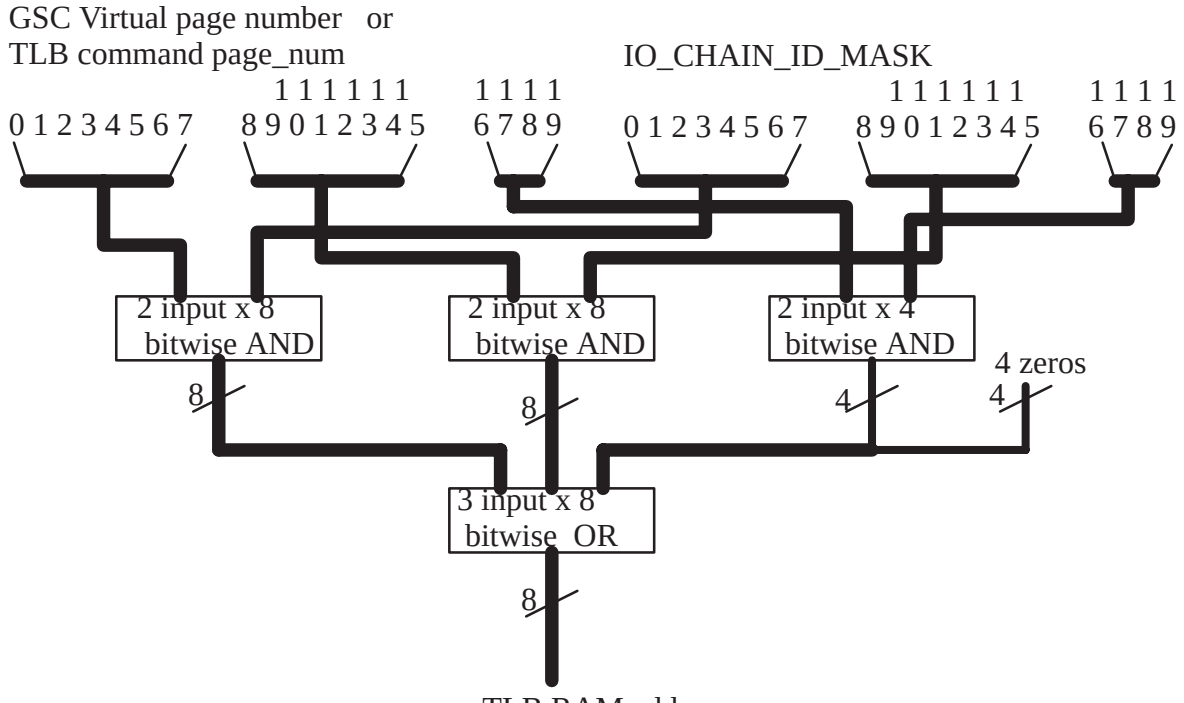
The IO_CHAIN_ID_MASK register determines which GSC virtual page number bits are used as address bits into the TLB RAM, it also indirectly determines the size of the IO_PDIR table in memory. The following is a list of rules that MUST BE ADHERED to for proper operation of coherent IO in UTurn.

- IO_CHAIN_ID_MASK should be the FIRST register written in the TLB initialization sequence (ERROR and NORMAL modes only, REAL mode does NOT require this). All TLB commands (purge, insert, direct write), and any accesses of the TLB RAM require that the mask register have been written first. This is not required for REAL mode since the TLB is essentially OFF in REAL mode.
- The value in the IO_CHAIN_ID_MASK register should have EIGHT CONSECUTIVE bits set. More than eight bits set will cause undetected aliasing of TLB entries (and hence random destinations for DMA) — this is VERY BAD, don't do it. Non–consecutive bits set in IO_CHAIN_ID_MASK cause similar problems— don't do it. Fewer than eight bits set cause the effective TLB size in UTurn to be reduced. The

TLB will operate properly, but with increased TLB misses (reduced performance) — probably something which you don't want to do. Seven bits set reduce the effective TLB size to 128 entries, six bits set reduce it to just 64 entries, and so on.

- Once the IO_CHAIN_ID_MASK register has a value written to it, software must GUARANTEE that all GSC addresses and all TLB command PAGE_NUM values have NO BITS SET LEFT OF (more significant than) the highest bit set in the mask register. Having bits set more significant than the mask value will cause aliasing of TLB entries for both TLB commands and GSC address translations.
- Once a value has been written to the mask register, a new value (i.e. different) can only be written if there is NO DMA activity, all outstanding TLB commands have completed, and the TLB RAM is re–initialized after the new value is written. Be cautious when re–writing the mask register.

6.5.3.2. TLB RAM address generation

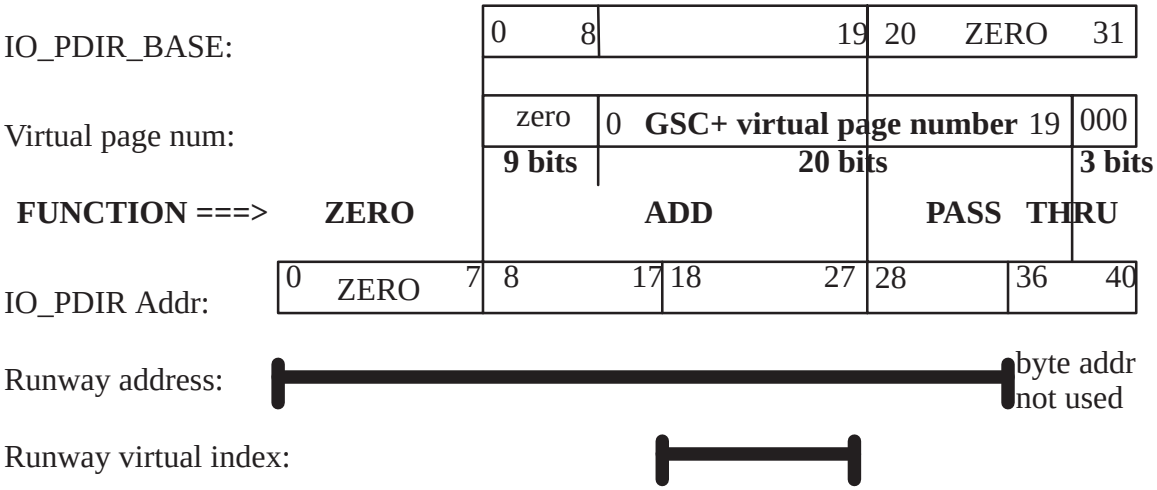


The TLB RAM address is generated by a field extraction of bits from the GSC virtual page number or TLB command PAGE_NUM as specified by the value in the IO_CHAIN_ID_MASK register. A key point is that this is NOT a full field extractor, but rather a simple eight bit only field extract. This allows the hardware field extractor to be much simpler (and hence faster), but with the side effects producing the IO_CHAIN_ID_MASK rules listed in the section above.

An important observation is that the field extractor may perform a hashing function as long as it always produces the same TLB RAM address for the same virtual page number AND same TLB command PAGE_NUM. The field extractor consists of two basic components: three of an eight bit wide AND array

and one of an eight bit wide OR array. With only eight bits set in the mask register, only eight of the twenty AND gates has mask bits set. The eight bits set in the mask register can have arbitrary placement, but they must be CONSECUTIVE in the register. This provides the guarantee that any bit disabled from one AND array is replaced by another bit enabled from one of the other AND arrays. Thus one and only one of the inputs to the OR array can ever active at one time. This performs a field extract with the side effect that as the value in the mask register is shift right, the newly added address bits fill in on most significant side.

6.5.4. Accessing the IO_PDIR on a TLB miss



If the TLB is operating in NORMAL mode, when a TLB miss occurs, the inbound side will fetch the 'missing' TLB entry automatically from the IO_PDIR in main memory. Recall that since there is no TLB entry for accessing the IO_PDIR, the IO_PDIR MUST be placed in an equivalently mapped region of memory, and the IO_PDIR must ALWAYS be memory resident. This allows UTurn to still be cache coherent on IO_PDIR fetches, and guarantees UTurn's ability to access it (UTurn has NO way to cause a page fault to bring a page in from swap).

Assuming the IO TLB is operating in NORMAL mode, once a TLB miss occurs, it is serviced as follows. First the IO_PDIR_BASE is added to the GSC+ virtual page number, and this result is zero extended producing the address in the IO_PDIR that contains FOUR TLB entries (only one of which we want). Ten bits of address are extracted to be used as the (equivalently mapped) virtual index. Note that although a cache coherent transaction is used to read the IO_PDIR (potentially reading the most current value for a PDIR entry from a processor cache), if the IO_PDIR entry changes, UTurn's TLB must be PURGED or a TLB_INSERT issued to cause the TLB to be properly updated. Once the IO_PDIR read data is returned, the desired TLB entry (one of four returned from the IO_PDIR read) is written to the TLB, and the GSC transaction which caused the miss continues.

6.5.5. TLB entry format

TLB entries can be loaded into the UTurn TLB from two different sources. One possible source is the IO_TLB_ENTRY_M,L HPA register (see the Architectural Requirements chapter). The other possible source is from memory (i.e. the IO PDIR table). The format of the TLB entry is identical for these two sources since the two 32 bit HPA registers concatenated together can be considered equivalent to a 64 bit entry read from memory.

Note that the TLB entry format has room for expansion. The format allows for a 40 bit physical page number, whereas the current Runway implementation only requires 28 bits of physical page number. Similarly, the format allows for a 12 bit virtual index, whereas Runway only requires 8 bits. The TLB entry format shown below indicates which bits are UNUSED as implemented by UTurn on Runway. The entry format below also shows which bits are used on Runway and which bits are used internal to the IOA.

6.5.5.1. First half of TLB entry (IO_TLB_ENTRY_M or word 0 from IO PDIR)

0 3	4 5	6 15	16 23	24 31
physical page number [0:3]	virtual index [0:1]	virtual index [2:11]	physical page number [4:11]	physical page number [12:19]
		Runway virtual index [0:9]		Runway real addr [0:7]
4 bits	2 bits	10 bits	8 bits	8 bits
NOT USED	NOT USED		NOT USED	

6.5.5.2. Second half of TLB entry (IO_TLB_ENTRY_L or word 1 from IO PDIR)

0 19	20 24	25	26	27 28	29 30	31
physical page number [20:39]	unused	prefetch enable	update enable	unused	coherent IO page type[0:1]	valid
Runway real addr [8:27]		IOA internal			IOA internal	IOA internal
20 bits	5 bits	1 bit	1 bit	2 bits	2 bits	1 bit
	NOT USED		NOT USED	NOT USED		

WHERE: PREFETCH ENABLE: 0 = no prefetching allowed
1 = prefetching enabled
COHERENT IO PAGE TYPE: 00 = FAST DMA page, STOP_MOST disabled
01 = SAFE DMA page, STOP_MOST disabled
10 = FAST DMA page, STOP_MOST enabled
11 = SAFE DMA page, STOP_MOST enabled
VALID: 0 = non valid TLB entry
1 = valid TLB entry

6.5.6. Accessing the TLB

Once a transaction has reached the top of the InQ, the GSC+ address must be converted into a Runway address. Assuming the IO TLB has been initialized (as described above), the GSC address is used to access the TLB (see the diagram on the previous page). There are three parts to the GSC address: the offset, the block ID, and the chain ID. The size of the offset field is fixed (by the architecture at 12 bits – 4K bytes/page). The size of the chain ID field is determined by the size of the TLB (and the value of IO_CHAIN_ID_MASK), since the chain ID is used as the address into the TLB. Note that the location of the chain ID field can vary, but its size is fixed. IO_CHAIN_ID_MASK actually determines effective

size of the TLB, but it CANNOT be bigger than 256 entries, and it really makes no sense to make it smaller than 256. Recall that there are 256 TLB entries, so chain ID is 8 bits wide, which also explains why the IO_CHAIN_ID_MASK register should have 8 bits set (no more, no less). Finally, the block ID field is whatever remains of the address BETWEEN the offset and chain ID, and is used as the tag to determine if the IO address hit or miss the TLB.

Note that the 32 bit GSC+ address is a byte address. Memory reads via Runway only need a quad word address (the upper 36 bits of a 40 bit Runway byte address). Hence bits GSC+ address [3:0] are not passed on to Runway; and Runway, a 40 bit address space, really only gets 36 bits of quad word address due to the cache line quantization of memory accesses.

The IO offset portion of the GSC address passes straight through the address translation unaltered. The chain ID field is extracted from the GSC address. The specific bits pulled from the GSC address for the chain ID vary depending on the value written to the IO_CHAIN_ID_MASK register. This allows software to trade off IO PDIR size against usable GSC address space. Larger GSC address spaces require larger IO PDIR tables in main memory, and hence consume more main memory. The block ID field only needs the remaining portion of the GSC address after offset and chain ID have been removed; however from a practical standpoint the block ID field can be fixed. When the chain mask is FF00_0000, the most significant eight bits are the chain ID, and the next twelve bits are the block ID (this produces the largest possible block ID field). If the chain mask was 0FF0_0000, the most significant four bits are UNUSED, the next eight are now the chain ID field, and the last eight are the block ID. Notice that although the block ID field got smaller (12 bits to 8 bits), its right side location did not move. For simplicity the block ID is ALWAYS extracted as twelve bits from GSC address bits [8:19] (where this is the IOA internal numbering for the GSC address. i.e. the GSC most significant address bit is GSC addr[0], complying with the PA convention). Depending on the value of the chain mask, twelve bits for block ID may be excessive, but twelve is the largest size for the field (when the chain mask is FF00_0000), and twelve bits must be allocated for the block ID field. When the chain mask is moved right (i.e. 0FF0_0000), redundant bits are stored in the block ID field, but this causes no problems.

TLB accesses proceed as follows: the chain ID is used to address the TLB, and a TLB entry is read out of the TLB RAM. The TLB entry contains the following fields: virtual index, physical page number, prefetch enable, coherent IO page type, tag, and a valid bit. The tag from the TLB is compared to the block ID field from the GSC+ address to determine if we got an actual TLB hit. Note that the valid bit in the TLB RAM is also used to determine a TLB hit. If the valid bit is not set, then there can be NO TLB hit.

Assuming we got a TLB hit, the physical page number read from the TLB entry is concatenated with the GSC+ address offset to form the physical address for accessing memory. The virtual index is provided on Runway so processors can snoop their caches to maintain cache coherency. The page type bits are used to assist in mapping from GSC+ transactions to Runway transactions, and in assertion of STOP_MOST on Runway. The prefetch bit is used to enable or disable prefetching. Recall that prefetching is only supported for GSC+ devices that can assert XQL on GSC+.

6.5.7. Definition of the PAGE TYPE bits

First, some general understanding of what the page type bits mean. The page type field is used as a HINT to the IOA to assist in mapping transactions from GSC to Runway. The page type field is currently defined to have two bits, providing four different page types. The IOA's implementation uses four of these page type encodings. The four pages currently defined are: FAST DMA with STOP_MOST enabled, SAFE DMA STOP_MOST enabled, FAST DMA STOP_MOST disabled, and SAFE DMA STOP_MOST disabled.

Note that semaphores (where atomicity requirements are limited to ONE and ONLY ONE cache line) will be supported regardless of the page type read from the TLB. Recall that atomicity limited to one cache line is provided by coherent IO, and does not require the assertion of STOP_MOST. If semaphores can be guaranteed to fall totally within one cache line, and the semaphore operation requires at MOST a read followed by a write, then STOP_MOST need not be asserted. On pages where STOP_MOST is disabled, the atomicity is limited to one cache line. This is sufficient for ALL semaphore accesses used by HP–UX and MPE–XL.

EISA locked transactions (where atomicity requirements span more than one cache line) are ONLY supported on pages where STOP_MOST is enabled. The assertion of the LSL line on GSC requests atomicity of some kind. The IOA will process the GSC guest's transactions with STOP_MOST asserted (preventing the other IOAs and the processors from accessing memory) if and only if the page type enables the assertion of STOP_MOST.

Similarly for semaphores, a GSC READ (connected) with LSL asserted will be interpreted as a semaphore regardless of the page type (a READ_PRIV is always requested on Runway, but STOP_MOST is only asserted if the page type has STOP_MOST assertion enabled). If the subsequent transaction is a GSC WRITE with LSL asserted to the same address, the IOA will perform an atomic READ/WRITE (i.e. semaphore) regardless of the page type. If the subsequent transaction is NOT a GSC write with LSL asserted, the READ_PRIV data in cache is discarded. For a GSC+ CLEAR transaction, there is no semaphore ambiguity, and an NIO style of CLEAR (atomic read cache line, write zero to first word) type semaphore will be executed, again regardless of the page type.

The following describes each of the aforementioned pages. Refer also to the table below to see how the different page types interact with the GSC+ XQL and LSL line to determine how GSC+ transactions map to Runway transactions.

- SAFE DMA, STOP_MOST enabled page (DEFAULT page type for REAL mode). This page type is for DMA access where the beginning/end of the DMA may share a cache line with some other unrelated data, and atomicity requirements span a multiple cache lines. Partial cache line writes must properly preserve dirtied cache data from the processor cache. This is accomplished using a coherent read to acquire the cache line, modifying it as necessary, and writing it back once done. This is the safe (i.e. low performance) DMA mode necessary if the DMA overlaps in a cache line with some unrelated, but modified data in the processor cache. The differences for SAFE pages are only observed for half cache line write transactions (GSC four word writes). The SAFE DMA with STOP_MOST enabled page is the default page type when the IOA is operating in REAL addressing mode. Recall in real addressing mode, all pages are equivalently mapped, and the TLB is NOT used for address translation. This means that there are NO page type bits to read out of the TLB. Hence the default page type. STOP_MOST enabled allows assertion of STOP_MOST when LSL is asserted. This page should be used only where DMA writes overlap with other data in a cache line, and atomicity requirements span multiple cache lines. (example EISA inbound partial page DMA with the potential for processor writes in the same cache line as the beginning/end of DMA).
- FAST DMA , STOP_MOST enabled page. This page type is for DMA access where the beginning/end of the DMA is aligned with a cache line or the other half of the data in the cache line does NOT need to be preserved, but atomicity requirements span a single cache line. Half cache line writes can be accomplished using the

WRITE16_PURGE transactions, which writes half the cache line, and will destroy any data on the remaining half of the cache line if it is held private dirty in a processor's cache. This is the high performance DMA mode, but requires either alignment of the DMA to cache lines, or knowledge that the remaining portion of the cache line will NOT see processor writes (i.e. is NOT held private dirty in a processor cache). The FAST DMA with STOP_MOST enabled allows assertion of STOP_MOST when LSL is asserted. This page should only be used where full cache lines writes are used (or you don't care about the lost processor data on the other half of a cache line) and atomicity requirements span multiple cache lines. (example EISA full page DMA or cache–line aligned DMA).

- SAFE DMA, STOP_MOST disabled page. This page is identical to the above SAFE page EXCEPT that assertion of LSL on GSC does NOT cause assertion of STOP_MOST on Runway. This still allows atomic accesses, but only if limited to a single cache line (i.e. semaphore). (example NIO partial page DMA). See the section below on Rules for Obtaining the Best DMA Performance, and the section on Semaphore Specification on GSC and GSC+ for more information.
- FAST DMA, STOP_MOST disabled page. This page is identical to the above FAST page EXCEPT that assertion of LSL on GSC does NOT cause assertion of STOP_MOST on Runway. This still allows atomic accesses, but only if limited to a single cache line (i.e. semaphore) (example NIO full page DMA). See the section below on Rules for Obtaining the Best DMA Performance, and the section on Semaphore Specification on GSC and GSC+ for more information.

6.5.7.1. Rules for Obtaining the Best DMA performance

- SEMAPHORES: If possible, semaphores should be specified with the GSC+ semaphore transaction (CLEAR 16), with no assertion of LSL; however, this transaction is only available to GSC+ guests (not to regular old GSC guests). The best choice for GSC guests is a GSC semaphore (read/write combination with LSL asserted) to a STOP_MOST disabled page (either FAST or SAFE). The final (and least desirable) semaphore is a GSC read/write combination with LSL asserted to a STOP_MOST enabled page (either FAST or SAFE) . Recall that assertion of LSL on a STOP_MOST enabled page causes assertion of STOP_MOST on Runway (locking all processors and the other IOAs out of memory); whereas the GSC+ CLEAR transaction and the GSC read/write combo to a STOP_MOST disabled page, are accomplished by acquiring the cache line privately, modifying it, and copying it back, without ever causing assertion of STOP_MOST. See the section on semaphore specification on GSC and GSC+ below for an explanation of the differences.
- DMA: The system performance effects of STOP_MOST are eliminated if a STOP_MOST disabled page can be used. The STOP_MOST enable/disable page choice primarily effects system performance. From a system performance perspective, if at all possible, all DMA should be done to the STOP_MOST disabled page types (either FAST or SAFE). The ability to use the STOP_MOST disabled pages is determined by the atomicity requirements of the memory accesses. Only devices requiring atomicity of accesses in excess of one cache line should need a STOP_MOST enabled page (i.e. some EISA devices). Even if semaphores are mixed with DMA data, if the atomicity requirements do no exceed ONE cache line,

the STOP_MOST disabled pages can be used. See the section on Semaphore Specification on GSC and GSC+.

- DMA: The transaction mapping effects of half cache line writes is determined by the FAST vs SAFE page type choice. Inbound DMA (memory writes) gets best performance on a FAST page; for outbound DMA (memory reads) FAST vs SAFE really makes no difference. To use a FAST page for inbound DMA requires that both the beginning and the end of a DMA stream should either be aligned to cache lines, or the DMA buffer should be made larger (by one half cache line) to align it to cache line boundaries, or buflets should be used. Buflets effectively separate the majority of the DMA (which is cache line aligned) from the beginning/end parts of the DMA (which may not be aligned). Thus the FAST DMA page can be used for the majority of the DMA transfer (the highest DMA performance), and the SAFE DMA page can be used for the unaligned portion at the beginning or end of the DMA. Preferably, inbound DMA should be aligned to cache line boundaries (even if there are small unused portions of a cache line at the beginning or end of the DMA). This allows the entire DMA to be done with one transfer (as opposed to buflets), and uses the high performance FAST DMA transactions.

6.5.7.2. Interaction of the PAGE TYPE with GSC+ XQL (prefetch) and LSL (lock) lines

In addition to the page type and prefetch enable fields read from the TLB, GSC provides an atomicity indication, and a prefetch request. The atomicity indication from GSC is on the LSL line which is active low; internal to UTurn this is renamed LOCK, which is active high. Similarly, the prefetch request from GSC is on the XQL line which is active low; internal to UTurn this is renamed NEXT, which is active high. These bits interact with the page type and prefetch enable bits to determine how transactions are mapped from GSC+ to Runway.

In general, assertion of LOCK requests atomicity of some type. For a STOP_MOST disabled page type, this atomicity is limited to one cache line, and results in a Runway read_priv to accomplish the atomicity request. For a STOP_MOST enabled page type, this atomicity is unlimited, and results in a Runway STOP_MOST assertion to lock all of memory. A Runway read_priv is still issued to provide the coherency required for read–modify–write (the sequence used for both sub–cache line writes and semaphores). The point here is that transaction mapping from GSC to Runway depends only on the GSC LSL line and the page type (FAST vs SAFE). The assertion of STOP_MOST on Runway depends on the GSC LSL line and the page type (STOP_MOST enabled vs STOP_MOST disabled). There is no cross–interaction between FAST/SAFE pages and STOP_MOST enable/disable page types. This can be seen in the tables below which independently describe the transaction mapping (FAST/SAFE pages and atomicity signal LOCK), and the assertion of STOP_MOST (STOP_MOST enable/disable and atomicity signal LOCK).

In general, assertion of XQL with a DMA read is a prefetch request. In order for prefetching to be done, the following requirements must be met. The GSC guest will only receive prefetching if XQL is asserted on a READ 4 or 8 (words). In addition, the LSL line must not be asserted (UTurn does not prefetch if the accesses has an atomicity request), and the prefetch enable TLB bit must be set, which allows prefetching on outbound DMA (memory reads).

6.6. Semaphore specification on GSC and GSC+

There are several ways in which a semaphore can be specified by a GSC+ guest, but only one way for a GSC guest. For GSC+ guests, either the semaphore transaction (CLEAR) can be used, or LSL (called

LOCK internal to UTurn) can be asserted for an atomic access. For GSC guests, LSL must be asserted to request an atomic access (either a semaphore or an atomic set of transactions). Also recall that the TLB page type field (STOP_MOST enabled vs STOP_MOST disabled pages) determines the level of atomicity provided when LSL is asserted. On a STOP_MOST disabled page, assertion of LSL provides atomicity for ONE read/write pair accessing ONE and ONLY ONE cache line. On a STOP_MOST enabled page, assertion of LSL provides atomicity of an arbitrary number of transactions to as much of memory as desired.

6.6.1. Semaphore specification on GSC

For GSC guests, semaphores MUST be of the form: READ (connected) with LSL asserted, followed immediately (and in the SAME bus tenure) by ONE WRITE to the SAME cache line address with LSL continuously asserted for BOTH transactions. GSC bus tenure for the read is guaranteed since the read MUST be connected; tenure must be continued through the write for atomicity of the semaphore. It is very important that the GSC semaphore adhere to the aforementioned semaphore specification. Other sequences WILL NOT result in an atomic access (required for semaphores).

Note that the above sequence results in a semaphore operation regardless of whether the page type is a STOP_MOST enabled or a STOP_MOST disabled. For atomicity limited to one read/write pair, the STOP_MOST disabled page type has the minimum impact on system performance, and is the most desirable choice. For atomicity requirements that exceed one read/write pair to one cache line, the STOP_MOST enabled page should be used. For example: READ (LSL) address1, WRITE (LSL) address1, WRITE (LSL) address1. On a STOP_MOST disabled page, the read and FIRST write would be atomic relative to one another, but NOT the second write, even though the second write is to the same cache line address, and LSL is continuously asserted. On a STOP_MOST enabled page, all transactions (the read, and BOTH writes) are atomic relative to one another. The sequence of transactions on Runway is the same in both cases; however, in the second case STOP_MOST is asserted for the duration of all three, thereby making them all atomic with respect to each other.

- BEST GSC semaphore: READ (connected), followed by ONE WRITE to the SAME cache line address with LSL continuously asserted for BOTH transactions to a STOP_MOST disabled page type. This is the best system performance semaphore possible on GSC.
- GSC ATOMIC (EISA): Any sequence of transactions with LSL continuously asserted to a STOP_MOST enabled page type. This provides atomicity for more than one read/write pair, and to an memory area larger than a single cache line. Due to system performance impacts of STOP_MOST, this should only be used where this level of atomicity is absolutely required (i.e. EISA devices with extensive atomicity requirements).

6.6.2. Semaphore specification on GSC+

GSC+ guest have two ways in which a semaphore can specified: either the GSC read/write pair or the GSC+ clear transaction. GSC+ guests can also use the STOP_MOST enabled page type which provides access to multiple–cache line atomic sequences IF NECESSARY.

GSC+ guests can perform semaphores using the same read/write pair as the above mentioned GSC semaphores. The same rules apply here. The semaphore MUST be of the form READ (connected) with LSL asserted, followed immediately (in the SAME bus tenure) by ONE WRITE to the SAME cache line address with LSL continuously asserted for BOTH. It is very important that the GSC+ semaphore using

the read/write pair adhere to the aforementioned semaphore specification. Other sequences WILL NOT result in an atomic access (required for semaphores).

Note that the above sequence results in a semaphore operation regardless of whether the page type is a STOP_MOST enabled or a STOP_MOST disabled page. For atomicity limited to one read/write pair, the STOP_MOST disabled page type has the minimum impact on system performance, and is the most desirable choice. For atomicity requirements that exceed one read/write pair to one cache line, the STOP_MOST enabled page should be used. Recall that a semaphore to a STOP_MOST enabled page is NOT the preferred semaphore for GSC+ guests due to the performance impact of assertion of STOP_MOST on Runway.

The preferential semaphore on GSC+ is the the CLEAR 16 transaction. This transaction performs an atomic read cache line and clear first word (identical with the NIO CLEAR 16 transaction) regardless of the page type or the LSL line. This allows the IOA to perform the semaphore in its local cache, without the assertion (and associated system performance penalty) of STOP_MOST on Runway.

For atomicity of MANY transactions (more than just a READ/WRITE) BOTH the assertion of LSL AND the STOP_MOST enabled page are required for both GSC and GSC+ guests. However, this level of atomicity (ownership of all of memory) should only be used where absolutely required due to the performance impact on the rest of the system.

- BEST GSC+ semaphore: CLEAR without assertion of LSL, or CLEAR to a STOP_MOST disabled page type. This provides an NIO style read and clear semaphore. This is slightly better performance than the read/write pair, since it is only one GSC+ transaction (instead of two for the read/write semaphore pair).
- NEXT BEST GSC+ semaphore: READ (connected), followed by ONE WRITE to the SAME cache line address with LSL continuously asserted for BOTH transactions to a STOP_MOST disabled page type. This is almost as good as the CLEAR transaction, but requires the additional address cycle for the write.
- GSC ATOMIC (EISA): Any sequence of transactions with LSL continuously asserted to a STOP_MOST enabled page type. This provides atomicity for more than one read/write pair, and to an area larger than a single cache line. Due to system performance impacts of STOP_MOST, this should only be used where this level of atomicity is absolutely required (i.e. EISA devices with extensive atomicity requirements).

6.7. Prefetching in the IOA

6.7.1. Deterministic vs Speculative Prefetch in the IOA

GSC+ provides a prefetch hint (called XQL on GSC+, and NEXT internal to UTurn) so that the IOA can prefetch for outbound DMA to reduce latency, and increase effective DMA bandwidth. As currently defined, GSC+ allows guests to indicate that they will want the next half or full cache line. Note that this is a deterministic prefetch. If a guest assertss XQL on a 16 byte read (GSC read4), the very next transaction from that guest MUST be a read4 of the next 16 byte chunk of memory, or the prefetch will be discarded (see the section on prefetch rules below).

So how does speculative prefetch fit in with this? Easy, any GSC+ module that wants to speculatively prefetch may do so with the following limitations. First, there must be an understanding of the DMA

model (with software) of the legality of performing speculative prefetch. Finally, in keeping with the GSC+ definition of prefetch, a GSC+ module may NEVER request a prefetch, and then not come back and request that read. Said another way, a GSC+ guest that wants to do speculative prefetching can assert XQL, but must request that location later, even if the data goes unused.

There is one other acceptable method of removing a prefetch once started. A guest may assert XQL, then decide it no longer wants the data. It must either read that location (and throw the data away, as discussed above), or it must read some other LEGAL location to cause the prefetch to be discarded. Note that the address must be one that the guest is allowed to read (i.e. an address already past in the DMA stream).

Note that under NO conditions may a module prefetch across a page boundary. This causes several problems. First, internal to the IOA we assume that if a TLB entry exists for a normal read, it also exists for the prefetch. A prefetch crossing a page boundary would violate this assumption. Second, prefetching across a page boundary assumes that physical memory exists, BUT it may not. This would at best cause a page fault, and the next page would be unnecessarily brought in from virtual memory (a disk read to fetch the page). At worst, the read from a non–existent page could cause a memory error and system crash. Prefetching across a page boundary is an architectural no–no; **DON'T DO IT!**

6.7.2. Deterministic Prefetch Implementation

The IOA implements a limited outbound deterministic prefetch. If a GSC+ guest asserts XQL (called NEXT internal to UTurn) on an appropriately sized transaction (READ 16 or READ 32) without assertion of LSL, and the prefetch enable bit from the UTurn TLB allows prefetching (i.e. prefetch enable bit set for that page), the IOA will fetch the appropriate cache line. Notice that the READ transaction requested by the GSC+ guest must be at least 16 bytes and LSL must not be asserted, the prefetch enable bit must be set for that page in the TLB, and XQL must be asserted. Under these conditions, the IOA will perform the requested read, and once that request is completed, will request the prefetch. Note that UTurn will read an entire cache line for a prefetch; however, if prefetch was caused by a 16 byte read, then only the appropriate half cache line will be returned to the GSC guest.

Note that the original read transaction information is placed into the Pool with the entry bit indicating a normal read. Similarly, the prefetch transaction information is also placed in the Pool with the entry type bit indicating a prefetch read. When the prefetch read is requested by the GSC+ guest, the prefetch's status bits are changed to indicate that the read has been formally requested; even though the entry type bit still indicates a prefetch read, the pool entry has logically been changed to a 'normal' read.

6.7.3. Prefetch Rules and Invalidation of Prefetch Entries

The prefetching implemented by the IOA is very limited, and has several restrictions. First, the IOA has only eight Pool entries for both normal reads and prefetch reads. Additionally, GSC guests may only have one outstanding normal read, and one outstanding prefetch read (this is a GSC+ bus protocol rule). This will work very nicely for GSC+ based devices reading down a single DMA stream, but provides limited benefits if multiple DMA streams are passed simultaneously through one GSC+ guest (i.e. bus converters). Due to the potential for these multiple streams to be intermixed, the single outstanding prefetch allocated per guest will cause prefetch thrashing. Prefetching in this case has limited (if any) benefit, and should probably not be used.

The following rules apply to prefetch in the IOA.

- One outstanding prefetch per GSC+ guest. Prefetch is limited to one and only one sequential DMA stream at a time. Two or more intertwined DMA streams from the

same guest will thrash the buffer, and generate additional memory requests with no benefit. Prefetch should be not be used in this case.

- Prefetch is limited to outbound DMA (GSC+ guest mastered reads) of size 16 or 32 bytes without LSL asserted on a page with the prefetch enable bit set in the TLB entry.
- A GSC+ guest's prefetch buffer will be discarded if the next transaction from that guest is not a read of the requested address or a DMA write. This allows prefetch data to be regularly discarded to solve the problem of GSC+ guests which get errors or get reset after requesting a prefetch. In cases where the GSC+ guest is a bus converter (i.e. intertwined DMA streams), this will cause prefetch data to be removed potentially before the GSC+ guest comes back to request it. Notice there are two ways this prefetch discard can be invoked. Either the next transaction from that guest has LSL asserted, or it is a read of the wrong address. An example: GSC guest 3 does a read16 at address 0 and indicates it wants a prefetch by asserting XQL. The prefetch will be discarded if guest 3 then reads some address OTHER than address 20 (then next cache line after 0). The prefetch will also be discarded if guest 3 performs any SHORT write (any write requiring a read–modify–write operation) to any address. The only way for the guest to get the prefetch data is for the read of address 20 to be the next READ transaction from guest 3 after requesting the prefetch. Note this allows a mix of inbound DMA (memory writes) with outbound DMA (memory reads) without discarding prefetches. However, this only works if the inbound DMA (memory writes) do NOT do read–modify–write (only writes of 8 words, or writes of 4 words to FAST pages).
- LSL causes a special case of prefetch discard. When LSL is asserted to mean split, UTurn does not know the GSC guest ID. Due to the problems of special casing splits vs locks; UTurn assumes any assertion of LSL prevents us from knowing the GSC guest ID, and UTurn discards ALL prefetches (from ALL GUESTS!) that have not yet been formally requested. NOTE: An actual GSC transaction must be done under the LSL split. If LSL is asserted to split a transaction, but no GSC transaction is mastered by the guest, then this assertion of LSL will NOT cause prefetchs to be discarded. Notice that if a guest formally requests the prefetch read, and is waiting for the data, the prefetch will NOT be discard, since the prefetch has been formally requested, and the guest expects a response.

Transaction type ===>	Reads with no LSL assertion	DMA writes (GSC write 8, write 4 to FAST page)	ANY GSC transaction with LSL asserted
Prefetch activity (Address match)	mark prefetch as 'requested'	— no action —	DISCARD prefetchs from ALL guests
Prefetch activity (Address miss)	DISCARD that guests prefetch	— no action —	DISCARD prefetchs from ALL guests

TABLE ? Prefetch discard behavior

The above table shows the conditions under which UTurn will discard prefetchs. Note that the GSC guest must master a transaction with LSL asserted to cause ALL prefetchs to be discarded. Just asserting LSL to split a host–mastered transaction does not cause prefetchs to be discarded.

6.8. Transaction Map (GSC+–>Runway)

The following tables indicate the interaction of the page types with the GSC XQL and LSL lines. Recall that the GSC+ prefetch indication is called XQL and is active low; internal to UTurn it is inverted (now active high) and called NEXT. Similarly, the GSC+ atomicity indication is called LSL and is active low; internal to UTurn it is inverted (now active high) and called LOCK.

The first table shows how the GSC LSL line (called LOCK internally) interacts with the STOP_MOST enable bit in the page type to enable or disable assertion of STOP_MOST on Runway. The second table shows how the transaction size, type, and LSL asserted on GSC interact with the FAST/SAFE page types to map GSC transactions to Runway transactions.

To minimize the impact to system performance, the STOP_MOST enable bit should only be set when multi–transaction atomicity is required or for GSC semaphores. For more information see the section on semaphore specification on GSC and GSC+.

6.8.1. Some general notes on the transaction map table and the atomicity table

TRANSACTION NOTE: the table below indicates use of a 'READ_SHAR', for Runway. The actual transaction used will be a 'READ_SHAR_OR_PRIV'. This was abbreviated for purposes of conserving space.

SIZE NOTE: the transaction length is reference to a Runway cache line of 32 bytes. Thus FULL is a full cache line (32 bytes), HALF is a half cache line (16 bytes), and PART is a partial half cache line (anything less than 16 bytes). Be careful, since GSC refer to transaction lengths in WORDS. So a GSC read 1 is a read of 1,2, or 4 bytes. Similarly a GSC read 2 is a read of 8 bytes, and a read 4 is a read of 16 bytes, and a read 8 is read of 32 bytes (or one cache line)

PAGE TYPE NOTE: FAST = fast DMA page, SAFE = safe DMA page, LOCK = transaction with LSL asserted regardless of the page type. Restated: the page types listed assume LSL is NOT asserted. If LSL is asserted, that page type is listed as LOCK. The allowable page types are listed below:

- FAST page: page type = x0 with LSL not asserted (LSL = 1)
- SAFE page: page type = x1 with LSL not asserted (LSL = 1)
- LOCK page: page type = xx with LSL asserted (LSL = 0)

STOP_MOST enable: Each page (either FAST or SAFE) can independently have STOP_MOST enabled or disabled. This allows programmatic control of the connection between GSC's LSL line and Runway's STOP_MOST line. See the sections above on Definition of the PAGE TYPE bits, and Semaphore specification on GSC and GSC+ for more details.

- STOP_MOST enabled: page type = 1x
- STOP_MOST disabled: page type = 0x

CONNECTED NOTE: All READs are required to be connected for GSC guests. GSC+ guests may choose to select non–connected (i.e. SPLIT) read data returns. However, any READ with LSL asserted is required to be connected as per the GSC+ bus specification. Similarly, any CLEAR is required to be connected. The connected bit also influences whether a GSC transaction is interpreted as a semaphore or not. See the section of semaphore specification of GSC and GSC+.

GSC+ TRANS	SIZE	PAGE TYPE	Connected	PREFETCH	Runway TRANSACTION	NOTE
READ	PART	FAST	0	NO	READ_SHAR	
		FAST	1	NO	READ_SHAR	
		SAFE	0	NO	READ_SHAR	
		SAFE	1	NO	READ_SHAR	
		LOCK	1	NO	READ_PRIV	1,5,7

GSC+ TRANS	SIZE	PAGE TYPE	Connected	PREFETCH	Runway TRANSACTION	NOTE
READ	HALF	FAST	0	YES	READ_SHAR	3
		FAST	1	YES	READ_SHAR	3
		SAFE	0	YES	READ_SHAR	3
		SAFE	1	YES	READ_SHAR	3
		LOCK	1	NO	READ_PRIV	1,5,7

GSC+ TRANS	SIZE	PAGE TYPE	Connected	PREFETCH	Runway TRANSACTION	NOTE
READ	FULL	FAST	0	YES	READ_SHAR	3
		FAST	1	YES	READ_SHAR	3
		SAFE	0	YES	READ_SHAR	3
		SAFE	1	YES	READ_SHAR	3
		LOCK	1	NO	READ_PRIV	1,5,7

GSC+ TRANS	SIZE	PAGE TYPE	Connected	PREFETCH	Runway TRANSACTION	NOTE
WRITE	PART	FAST	0	NO	READ_PRIV,WRITE_BACK	4
		FAST	1	NO	READ_PRIV,WRITE_BACK	4
		SAFE	0	NO	READ_PRIV,WRITE_BACK	4
		SAFE	1	NO	READ_PRIV,WRITE_BACK	4
		LOCK	0	NO	READ_PRIV*,WRITE_BK	4,5,6
		LOCK	1	NO	READ_PRIV*,WRITE_BK	4,5,6

GSC+ TRANS	SIZE	PAGE TYPE	Connected	PREFETCH	Runway TRANSACTION	NOTE
WRITE	HALF	FAST	0	NO	WRITE16_PURGE	
		FAST	1	NO	WRITE16_PURGE	
		SAFE	0	NO	READ_PRIV,WRITE_BACK	4
		SAFE	1	NO	READ_PRIV,WRITE_BACK	4
		LOCK	0	NO	READ_PRIV*,WRITE_BK	4,5,6
		LOCK	1	NO	READ_PRIV*,WRITE_BK	4,5,6

GSC+ TRANS	SIZE	PAGE TYPE	Connected	PREFETCH	Runway TRANSACTION	NOTE
WRITE	FULL	FAST	0	NO	WRITE_PURGE	
		FAST	1	NO	WRITE_PURGE	
		SAFE	0	NO	WRITE_PURGE	
		SAFE	1	NO	WRITE_PURGE	
		LOCK	0	NO	READ_PRIV*,WRITE_BK	4,5,6
		LOCK	1	NO	READ_PRIV*,WRITE_BK	4,5,6

GSC+ TRANS	SIZE	PAGE TYPE	Connected	PREFETCH	Runway TRANSACTION	NOTE
CLEAR	HALF	FAST	1	NO	READ_PRIV,WRITE_BACK	2,7
		SAFE	1	NO	READ_PRIV,WRITE_BACK	2,7
		LOCK	1	NO	READ_PRIV,WRITE_BACK	2,5,7

* = READ_PRIV may have already been done due to previous READ. See note 6 below.

TABLE ? Transaction map GSC+ –> Runway and effects of PAGE TYPE and CONNECTED

NOTES:

1) MAYBE a semaphore (transaction=READ with LSL=0 (asserted)) This MAY be a semaphore, implemented as READ_PRIV. If followed by a WRITE to the SAME address it is a semaphore, so modify in cache, WRITE_BACK. Any READ with LSL asserted is required to be a connected transaction as per the GSC bus specification. Special hardware enforces this. There is NO SUCH thing as a non–connected read if LSL is asserted. If XQL is asserted, it is disregarded, since prefetching is NOT allowed if atomicity is requested (LSL asserted). READs with LSL asserted may be semaphores (UTurn will issue read_priv); but if the next transaction is NOT a WRITE to the SAME address, then it was not a semaphore and the cache line read privately is released. See note 6 below.

2) The unambiguous semaphore for GSC+ guests ONLY. The CLEAR transaction is only defined for GSC+ guests, and only for a four word length (half cache line). The CLEAR transaction reads four words of data passing the data back to the GSC+ guest assuming a connected read, clears the first word, and writes the four words back ALL ATOMICALLY. CLEAR is required to be requested as a CONNECTED transaction as per the GSC+ bus specification. Special hardware in the inbound side forces a connected return of the data EVEN if the guest did not request it. There is NO SUCH thing as a non–connected CLEAR transaction. LSL may be asserted for the CLEAR, but it makes very little sense. CLEAR is

already an atomic read/write, asserting LSL would only impact system performance by stalling all processors, and other DMA activity from the other IOAs/UTurn.

3) Prefetch will be performed if requested (by assertion of XQL), and enabled (where indicated by a 'YES' in the PREFETCH column in the tables above), and the prefetch enable bit is set for the TLB entry for the page being accessed.

4) READ_PRIV, WRITE_BACK combination is used to perform a read–modify–write in the IOA

5) Prevent all other Runway devices from accessing memory to guarantee multi–cache line atomicity for EISA or ISA devices. This mode is very safe, and has tremendous performance impact since all other devices (processors, other IOAs) are prevented from accessing memory. This should be avoided unless absolutely necessary since there is a large performance impact on the rest of the system. See the section on semaphore specification on GSC and GSC+ to find out how to minimize the assertion of STOP_MOST for semaphores. Note that STOP_MOST will only be asserted if the guest asserted LSL AND the page type has STOP_MOST enabled. Otherwise atomicity request is limited to one cache line, and STOP_MOST is NOT asserted.

6) WRITE transactions with LSL asserted MAY be the second half of a READ/WRITE semaphore combination. This requires that the IOA check the contents of the cache to see if a READ_PRIV to the same address was ALREADY performed. If a READ_PRIV was NOT already done, then it will be issued on Runway. If a READ_PRIV to the SAME address was done just prior to the WRITE, then the WRITE transaction will NOT generate a READ_PRIV. Recall, that semaphores on GSC are required to be a READ (connected) with LSL asserted followed IMMEDIATELY by a WRITE with LSL asserted to the SAME cache line address. Semaphores on GSC+ should use the CLEAR transaction. NOTE that full cache line writes that look like the second half of semaphores will use read–modify–write in order to get maintain proper cache coherency with the processors.

7) GSC+ guest may request that reads be split (also called 'pended' or not connected). However, if LSL is asserted OR if the GSC+ transaction is a CLEAR, then the GSC+ data return MUST be connected. Special hardware on the inbound side guarantees this. See the GSC+ bus specification for details. GSC guests do not have any options for read returns, they are connected.

The table below shows the interaction of the GSC atomicity line and the TLB STOP_MOST enable bit that together determine if STOP_MOST will be asserted on Runway. Recall that the GSC atomicity indication is active low (LSL), whereas internal to UTurn this signal is inverted (high true) and called LOCK.

GSC LOCK indication (high true version of LSL)	TLB STOP_MOST enable bit	Runway atomicity (STOP_MOST)	ATOMICITY NOTE
NO LOCK (lsl = 1)	DISABLE (0)	NO	disabled, not requested
LOCK (lsl = 0)	DISABLE (0)	NO	disabled, requested
NO LOCK (lsl = 1)	ENABLE (1)	NO	enabled, not requested
LOCK (lsl = 0)	ENABLE (1)	YES	enabled, AND requested

TABLE ? Atomicity specification interaction between GSC LSL line and TLB STOP_MOST enable bit

6.9. Hardware Blocks

6.9.1. Inbound Queue

The Inbound Queue (InQ) is used to synchronize and store all inbound transaction information (note that the data associated with a transaction is stored in the INBOUND RAM, not in the queue). The InQ is written to by the GSC+ master and slave state machines. Three types of transactions are placed in the InQ: guest mastered GSC+ transactions (memory reads and writes), data returns from GSC+ guests initiated by Runway direct IO reads, and read returns of IOA internal HPA registers. The ENTRY bit is used to distinguish between guest mastered GSC+ transactions (ENTRY = 0), and data returns and IOA internal HPA register accesses (ENTRY = 1).

1	6	3	1	4				4	20	10				1	1	1	1
0 = GSC	1 6 NOT USED	7 9 GSC guest ID	10 Force time–out for read	11 14 GSC byte enables				15 18 GSC trans type	19 38 GSC addr (virtual page number)	39 48 GSC addr (page offset)				49 connect	50 lock	51 next	52 diag
1 = data rtn	Run–way TransId	Run–way Mastr ID	NOT USED	11 HPA REG	12 TLB CMD	13 rtn zeros	14 NOT USED	data return type	NOT USED	39 NOT USED	40 REG ADR	41 43 NOT USED	44 48 HPA REG addr (lsb)	NOT USED	NOT USED	NOT USED	NOT USED

TABLE ? InQ entries and descriptions
(ENTRY = 0: GSC+ sourced transactions) (ENTRY=1: data returns, tlb commands)

6.9.1.1. Inbound queue entry (0 = GSC mastered transaction)

For guest mastered GSC+ transactions, ENTRY=0, and the InQ entries have three special bits (CONNECTED, LOCK, NEXT) that effect how the GSC+ transaction is processed. See the section above (interaction of TLB PAGE TYPE bits with LOCK, NEXT, and CONNECTED) for an explanation of how LSL interacts with the TLB page type bits.

The GSC guest ID indicates which guest to return data to (for reads) or which GSC guest caused a problem (for error logging).

The Force timeout bit is used for reads only. It allows a dummy entry to be placed in the pool, which will cause a data return on GSC with an error indication. This is used to get a connected guest read off GSC once an error has occurred.

The GSC trans type indicates which transaction (READ, WRITE, CLEAR) and how long (word, double word, four word, or eight word).

The GSC addr indicates the target address of the GSC transaction. The GSC address is composed of two parts: the IO virtual page number (used to access the TLB), and the IO page offset (passed on to Runway).

The CONNECTED bit is used to indicate that the read on GSC+ was a connected read, and the data needs to be returned immediately, bypassing the OutQ.

The LOCK indication is the high true (inverted) version of the GSC atomicity line called LSL. LOCK is asserted to request an atomic memory access. The level of atomicity is determined by the value of the TLB STOP_MOST enable bit. If the TLB STOP_MOST enable bit is set, STOP_MOST is asserted on Runway for GSC transactions with LSL asserted. This provides the GSC guest with the highest degree of atomicity (ownership of ALL of memory). If the TLB STOP_MOST enable bit is NOT set, STOP_MOST is never asserted on Runway regardless of the assertion of LSL. This provides the GSC guest with atomicity limited to ONE read/write pair accessing ONE cache line. See the section above on Definition of the Page Type bits, and the section on Semaphore Specification on GSC and GSC+ for more details.

The NEXT bit is the high true (inverted) version of the GSC prefetch line called XQL, used by a GSC+ guest to indicate that a prefetch can be performed on the next sequential address to reduce latency on sequential outbound DMA. See the section on deterministic prefetch, and the description of the PREFETCH area for further description.

The InQ entries for guest mastered GSC+ transactions contain the the Diag bit, used to indicate a special UTurn internal diagnostic transaction. This is passed along to the pool (i.e. this bit does not directly affect behavior on the inbound Runway side).

6.9.1.2. Inbound queue entry (1 = GSC data returns, internal data returns, TLB commands)

For data returns bound for Runway, and for IOA internal HPA register reads the bit ENTRY=1. There is a different InQ entry type containing different fields. The address field is unused, and the remaining fields are redefined to include:

The TRANS_ID indicates what transaction id should be used when the data is returned.

The MASTER_ID indicates which Runway master to return the data to.

HPA REG indicates (if asserted) that this entry is a read of an HPA register (the address of the register is indicated in the HPA REG ADDR field).

TLB CMD indicates (if asserted) that this entry is a TLB command (insert, purge, write)

Return Zeros is used to force a data return of zeros on Runway. This allows us to return zeros on reads of unimplemented registers, and return zeros for reads of the io status register during command reset (i.e. UTurn is NOT ready).

DATA RETURN TYPE which distinguishes GSC+ data returns from HPA register data returns from other special transactions. These 'special' transactions are a catch–all for transactions not encoded in the GSC+ transaction field (e.g. a GSC+ interrupt which needs to be translated into a Runway WRITE_SHORT to the interrupt destination address).

The HPA REG ADDR (msb, four lsbs) indicates which HPA register to read.

6.9.2. Pool

The Pool is used for temporary storage of transaction information. There are two types of Pool entries: one for normal reads, and one for prefetch reads. A Pool entry is written anytime a transaction expecting return data is driven out onto Runway. This Pool entry is used by the Runway receiver side to know which GSC+ guest to return the data to when a data read return is captured on Runway. Pool entries are written to when a normal read or a prefetch read is at the head of the InQ. The Pool entry is used when data is returned to create a data return entry in the OutQ . Note that there are eight Pool entries, one for each possible outstanding transaction, since the IOA is limited to only eight transaction IDs on Runway. This makes is unnecessary to store the transaction id, as each Pool entry has a hardwired transaction id associated with it.

See the section in chapter five on Pool Buffers for details of the physical positions of the pool fields within the Pool.

6.9.2.1. Pool fields for normal reads

Two different types of entries are created depending on whether the read was a regular read or a prefetch read. For normal reads the entry bit field in the Pool is a zero (0). The normal read pool contains the following fields:

The IN–USE bit indicating that the Pool entry contains valid information.

The ENTRY bit indicating either normal read entry or prefetch entry.

The RAMADDR bit used by the outbound side to determine which RAM location to use for the returned data.

The TIMEOUT bit used to keep track of timeouts on responses to read requests.

The TIMED_OUT bit used to record which transaction timed out.

The TLB bit indicates that the data return is necessary to service a TLB miss, and the data should be placed immediately into the TLB.

The CACHE bit indicates that the data return is bound for the cache, and the data should be placed immediately into the cache location.

The CONNECTED bit indicates that the data return is necessary to service a connected read on GSC+. This is necessary as the GSC+ bus is tied up until the data is returned, so the data return must bypass all other entries in the OutQ to avoid deadlock. If both the CACHE and the CONNECTED bits are set, then the return data should be placed in the cache and returned to the GSC guest.

The GUEST_ID field which indicates to the outbound side which GSC guest to return the data to.

A Portion of the GSC TYPE field is also stored, indicating to the outbound side how much data should be returned to the GSC guest.

The GSC_ADDR field stores the address of the desired word on the cache line since entire cache lines are read from memory, but the GSC guest may only want one word.

6.9.2.2. Pool fields for prefetch reads

The Pool area is also used for temporary storage of transaction information for prefetch reads. A prefetch entry is written when an appropriately sized (4 word or 8 word) GSC+ read transaction has the XQL line

asserted (see the table above on the interaction of the page type and prefetch hint bits with GSC+ transactions, and the section on prefetching). Prefetch requires a couple special bits to keep track of the status of the prefetch. These are the REQ_BY_GSC, RETURNED, and DISCARD bits explained below. The prefetch entries contains roughly the same information as the normal read entries, but prefetches do not use the TLB, or CACHE bits. Prefetch Pool entries contain the following fields:

The IN–USE bit identical as for normal read pool entries.

An ENTRY type bit which always has the value one (1) for prefetch reads.

RAMADDR, TIMEOUT, TIMED_OUT, GUEST_ID, and TYPE fields are used in the same way as for normal reads.

The CONNECTED bit is not valid when the prefetch is first issued and is set to zero. However, when the GSC guest comes back to formally request the read (i.e. causing the REQ_BY_GSC bit to be set), the guest may request the read be connected. At this point the CONNECTED bit can be set to indicate that the data should be returned immediately via the connected read return queue. See the section in the OUTBOUND chapter on the connected read return mechanism and the section on the pool.

RETURNED indicates that the prefetch data has been returned on Runway.

REQ_BY_GSC indicates that the prefetch has been formally requested on the GSC+ side. Once formally requested, the prefetch read is now a normal read, though the entry type bit does not change.

The DISCARD bit is used to indicate that the pool entry should be marked as unused, and the data discarded once it returns. This allows the IOA to flush the prefetch buffers periodically to prevent a stale data problem if a GSC guest requests a prefetch and then does not formally read it (GSC card gets reset or takes an error). The Pool entry must be maintained so the returning data is accepted by the IOA, even though it will not be used. See the section on prefetch for more details. The GSC address is stored so that it can be compared against reads from that same GSC guest to determine when the prefetch turns into a normal read (i.e. formally requested).

6.9.3. TLB

As discussed previously, the TLB is a cache of address translation entries from the IO PDIR or pre–loaded by software into the TLB. The TLB allows the IOA to translate from the 32 bit GSC+ addresses to the 40 bit Runway physical address necessary for accessing memory. The TLB contains the following fields:

Eight bit of virtual index corresponding to the physical address which processors use to snoop their caches to maintain coherency. The virtual index is driven onto Runway for all coherent transactions.

Twenty Eight bits of real page number which is concatenated with the twelve bit IO offset to form a forty bit Runway physical memory address.

One prefetch enable bit used to enable prefetching for the page in question. Note prefetching will only be performed if the GSC+ read transaction is a half cache or full cache line, and the guest requests a prefetch (by asserting GSC+ XQL line), and the prefetch enable bit is set for that page, and no atomicity is requested (i.e. LSL is NOT asserted for that read transaction).

Two bits of page type, used to assist in mapping GSC+ transactions to Runway transactions. See the section in this chapter on Definition of the PAGE TYPE bits, and the section on Transaction Map (GSC+ –> Runway).

Twelve bits of tag used to determine a TLB hit. The TLB tag is written from and compared to the GSC+ address bits [8:19]. GSC+ address bits [20:29] are the IO offset, and address bits [0:7] are typically the address into the TLB RAM (though this depends on the value written to the IO CHAIN ID MASK register). See the section above on accessing the TLB for a more detailed explanation of how the GSC+ address is used in the TLB.

Note that the IOA architectural IO_CONTROL_HV_MODE register selects the address translation mode. See the section above on address translation mechanisms.

0 9 10 bits	10 37 28 bits	38 1 bit	39 40 2 bits	41 52 12 bits	53 1 bit
Runway VIRTUAL INDEX	Runway REAL PAGE NUMBER	pre– fetch enble	PAGE TRANS MAP	TLB TAG FIELD	Valid

TABLE ? TLB fields and physical positions within the TLB RAM

6.9.4. Cache

Each IOA has a small one line cache used for short writes (writes of a half cache line or less), and for atomic read/write combinations (semaphores). Any time the IOA reads a memory location privately the data is placed into the cache. Read privates are only issued for short writes (requiring read–modify–write), and semaphores (or things that 'look' like semaphores — see the section on Transaction Map GSC+ –> Runway). Data is loaded into the cache when the CACHE bit is set into the pool on a read operation. When the data is returned on Runway, the outbound side asserts load signals to cause the data to be loaded into the cache location.

Once the cache is loaded with the data from the read private, the inbound side is able to modify the data in the cache. Each of the 32 bytes in the cache is individually addressable. The inbound side is able to modify any one, two, three, four, eight, or sixteen bytes in the cache (with some addressing restrictions). As with all PA memory accesses, writes are required to be quantity aligned (i.e. write four bytes must be done on a word address boundary). There is also special hardware to support the CLEAR 16 GSC+ semaphore. Recall the CLEAR transaction reads a cache line (privately), returns the data to the GSC+ guest, writes all zeros to the FIRST word in the cache line, and writes the cache line back to memory.

In addition to the storage for 256 bits of data (a cache line), and steering logic for loading the cache from the inbound side, and writing individual bytes in the cache, the state of the cache line is stored with a small state machine. Refer to the section on the cache for more details.

6.10. Inbound Side HPA registers

Due to the locality of their use, some of the IOA internal HPA registers have dedicated outputs to the INBOUND side even though they are physically located in the OutQ side of the chip. These registers are written to by the OUTBOUND side, and read from the INBOUND side. This has the interesting side effect that writes take effect immediately, whereas reads go through the OUTBOUND and then the INBOUND queues. This introduces the possibility of a read being bypassed by a subsequent write to the register. This can be avoided by having software wait for the read return before issuing subsequent writes.

The following IOA internal HPA registers control aspects of the inbound transactions or TLB access: IO_COMMAND, IO_STATUS, IO_CONTROL_HV_MODE, IO_TLB_ENTRY_M,

IO_TLB_ENTRY_L, IO_PDIR_BASE, IO_CHAIN_ID_MASK, Runway_TIMEOUT, INTERRUPT_DESTINATION, TOC_DESTINATION. For more information on IOA internal HPA registers refer to the chapter on Architectural Requirements.

7. UTurn 120+ MHz Logic

A portion of the UTurn logic must operate at the full Runway clock rate (120 MHz or more). This logic includes the Runway pads, the interface between the Runway pads and the I/O Adapter in the core, and the Runway arbitration block. This chapter will describe the functionality of these blocks.

7.1. Runway Pads

Electrical details regarding the Runway I/O pads are included in the electrical chapter of this document.

7.2. Signals Connecting Runway Pads With IOAs

Detailed technical documentation regarding design and timing issues can be found in chapter {Internals}.

7.3. Runway Arbitration

There can be up to two UTurns (or four I/O Adapters) in a system. These I/O Adapters coexist on the same client sub–group arbitration net. This means that the four I/O Adapters arbitrate amongst one another to determine which one wins Runway. The Runway arbitration policy implemented in the UTurns is dual round robin.

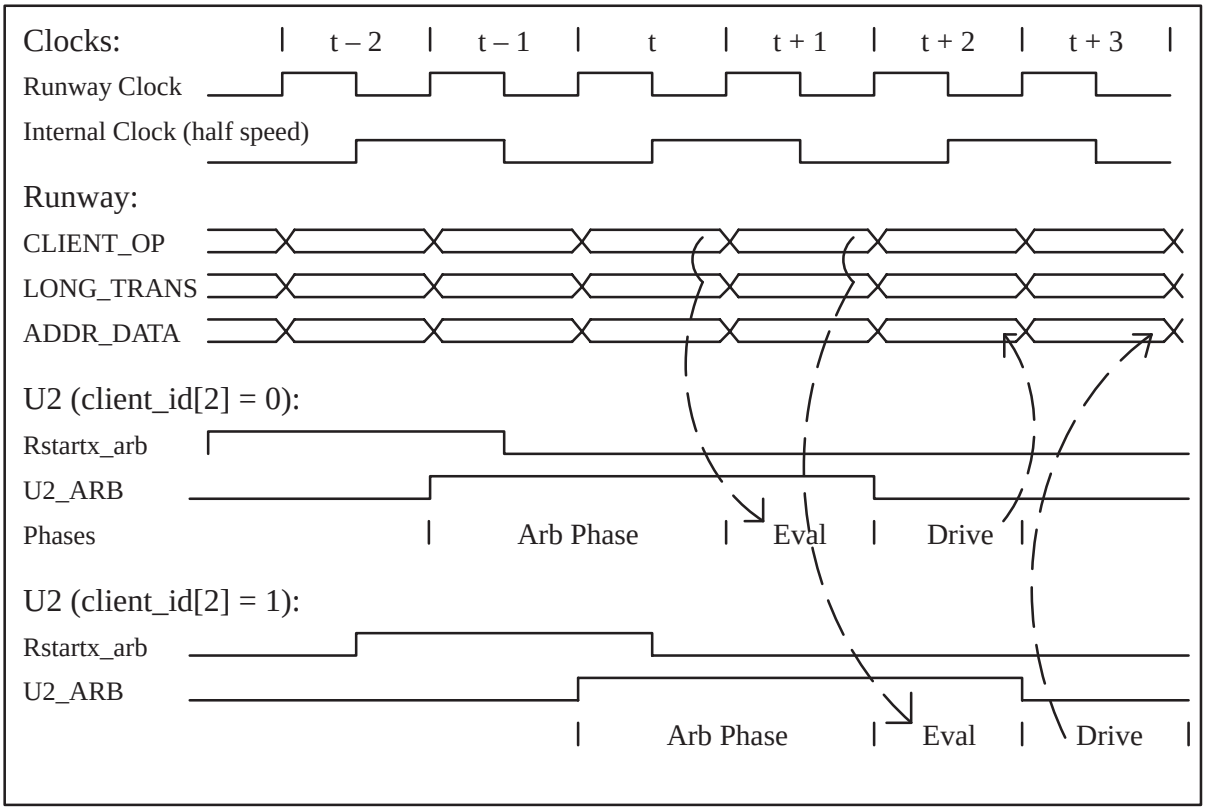
Each UTurn keeps track of three pointers: one pointer points to the highest priority I/O Adapter for the ANY_TRANS or NO_IO CLIENT_OP, the second pointer points to the highest priority I/O Adapter for winning the ATOMIC CLIENT_OP, and the third pointer points to the highest priority I/O Adapter for all other CLIENT_OPs. For each pointer, when I/O Adapter #2 finishes its transaction (or its atomic hold, in the case of the ”stop_most” pointer), I/O Adapter #0 is highest priority for ownership, followed by #3, #1, with I/O Adapter #2 lowest (assuming a two UTurn subnet). If no I/O Adapters arbitrate, then the pointers don’t advance.

7.3.1. Algorithmic Details

Whenever an I/O Adapter wants to drive a transaction, it asserts its Rstart_arb line. The arbiter for that UTurn then drives the UTurn’s outgoing UTurn_ARB signal based on the Rstart_arb line being asserted now or having been asserting previously without a corresponding bus win.

Each UTurn receives the UTurn_ARB line of the other UTurn. When either of the two UTurns has arbitrated in each of the current cycle and the last cycle, it will win Runway the second cycle after the current cycle. (Two cycles of arbitration are required due to UTurn residing on the I/O client sub–group arbitration net.) The two cycles in which the arbitration lines are driven is referred to as the ”ARB” phase. The following cycle (known as the ”EVAL” phase) is when both UTurns evaluate the CLIENT_OP from the previous Runway cycle, LONG_TRANS from the previous Runway cycle, UTurns’ priority relative to each another (indicated by the pointers described in the previous section), and both UTurns’ assertion of the arbitration lines as of the previous cycle, to determine (unanimously) the arbitration winner. The very next cycle is the ”DRIVE” phase, when the winning UTurn drives Runway. Although this is a four cycle exercise, these phases are all pipelined, so that in any given state, the UTurns may actively assert their respective UTurn_ARB lines, the host drives the CLIENT_OP bus, each Runway client performs their evaluation, and the current owner of Runway drives LONG_TRANS.

The following timing diagram illustrates internal UTurn timing relative to Runway, and the different phases of arbitration:



7.3.2. Forward Progress Guarantee

When an ANY_TRANS or NO_IO CLIENT_OP occurs, the I/O Adapter is allowed to drive any transaction. Therefore, the ”any” pointer guarantees forward progress and non–starvation for all I/O Adapters, since the ANY_TRANS and NO_IO CLIENT_OPs have to occur occasionally, at which point the highest priority I/O Adapter is guaranteed that it can issue its transaction, and then the pointer advances.

The ”round–robin” pointer allows progress to continue, even though one or more I/O Adapter may be arbitrating for a transaction that is currently disallowed. When this occurs, the I/O Adapter will drive idle cycles and back off from arbitration, passing control to the processor subnet to guarantee forward progress. Eventually the CLIENT_OP will change back to ANY_TRANS or NO_IO, at which point priority automatically switches back to the ”any” pointer which will eventually advance to the starving I/O Adapter.

When the client_op is atomic, the stop_most pointer indicates which I/O Adapter is allowed to drive any transaction. All other clients on Runway view the atomic client_op as an effective ret_only, and they can therefore only drive return transactions. Following an atomic sequence, all I/O Adapters must back off from Runway arbitration until an ANY_TRANS CLIENT_OP is observed thus guaranteeing forward progress.

The following table illustrates what UTurn will drive on Runway based on the client_op and the pending transaction type:

Client_Op	Transaction Type	Runway Cycles
Shared Return	Any Transaction Type	Doesn’t Drive

Host Control	Any Transaction Type	Doesn't Drive
None Allowed	Any Transaction Type	First Cycle: Idle Second Cycle: Doesn't Drive
One Cycle	One Cycle Transaction Type	First Cycle: Transaction Second Cycle: Doesn't Drive
	Multi Cycle Transaction Type	First Cycle: Idle Second Cycle: Doesn't Drive
Return Only	One Cycle Return Transaction Type	First Cycle: Return Data Second Cycle: Idle
	Write_Back	Drive Complete Transaction
	Non Return Transaction Type	First Cycle: Idle Second Cycle: Idle
No I/O	Any Transaction Type	Drive Complete Transaction
Atomic	Any Transaction Type from Atomic Owner	Drive Complete Transaction
	Return Transaction Type from non Atomic Owner	Drive Complete Transaction
	Non Return Transaction Type from non Atomic Owner	First Cycle: Idle Second Cycle: Idle
Any Trans	Any Transaction Type	Drive Complete Transaction

7.3.2.1. Starvation Metaprotocol

Without some additional metaprotocol, multiple UTurns have the ability to starve the processors. There are three particular instances of this type of starvation. One occurs simply as a consequence of having the I/O client sub–group arbitration net as highest priority on Runway. If both UTurns are able to arbitrate for only 50% of the Runway cycles, by alternating back and forth they could potentially prevent the processors from ever winning the bus. The metaprotocol required to avoid these starvation cases is for the I/O Adapters as a group to periodically back off from arbitration until an uncontested ANY_TRANS CLIENT_OP occurs. The current implementation counts up to 256 cycles and if one of these cycles was an any_trans client_op won by an I/O Adapter, then all UTurns back off from arbitration until either an uncontested any_trans client_op is detected, an uncontested no_io client_op is detected, or an I/O Adapter within this UTurn is arbitrating for a return or atomic transaction.

Likewise, starvation could occur if a processor wishes to issue a coherent request (such as READ_PRIV) and the CCC queue in the host is full. In this case the CLIENT_OP will be RET_ONLY, preventing the processor from issuing its transaction. Conceivably, UTurn could arbitrate as soon as the CCC queue has a vacancy, thereby immediately filling the CCC queue. If UTurn consistently wins arbitration and fills the queue, the processor will be starved since it only wins the bus when the CLIENT_OP is RET_ONLY. To recover from this starvation case, UTurn will back off from arbitration whenever it wins a ret_only client_op but does not have a return transaction to issue or whenever the client_op is atomic and an I/O Adapter which is not the winner of this atomic client_op is arbitrating for something other than a return transaction. UTurn will continue to back off from arbitration until either a host_control, shar_return, none_allowed, one_cycle, no_io, or any_trans client_op occurs.

Finally, the third starvation case exists when both UTurns are allowed to simultaneously assert STOP_MOST enabling them to win back to back atomic sequences, potentially preventing the processors from ever winning the bus. To avoid this starvation case, I/O Adapters must back off from asserting STOP_MOST and arbitrating for an atomic client_op when either another I/O Adapter is already in the midst of an atomic sequence or when the I/O Adapter has just completed an atomic sequence. The back off will clear when an any_trans or no_io client_op is observed. This mechanism not only prevents starvation caused by UTurns alternating back and forth with atomic sequences, but also by preventing the same I/O Adapter from continuously arbitrating for atomic sequences.

7.3.2.2. Deadlock Metaprotocol

Metaprotocol is necessary to avoid deadlock. A deadlock case will occur if UTurn wants to issue a coherent request (such as READ_PRIV) and the CCC queue in the host is full. In this case, the CLIENT_OP will be RET_ONLY, preventing UTurn from issuing its transaction. And the processor which needs to issue the return to allow the CCC queues to progress, never wins because the I/O Adapter continuously arbitrates, hoping for a favorable CLIENT_OP.

The metaprotocol solution to this problem is for the I/O Adapter to back off from arbitration when it wins, but cannot issue its transaction because the effective CLIENT_OP is RET_ONLY. In order to ensure forward progress during the arbitration back off, the host must go from a RET_ONLY CLIENT_OP to either HOST_CONTROL, SHAR_RTN, NONE_ALLOWED, ONE_CYCLE, NO_IO, or ANY_TRANS. It is when one of these six CLIENT_OPs is issued, that the I/O Adapter can continue arbitrating.

7.3.2.3. Backoffs

As described in the previous sections, the method for guaranteeing forward progress and avoiding deadlock is arbitration backoffs. The following table illustrates when backoffs are turned on and off:

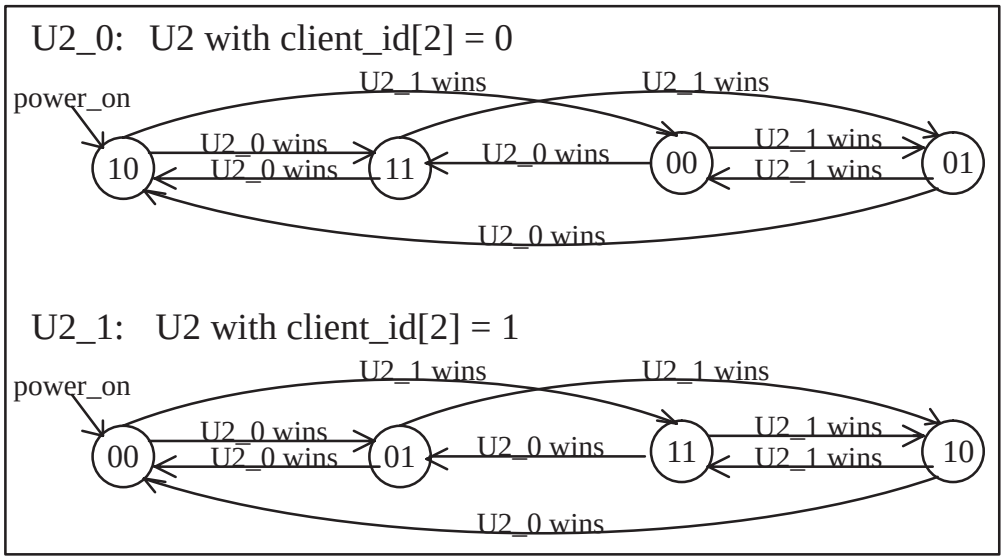
BackOff	**Enabled**	**Disabled**
Any	one of past 256 cycles was an any_trans client_op won by UTurn	uncontested any_trans client_op is detected, uncontested no_io client_op is detected, or IOA is arbitrating for a return or atomic transaction
Ret_Only	client_op=ret_only but UTurn does not have a return transaction to issue or client_op=atomic and IOA which is not atomic owner is arbitrating for something other than a return transaction	client_op is host_control, shared_return, none_allowed, one_cycle, no_io, or any_trans
Atomic	client_op=atomic and non–owning IOA is also arbitrating for an atomic sequence or whenever an IOA has just completed an atomic sequence	client_op is any_trans or no_io

7.3.3. ”Any” Priority Function

The ”any” pointer points to the I/O Adapter with the highest priority. It is a two bit pointer – the most significant bit indicates which UTurn (0=local and 1=remote) and the least significant bit indicates which I/O Adapter within a UTurn (0=ioa0 and 1=ioa1). To distinguish between the UTurns and guarantee

unanimous arbitration decisions, the pointer comes up in identically mirrored states within each UTurn. So, the UTurn distinguished by client_id[2] = 0 will initialize its ”any” pointer to 2’b10 and the UTurn distinguished by client_id[2] = 1 will initialize its ”any” pointer to 2’b00, where the symbol 2’b simply indicates that this is a two bit binary number. In this way, both UTurns agree that the highest priority I/O Adapter is ioa0 within UTurn client_id[2]=1.

Once power is stable, the ”any” counter will advance on every relevant evaluation phase. An evaluation phase is considered relevant if power is stable, the CLIENT_OP is either NO_IO or ANY_TRANS, LONG_TRANS is false, and this cycle isn’t a default drive cycle caused by a previous arbitration win. If all of these conditions are met, then the pointer will exhibit the behavior illustrated below:



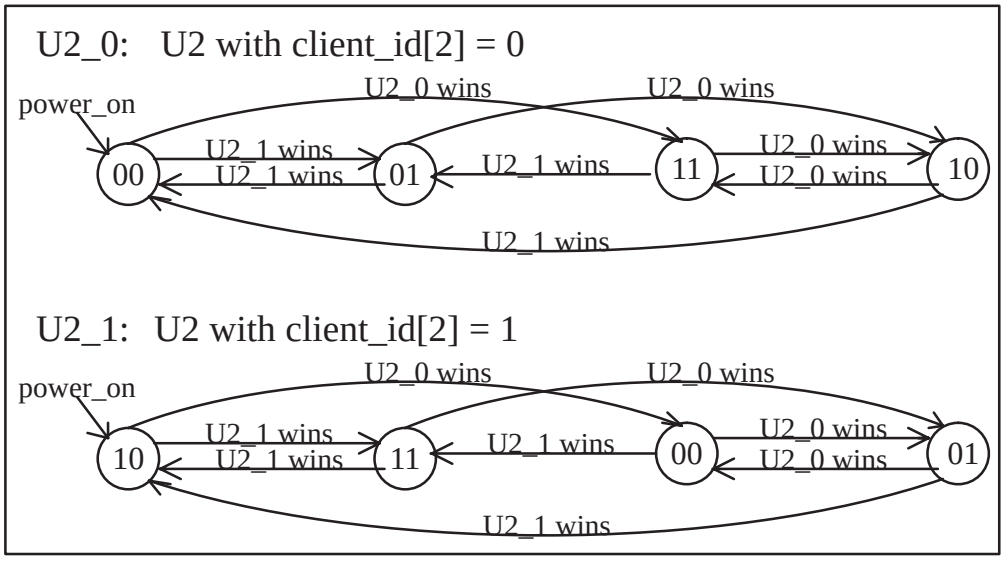
Note that if neither UTurn has arb’d, then neither will win, and therefore, the pointer will not advance. In addition to changing state, if a UTurn has won, then three lines must be asserted to drive the Runway bus. The signals ioa0_select and ioa1_select indicate which ioa, of the UTurn’s two internal ioas, wins; if neither ioa wins then ioa_idle must be asserted to cause an idle cycle on Runway. Also, the arbitration block is responsible for asserting the runway_drv line which goes directly to the Runway pads as a drive enable. Finally, the state variable second_cycle_default_win is asserted. If this UTurn won an ANY_TRANS CLIENT_OP and the transaction was a single or two cycle transaction, then this UTurn is responsible for driving the following cycle, with either idle or valid transaction data, respectively. Likewise, if this transaction is greater than two cycles, then both UTurns must be aware that the bus is not available on the cycle following the first drive cycle, even though LONG_TRANS is not observable until the pads are already loaded for the second cycle. Second_cycle_default_win fires on every UTurn

arbitration win, to ensure that Runway is always driven valid on the following cycle. Once an ioa is selected, the select line will remain asserted until the ioa's transaction buffers are drained. The ioa decrements a transaction length counter and asserts the signal, ioa_deselect, to indicate when the transaction is complete, at which point the arbitration block will deassert the corresponding select line and runway_drv.

7.3.4. Round Robin Priority Function

The round robin pointer operates nearly identically to the "any" pointer. It is also a two bit pointer which points most significantly to the highest priority UTurn and least significantly to the highest priority I/O Adapter within that UTurn. It also powers on in identically mirrored states, to guarantee unanimous arbitration decisions. However, the round robin pointer initializes to 2'b00 on the UTurn whose client_id[2] = 0 and for the UTurn whose client_id[2] = 1, the round robin pointer initialization value is 2'b10. Notice that this is switched from the initialization values of the"any" pointer, so that both UTurns come up with one of the two pointers giving them highest priority.

Once power is stable, the round robin counter will advance on every relevant evaluation phase. An evaluation phase is considered relevant by the round robin state machine if power is stable, the CLIENT_OP is either RET_ONLY or ONE_CYCLE, LONG_TRANS is false, and this cycle isn't a default drive cycle caused by an arbitration win on a RET_ONLY CLIENT_OP. If all of these conditions are met, then the pointer will exhibit the behavior illustrated below:



Note that if neither UTurn has arb'd, then neither will win, and therefore, the pointer will not advance. Conversely, if either UTurn has arb'd, then it must win, despite the possibility that the CLIENT_OP does not match the transaction type (such as a read or write transaction and a RET_ONLY CLIENT_OP or a many cycle transaction and a ONE_CYCLE CLIENT_OP). Since it would be illegal to issue the pending transaction under these circumstances, the winning UTurn has responsibility for forcing an idle cycle. On an arbitration win, the round robin pointer must advance, even if the transaction clashes with the CLIENT_OP and the UTurn is forced to drive idle cycles. In addition to changing state, if a UTurn has won,

then the signals ioa0_select, ioa1_select, ioa_idle, runway_drv, and second_cycle_default_win must be asserted to their proper values, as described in the previous section.

7.3.5. Stop Most Priority Function

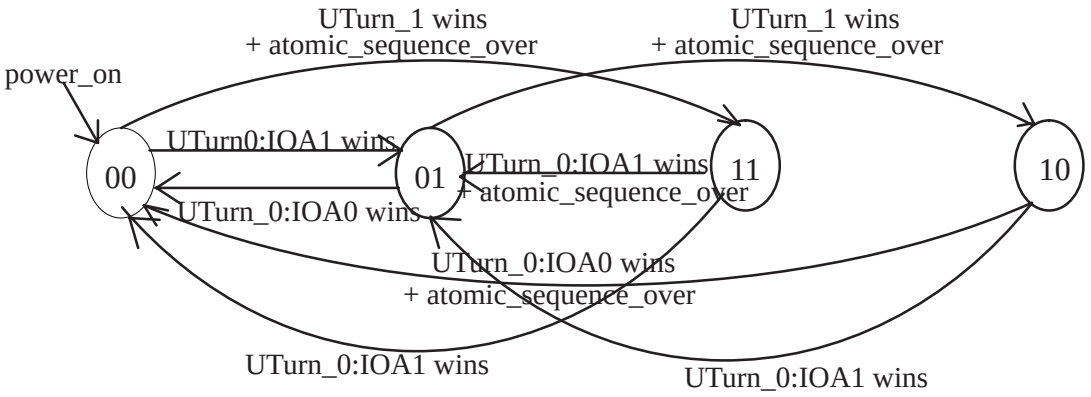
Stop Mosts occur when an I/O Adapter requires an atomic CLIENT_OP for the duration of a sequence of linked transactions. Whenever an I/O Adapter wants an atomic CLIENT_OP, it asserts its stop_most_out line. The arbiter then drives the UTurn's outgoing STOP_MOST_OUT signal. Each UTurn receives the STOP_MOST_OUT line of the other UTurn. When both UTurns assert their STOP_MOST_OUT lines simultaneously, the UTurn with the highest priority continuously wins the atomic CLIENT_OP until it has completed its entire sequence of atomically linked transactions. The UTurn with lower priority must back off from its stop most assertion, until the higher priority atomic sequence completes, and an any_trans client_op has occurred, ensuring forward progress. Whenever an atomic sequence completes, the stop most priority advances to the next I/O Adapter.

So, like the any and round robin pointers, the stop most pointer also points to the I/O Adapter with the highest priority. It, also, is a two bit pointer with the most significant bit indicating which UTurn and the least significant bit indicating which I/O Adapter within a UTurn. And again like the other pointers, the stop most pointer comes up in identically mirrored states within each UTurn to guarantee unanimous arbitration decisions.

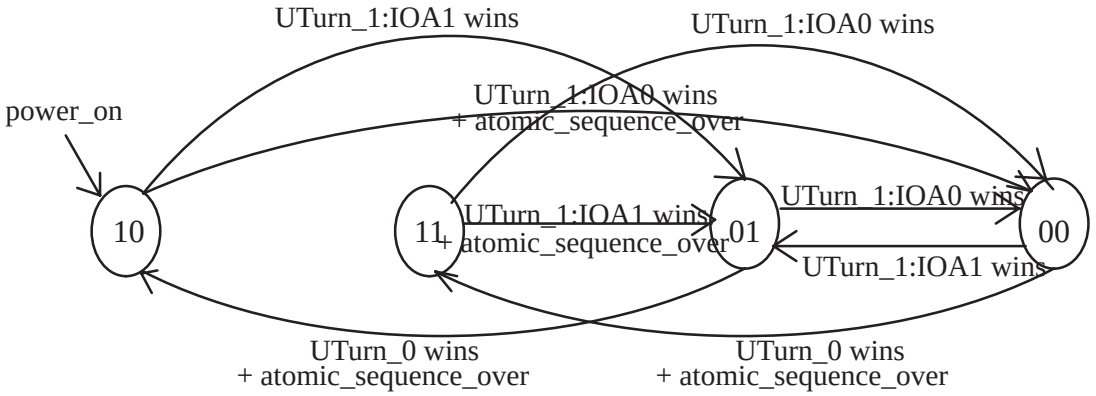
The stop most pointer differs from the any and round robin pointers in that it remains in the same state throughout a number of transactions, advancing only after the entire atomic sequence completes. The following diagram illustrates the behavior of the stop most pointer:

Note that the only conditions under which the pointer will advance is – (1) when the atomic sequence has just concluded (indicated by signal "atomic_sequence_over"), to rotate the priority or (2) when an atomic sequence is about to commence, to bring the pointer to the I/O Adapter which arb'd for the atomic sequence. Otherwise, throughout the assertion of an atomic CLIENT_OP, the stop most pointer stays constant. The detection of an atomic sequence conclusion is actually a bit of a corner case. The logic must detect that the atomic CLIENT_OP is still asserted, but neither UTurn is asserting its STOP_MOST output. Another noteworthy corner case, is when either UTurn is arb'ing after the completion of an atomic sequence, and the CLIENT_OP has not yet changed from atomic or when either UTurn is arb'ing for the atomic client_op prior to when the CLIENT_OP has changed to atomic. In these cases, the arb'ing UTurn is responsible for driving ioa_idle.

UTurn_0: UTurn with client_id[2] = 0



UTurn_1: UTurn with client_id[2] = 1



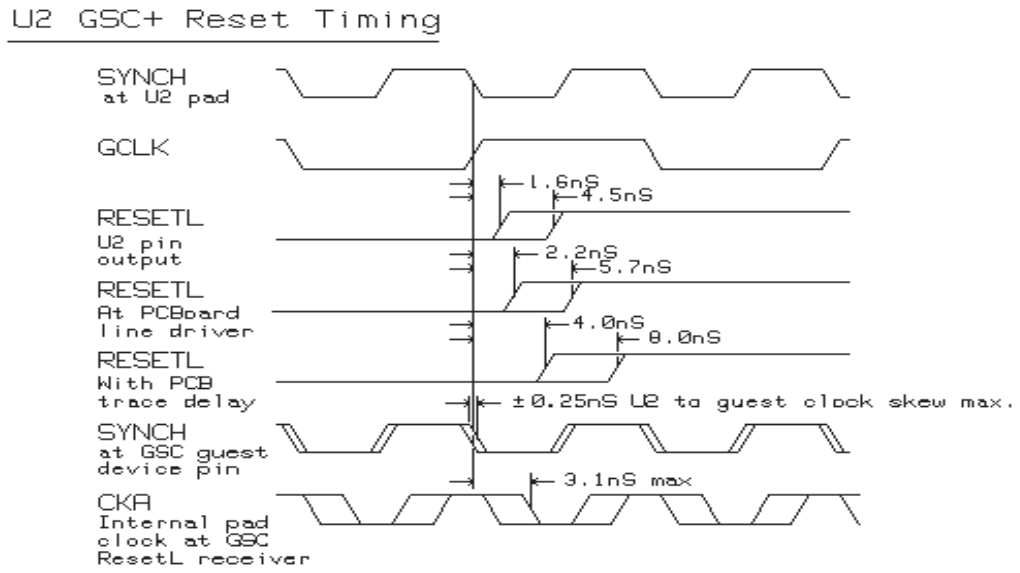
8. UTurn GSC+ Bus Ports

8.1. Overview

UTurn incorporates two identical 32–bit GSC+ bus ports. Each bus port consists of 3 basic elements: the clock circuitry, arbitration logic, and the data transfer protocol logic. The clocking, arbitration, and data transfer on UTurn's two GSC+ ports are independent. UTurn is the host on each of the GSC+ busses. UTurn will master transactions directed to GSC+ guests operating as slaves, and will be the slave in transactions mastered by GSC+ guests (guest to guest transfers are not supported).

8.2. Reset and clock synchronization

UTurn drives the RESETL line for each of its GSC+ bus ports. RESETL performs two functions: it provides a hard reset of GSC guests, and it is used by the GSC guests to synchronize their internal GCLKs to the SYNCH/L clocks (2X frequency) that are distributed externally. UTurn drives 2 RESETL lines, one for each GSC+ bus port. To meet the timing requirements for RESETL as given in the GSC specification, UTurn requires certain buffering and distribution delays to be added with external circuitry. The following timing diagram shows UTurn's RESETL line timing and the external circuit delays required for correct clock synchronization.



Hold Time Equation:

$$RESETL_{min@U2} + LINE\ DRIVER\ DELAY_{min} + PCB\ DELAY_{min} \geqslant U2\ TO\ GUEST\ CLOCK\ SKEW_{max} + RECEIVER\ CLOCK\ INSERTION\ DELAY_{max} + RECEIVER\ SETUP_{max}$$

Setup Time Equation:

$$RESETL_{max@U2} + LINE\ DRIVER\ DELAY_{max} + PCB\ DELAY_{max} + PART\ TO\ PART\ CLOCK\ SKEW_{max} \leqslant RECEIVER\ CLOCK\ INSERTION\ DELAY_{min} + SYNCH_{period}$$

The timing diagram shows minimum and maximum delays for a Kitty Hawk system; delays expected in Firehawk are similar. The delays shown for UTurn are independent of the systems in which UTurn is operating, and are a function of supply voltage, process skew, and temperature. The hold time equation and setup time equation mathematically constrain what is shown in the timing diagram. The delay from UTurn is the time from the crossing of SYNCH and SYNCL at UTurn's pins to the time that RESETL is driven on the PC board pin. The time shown for UTurn does not include device–to–device clock skew. Device–to–device clock skew is shown at the receiver end.

At higher GSC+ bus frequencies (or longer trace lengths), RESETL needs to incident wave switch the GSC guest's receivers. The RESETL pad driver in UTurn can successfully incident wave switch two guest receivers when the two RESETL traces fork at UTurn's PC board pin. When more than two GSC devices need to be reset from a single UTurn RESETL signal, a buffer/splitter is required. NOTE: A few GSC slave devices use the RESETL signal to tri–state their pad outputs during power on. For this reason it is important that the combination of UTurn and splitter provide a low RESETL signal during power on (i.e. before the first RESETL rising edge).

The device–to–device clock skew in conjunction with the insertion delay of the guest's clock to its RESETL pad cause a potential HOLD TIME violation to occur. This violation would cause the guest to synchronize on the SYNCH falling edge just preceding the correct one. To compensate for this effect, an artificial delay must be made to ensure that the guest synchronizes on the correct clock edge. In the timing diagram above this delay comes from the splitter/buffer circuit and delay on the PC board trace. This delay must be controlled (kept as small as possible) so that it does not cause a SETUP TIME violation at the operating frequency of interest.

It should be noted that the timing requirements for RESETL with respect to clocks must be met by the system designer. For GSC functions that are built directly into the system, timing relationships must meet the expectations of the components that are built into the system as part of the core functionality. For GSC functions that are added as expansion modules (via a connector), the system designer must ensure that the RESETL/clock relationship is correct at the expansion connector. It is then the responsibility of the add–in module designer to ensure that the RESETL/clock relationship is maintained correctly to the GSC interface logic residing on the add–in module.

8.3. GSC Arbitration

UTurn contains a central arbitration circuit for each of its GSC+ ports. Round robin arbitration for six external modules is supported with six request/grant pairs of pins that connect radially to the UTurn GSC+ port. (The number of external modules actually supported in a given system is dependent upon the limitations within the system implementation.) In addition to the external modules, the UTurn GSC+ master block competes for ownership of the GSC+ bus on behalf of one of two internal resources. One of these internal resources is the outbound command queue (OutQ), containing processor initiated read and write traffic for graphics and for general control/status transactions with I/O modules. The other internal resource that may cause UTurn to master a GSC+ transaction is the read return queue (RRQ), the channel for returning data to satisfy pended DMA read request transactions.

DMA read return traffic is the highest priority traffic on GSC+. Next in the priority order is outbound command queue traffic, including transactions to PDH space and to HPA registers either in UTurn or in GSC guests. Below this in the priority list come guest–mastered transactions. UTurn processes bus requests from external devices in round robin order. The order is fixed in hardware, and is not tied necessarily to GSC+ geographic address (OFFSET) assignment (geographic address assignments and Bus Request / Bus Grant pair assignments are made in a system–specific manner).

If a GSC+ guest asserts the RETRY line in response to a UTurn–mastered transaction, a forward progress algorithm is invoked in the GSC+ arbiter. The transaction to be retried is held at the head of the outbound command queue, preventing further OutQ traffic from proceeding until the retried transaction is successfully completed. Arbitration then proceeds in normal order. UTurn will retry its transaction after completion of either a DMA read return transaction sequence or a guest–mastered transaction sequence, or after a fixed time delay of 16 GSC gclk cycles.

UTurn implements a per–guest “pulsed grant” option. When enabled for a given guest, any bus grant issued to that guest will be asserted for only one GSC bus cycle. In normal operation, a bus grant is issued to a guest for as long as the guest continues to request ownership of the bus; this is a “level grant” mode. The pulsed grant mode allows for further host control of individual guest bus tenure by essentially telling the guest to relinquish bus ownership as soon as possible. Default bus grant mode for UTurn is the level grant mode. Pulsed grant mode is configurable on a per–guest basis by a field in the GSC+_CONFIG register.

UTurn’s GSC arbiter incorporates several specific forward progress mechanisms designed to ensure that guest devices are not prevented from obtaining GSC bus ownership indefinitely. Since OutQ traffic is prioritized above guest–mastered GSC traffic by the arbiter, and since theoretically the OutQ could always contain a valid entry (initiated by a processor) in a multiprocessor system, forward progress mechanisms must be invoked or guests may starve. After every DIO read transaction and after every PDH access (read or write), UTurn ignores the OutQ for a GSC bus cycle, allowing the bus to be granted to a requesting guest device. An optional forward progress mechanism, enabled amd configured by bits in the GSC+_CONFIG register, allows guest mastership after every 4, 8, 12, or 16 OutQ write entries processed. This forward progress screen is useful in cases where long write streams from a processor to a graphics device would otherwise starve out a guest attempting to perform DMA transactions. Late in the UTurn project, it was suggested that due to changes in the processor–to–frame buffer flow control scheme for future graphics hardware and software, a better implementation of the DMA forward progress mechanism would allow guest bus mastership after a given number of GSC clocks, not after a given number of OutQ entries processed. The UTurn team agrees that the clock counting solution is better given the architectural changes, but coordinating such a change in UTurn at a point late in the development cycle entailed more negative schedule and risk impact than could be justified by the expected benefit for the change. The OutQ entry count implementation is functional, and is the implementation plan.

8.4. Transaction Types

”DIO” transactions are transactions mastered by UTurn on the GSC+ bus. DIO transactions are ultimately originated by a processor on the Runway side of UTurn. ”DMA” transactions are transactions mastered by guests on the GSC+ bus. UTurn assumes that it is the slave for all DMA transactions. DMA transactions generally address memory space, although some guest–initiated single–word writes can be directed to a processor’s I/O address space on the Runway side of UTurn.

Each UTurn GSC+ port can master the following GSC+ transactions.
Connected ”DIO” reads of <= 1 or 2 words (1–4 or 8 bytes)
Pended (split) ”DIO” reads of <=1 or 2 words (1–4 or 8 bytes)
”DIO” writes of <=1 or 2 words (1–4 or 8 bytes)
Variable length ”DIO” writes of 1 to 8 words (4 to 32 bytes)
DMA read returns of 1, 2, 4, or 8 words (4, 8, 16 or 32 bytes)

Each UTurn GSC+ port can be the slave in the following GSC+ transactions.
Connected DMA reads of <=1, 2, 4, or 8 words (1–4, 8, 16, or 32 bytes), with or without lock (LSL)

asserted
Pended (split) DMA reads of 1, 2, 4, or 8 words (4, 8, 16 or 32 bytes)
DMA Writes of <=1, 2, 4, or 8 words (1–4, 8, 16, or 32 bytes)
DIO read returns of <=1 or 2 words (1–4 or 8 bytes))
Guest–mastered Clear transaction (DMA read with clear side–effect in memory)
Guest–mastered GSC Error transaction

8.5. Transaction Duration

Transactions mastered by UTurn on GSC+ are time limited. The connected transaction time limit is programmable by system software; default value is 16380 GSC bus cycles (approximately 512 us for a 32 MHz GSC implementation). Responsibility for GSC transaction length checking rests on UTurn; guests need not have transaction length checking capability. Transaction length checking follows the requirements of the GSC specification. Transactions mastered by guest devices are timed in a very coarse sense by a watchdog timer. GSC interfaces to devices such as EISA cards may assert LSL and monopolize GSC for indeterminate periods of time. This behavior is highly discouraged, since all system activity (including processor accesses to memory) will be blocked while LSL is asserted.

Pended DIO reads also have a pended transaction timeout associated with them. If the guest that pended a DIO read transaction does not respond with the data, using a corresponding DIO read return transaction, within the pended transaction timeout value, a timeout error will be logged and appropriate processor notification will occur. The pended DIO read transaction timeout value is software programmable, and defaults to 16380 GSC bus states (approximately 512 us for a 32 MHz GSC implementation).

8.6. GSC+ Master interface

8.6.1. Overview

The GSC+ master portion of the UTurn design takes transactions from UTurn's outbound queue or read returns from the read return queue and processes them appropriately. Entries in the read return queue are always bound for a GSC+ guest. Transactions in the outbound queue can be destined for a GSC+ guest, an internal (HPA) register, the PDH port, or the inbound queue ("U–turn" transactions). The GSC+ master block makes a determination as to the appropriate destination. If the transaction is a PDH space access, the necessary decoding and byte packing/unpacking is performed as part of the transfer. See the PDH port description for more details.

It is assumed that by the time a non–UTurn–HPA/non–PDH transaction reaches the GSC+ master in the outbound queue or the read return queue, the RUNWAY outbound block has already done appropriate address range comparisons to ensure that the transaction is indeed destined for a guest attached to the GSC+ port shared by the master block.

Read return queue entries contain a GSC+ guest ID, a transaction size, and a pointer to an on–chip RAM location that contains the actual data. Outbound queue entries include a GSC+ address, transaction size, and valid byte(s) indication. Reads also require a RUNWAY master ID and transaction ID field. The following transaction types may be found in the outbound queue:

"U–turn"	transaction	Destination: Inbound Queue
READ 1	(Single word read)	Destinations: PDH, Internal registers, or GSC+
READ 2	(Double word read)	Destinations: PDH, Internal registers, or GSC+

WRITE 1 (Single word write) Destinations: PDH, Internal registers, or GSC+
WRITE 2 (Double word write) Destinations: PDH or GSC+

If the destination for a particular queue entry is determined to be GSC+, the UTurn master block arbitrates for GSC+ and then performs the necessary transaction. In the case of a write to a GSC+ guest, UTurn sources address and transaction type information followed immediately by data. After acknowledgement by a guest, assuming no error occurs, the transaction is finished, and no further action is taken by UTurn. For DMA read returns to GSC+ guests, UTurn ”addresses” the guest explicitly by asserting the guest’s unique bus grant line along with a data read return signal. Data is then sent to the guest. Handling of reads mastered by UTurn is more complex, since reads can be connected, pended (split), or retried. The pend and retry cases are discussed later. For a connected read transaction, UTurn sources an address and transaction type and then waits for data while retaining bus ownership. The first datum returned from a guest is accompanied by the assertion of READY_L by the guest. As the data arrives, it is stored in an on–chip RAM location associated with an inbound queue entry that is built for each completed read. The inbound queue entry includes the RUNWAY master ID and transaction ID of the initiating RUNWAY module (copied over from the outbound queue), a transaction size and valid bytes indication. For each inbound queue entry, there is an implicit pointer into the on–chip RAM array containing the data associated with the queue entry.

8.6.2. Protocol implementation

8.6.2.1. Basic GSC protocol

At the core of UTurn’s GSC+ master interface, the base GSC protocol is implemented to support peak DIO transfer rates for single word and double word DIO write transactions. This is somewhat independent of design optimizations that enhance DMA performance. DMA concerns are discussed in the GSC+ slave interface section.

One specific transaction type note: for UTurn–mastered transactions to broadcast address space on GSC (should only be write transactions), UTurn will assert READYL itself during the data cycle (broadcasts are single word transactions). Guests that have registers in broadcast space must not assert READYL when slave to such broadcast transactions.

8.6.2.2. XQL prefetch hint

The GSC extended data request line, XQL, is used by UTurn as a deterministic prefetch indication for DMA read transactions issued by a GSC+ guest master. Assertion of XQL by the guest during the address cycle of a DMA read transaction indicates that, in addition to the data requested in the current transaction, the data for the next contiguous block of memory will be requested ”soon” by the same guest. Along with fetching the explicitly requested data, then, UTurn may elect to also prefetch the next contiguous chunk of data (where chunk size is the size of the explicitly requested transaction). When the prefetched data is actually requested by the guest, UTurn can supply the data from an on–chip prefetch buffer, eliminating the latency to main memory across RUNWAY. The prefetch is not speculative. When the guest asserts XQL, it is committing to fetch the data from UTurn. For further implementation details, see section {Prefetching in the IOA} in chapter {Inbound (GSC+–to–RUNWAY) Transaction/Data Flow}. Note that XQL is being implemented as documented in the GSC+ extensions document (ver 0.95 and later), in that UTurn will never actually extend a DMA read transaction by streaming more data to the guest than was explicitly requested. UTurn will also ignore XQL assertion during DMA write transactions.

8.6.2.3. GSC+ protocol extensions

The GSC+ protocol additions to GSC add extra performance capabilities required in systems where UTurn is the I/O bus bridge. Protocol additions include a true split read transaction protocol (terminology used for GSC+ is ”pended reads”), and a RETRY capability for deadlock avoidance and/or guest flow control.

8.6.2.3.1. Pended reads

The GSC+ pended read protocol implements a true split transaction capability. (GSC describes the deadlock resolution usage of the LSL line as a ”split”, but it is really more of a ”swap” of bus ownership between the host and a specific guest.) Implementation requires three external lines – a ”pend capable” line (PENDL), a pend acknowledge line (PACKL), and a data read return line (DRRL). PENDL and DRRL are driven by UTurn and received by all guests. PACKL is shared by all guests, and is received by UTurn. Reads initiated by either UTurn or by a GSC+ guest may be ”pended”.

Pended DMA reads, initiated by GSC+ guests, operate as follows. At the time UTurn issues a bus grant to a GSC+ guest, if it is capable of supporting a pended DMA read, it asserts the PENDL line, which is bussed to all GSC+ guests. If a GSC+ guest bus master wishes to perform a read transaction and supports the pended DMA read capability and senses PENDL asserted, it can elect to source only the address phase of the read transaction while asserting the PACKL line. The guest then gives up the bus, allowing other devices in turn to use the bus while the host fetches the requested data. When the data is available, UTurn’s GSC master state machine arbitrates for the bus and, after gaining control, asserts the DRRL line along with the Bus Grant (BGL) line connected to the requesting guest. The return data is transferred in subsequent bus cycles. DRRL and BGL are released during the last data cycle. The advantage of the pended DMA read cycle is that the requesting guest need not tie up bus bandwidth waiting for return data to become available. For a moderately busy system in which Runway is the processor/memory bus and DMA reads must go all the way to memory (no prefetch), the wait time to the first datum on GSC+ (the effective latency) can be twice as long as the time required to actually transfer the data.

Pended DIO reads (reads initiated by UTurn) operate as follows. UTurn makes a determination before each DIO transaction it sources whether or not a pended DIO read will be allowed. If a particular DIO read can be pended, UTurn will assert the PENDL line during the address cycle of the read transaction. If the guest that determines that it is the slave in the transaction supports the pended read capability and senses PENDL asserted, it can elect to assert PACKL in place of data / READYL. UTurn internally stores necessary state regarding the read (note that it does not know the identity of the guest), then terminates the read, allowing other transactions (including ones mastered by UTurn) to proceed. At some point, the guest that asserted PACKL will have retrieved the read data. To complete the read, the guest must arbitrate for and win mastership of the bus, then source a transaction with the type field encoded to indicate a read return transaction. Other than the type encoding, this transaction is just like a write transaction. UTurn decodes the type, associates the data with the earlier request, and returns the data to the requesting processor via the inbound queue. Since only one pended DIO read transaction at a time can be outstanding on GSC+, the ”address” presented by the guest on the AD[] lines during the address cycle of the read return is unnecessary and is currently ”reserved”. While a pended DIO read is outstanding on GSC+, UTurn will not assert PENDL while mastering a subsequent DIO read transaction. UTurn may assert ERRORL and log a hard error if it sees a read response transaction when there is not a pended DIO read request outstanding. As with pended DMA reads, the advantage of the pended DIO read is that the responding guest need not tie up GSC+ while fetching read data. For devices that have long read latencies, as might be expected for bus converters to slow busses like EISA or NIO, 40 or more bus cycles per read may be freed up for further data transfer. A specific example of this is a system where graphics, an EISA bus converter, and a DMA device sit on the same GSC+ bus. A read of an EISA register could be pended

by the EISA–to–GSC+ bus converter, and while waiting for the read response the bus is free for UTurn to complete writes to a graphics device or for a DMA device to continue its transfers. Quantification of the benefits of support of pended DIO reads is difficult, but it was felt to be needed. It is at the bottom of the required features list, and has the greatest chance of being unsupported in first silicon.

8.6.2.3.2. RETRY

One of the extensions to the GSC specification implemented by UTurn is a busy/retry capability. By asserting the RETRYL line, a GSC+ guest may indicate that it is unable to process a DIO transaction (read or write), expecting the GSC+ host (UTurn) to re–attempt the transaction at some time in the future. Forward progress guarantees are the responsibility of the guest device. That is, a guest implementing the RETRYL line must ensure that a given transaction will not be retried indefinitely. This prevents the possibility of deadlock (or ”livelock”) in the system.

Retry capability is required if pended DIO reads are to be supported with maximum concurrency. If a guest pends a DIO read, UTurn cannot in general determine the identity of the guest. To maximize bus usage, UTurn will continue to master DIO transactions after a DIO read has been pended, raising the possibility that a second DIO transaction will be directed to the guest that is working on the pended DIO read. The guest must now have a way to process this second DIO transaction without violating any ordering constraints and without pending the second transaction if it is a read (only one pended DIO read on the bus at a time). The guest may assert LSL to split the second DIO transaction, returning the DIO read data under split. Alternatively, the guest may elect to assert RETRYL if its GSCL input is high, causing UTurn to terminate the second DIO transaction and retry it at some later time.

The GSC+ RETRY capability has several uses. One use is to resolve a bus interlock (that is, deadlock) condition. This condition arises in I/O subsystems when shared busses on both sides of a module like a bus converter are simultaneously locked by devices attached to those busses, each of which must acquire ownership of the ”other” bus in order to complete some operation before relinquishing the shared bus resource it owns. If neither device can be ”backed off”, deadlock occurs because neither bus will ever be free. In systems with Runway being the processor/memory bus, the GSC+ host (UTurn) may be attempting a read or write transaction to a device on a bus like EISA through an appropriate bus converter, and at the same time a master device on the EISA bus may be attempting a read or write from/to memory. Since EISA devices cannot be ”backed off”, GSC+ must provide the back–off mechanism to avoid deadlock. The GSC+–to–EISA converter would have to initiate a deadlock resolution sequence on GSC+ in the middle of the UTurn–mastered transaction. In general, there are two ways for GSC+ guests to resolve a potential deadlock condition. One way, the GSC–defined mechanism, is for the guest to assert the LSL line in the middle of the UTurn–initiated GSC+ transaction to essentially swap ownership of GSC+ with UTurn, suspending UTurn’s transaction and allowing the guest to master one or more transactions in a memory–locked mode. After completion of these transfers, the guest releases LSL and the UTurn–mastered transaction is resumed and completed. Use of LSL has significant negative performance impact in systems incorporating UTurn, exposing all devices on all busses to a significant delay due to the time required for the guest to complete the necessary transactions. The guest will not likely be buffering and bursting efficiently when asserting LSL. Additionally, since LSL also requires memory lock, all modules in the system (including processors) are locked out of memory for the time required to complete the interlock resolution transactions. A second way to resolve deadlock conditions without LSL’s negative side effects is available to GSC+ modules – a retry capability. With this capability, if an interlock condition is encountered, the GSC+ guest asserts RETRYL in the middle of the UTurn–mastered transaction, causing UTurn to relinquish bus ownership (cease the present transaction) and allow other devices including the RETRYing guest to arbitrate for and be granted the bus. The guest need not

take over GSC+ right away, so it can buffer writes and burst efficiently, leaving the bus free for other guests to use in the meantime. Perhaps more importantly, memory is not necessarily locked during the deadlock resolution phase. Once the guest has finished the transaction(s) needed to resolve the bus interlock, a UTurn re–attempt of the earlier transaction will successfully conclude. During this sequence, the GSC+ bus and memory are not locked by the GSC+ guest.

A second possible reason for a guest to assert RETRY is as a flow control mechanism, to indicate ”not ready for data”. The example here is a graphics device that only accepts processor reads and writes (as opposed to using DMA). A processor may be able to generate transactions much faster than a graphics device may be able to process them. To avoid overrunning the graphics device, several alternatives may be employed. One of these is to allow the graphics device to assert RETRYL when its input FIFO is full. Some time later, the write transaction will be retried, and by that time the graphics device may be ready to accept it. In systems incorporating UTurn, the graphics device sitting on a GSC+ bus could assert RETRYL to UTurn, and UTurn would hold the transaction at the head of its outbound queue, retrying the transaction after some minimum delay. Other guests connected to GSC+ would be able to proceed with their own DMA or directed write (interrupt) transfers while UTurn waits to retry. UTurn’s outbound queue is blocked by this to–be–retried queue entry, so eventually UTurn’s outbound queue may be full. This will result in issuance of a STOP_IO on RUNWAY, which will in turn prevent all processors from issuing transactions to I/O space. This is a potentially ugly side effect.

A third use for the RETRYL line is as an alternate way to free GSC+ for other bus activity while a long–latency read operation is performed by a GSC+ guest. The EISA or NIO bus converter may assert RETRYL and initiate a fetch of the data. While the read data is being fetched from the EISA or NIO device, GSC+ is available for other guests to perform DMA transactions. (Note that processor–initiated graphics traffic will still have to wait for the read to complete successfully.) UTurn may retry the read several times before the data is actually available, but eventually the read will be satisfied. The opportunity for other guests to proceed with DMA in the interim results in improved overall system throughput for systems with significant DMA activity. Because graphics traffic gets backed up behind a busied transaction, however, the recommended way for GSC+ guests to hide long read latencies is to assert PACKL to initiate a pended (split) transaction sequence if possible.

A fourth necessary use for RETRY relates to overflow conditions when using pended DIO reads (as was noted above). Architecturally, unless unlimited pended transactions are supported (difficult to do) or unless a pended transaction prevents UTurn from initiating further DIO transactions (making pend look a lot like retry), RETRYL will always be needed when pended DIO reads are supported.

The following UTurn–specific RETRYL implementation notes may be of use to guest module designers.

- UTurn will retry a transaction after 16 GSC cycles or after a transaction sequence mastered by a single guest or after one or more DMA read return transactions, which ever occurs first. The retry will not begin until after a guest has relinquished bus ownership and UTurn has returned all queued pended DMA read data to guests.

- UTurn will discard a transaction held for retry if, while waiting to retry, UTurn logs a hard or fatal error in either the Runway clock domain or the GSC clock domain. If the discarded transaction is a read, PATH_ERROR is returned to the processor that initiated the read.

- A transaction held for retry blocks UTurn’s outbound command queue (OutQ). Even PDC accesses will not be completed as long as a retry is outstanding. Thus, it is extreme-

ly important that guests implementing RETRYL have a forward progress mechanism that prevents indefinite retry of a given transaction. PDC access must not be permanently blocked.

8.6.2.4. GSC1.5X and GSC2X extensions

To enable higher throughput for processor–mastered I/O writes (DIO writes), the GSC protocol has been extended to include a variable–length write transaction (writeV). UTurn can master writeV transactions on GSC as a result of write transactions in an IOA OutQ when appropriately configured. (Guest–mastered writeV transactions are not supported.) If certain conditions are met (detailed in a subsequent section), UTurn will combine multiple OutQ entries into a single GSC writeV transaction, an operation referred to as coalescing. "GSC1.5X" refers to support for the writeV transaction with data transferred at the normal GSC rate (one data word per GSC bus cycle). "GSC2X" refers to support for writeV transactions with data transferred at twice the normal GSC rate (2 data words per GSC bus cycle, one word per GSC GCLK edge).

8.6.2.4.1. Configuration of GSC1.5X and GSC2X capability

UTurn incorporates two registers that are used to enable GSC1.5X and GSC2X capability. These registers, GSC1.5X_CONFIG and GSC2X_CONFIG, are located in the GSC bus–specific register set at register offsets 8 and A (see architectural chapter for details). Each register is 32 bits long, and each register bit corresponds to an 8 MByte "chunk" of I/O address space on the GSC bus. If a particular bit of the GSCnX_CONFIG register is set, writeV transactions are enabled to that address space (if the bit is set in the GSC2X_CONFIG register, writeVs will be performed with data being transferred at the 2X rate). It is expected that GSC guests supporting the GSC1.5X and GSC2X capability will follow the capability reporting requirements set forth in the GSC2X specification, and that system software will correctly determine guest capability and appropriately configure UTurn to enable the appropriate GSCnX operation in UTurn.

8.6.2.4.2. WriteV protocol

The GSC2X specification contains details of the GSC1.5X and GSC2X transaction protocol. The essential ingredients of that protocol are detailed here. The reader is encouraged to consult the GSC2X specification for further details.

UTurn bus arbitration for a GSC writeV transaction is identical to that for any other UTurn–mastered transaction. Once UTurn has secured bus ownership, it masters a writeV address cycle. The address cycles is a full GSC bus cycle in duration. The value driven on the TYPE[0:3] bus is 4'b1111. The least significant 2 bits of address are driven to 0 for a writeV in 1.5X mode (normal data rate). For writeVs in 2X mode, the 2 address LSBs are driven to 2'b01.. There are no address alignment restrictions for writeVs (other than the state of the 2 LSBs as just noted). Data cycles immediately follow the address cycle. All data cycles are assumed to have all byte lanes valid. Bit 0 of the TYPE bus is used to indicate when the writeV transaction is complete. Specifically, TYPE[0] will be high until the last data cycle; during the last data cycle, TYPE[0] will be driven low. The remaining TYPE bits are driven to 0 by UTurn during data cycles. For data in 2X mode, TYPE[0] is driven with the same timing as data. If a writeV transaction in 2X mode has an odd number of data words, the remaining half GSC cycle is unused.

GSC writeV transactions must be acknowledged with READYL just as write1 and write2 transactions are. A guest may assert READYL in response to a writeV transaction as early as the first data cycle. Even in 2X mode, GSC control lines (with the exception of TYPE[0:3], as noted earlier) are driven for a full GSC GCLK cycle. A guest may delay READYL assertion as a flow–control mechanism, but since

UTurn will signal a transaction timeout even for a writeV transaction, guests must limit acknowledgement delays. A guest may not acknowledge a writeV transaction by asserting RETRYL. UTurn will ignore RETRYL when mastering a writeV transaction, so if a guest acknowledges a writeV with RETRYL, a GSC transaction timeout will result.

UTurn can master back–to–back writeV transactions, much as the “fast writes” case for write1s and write2s documented in the GSC specification. Guests supporting GSC1.5X and/or GSC2X should be able to process the address that occurs in the cycle following guest assertion of READYL in response to a writeV. (That address cycle may be for another writeV, or may be for any other UTurn–mastered transaction).

8.6.2.4.3. DIO write processing and coalescing rules

Once GSCnX capability is enabled via the GSCnX_CONFIG registers, any OutQ write entry (DIO write) whose address falls within an address space enabled for GSCnX capability will be performed as a writeV transaction on GSC. Single word writes will be performed as a writeV, even though there is no cycle savings even in 2X mode over a write1 transaction. Double word writes performed as writeV have the same cycle count as write2s in 1.5X mode, but in 2X mode only one GSC cycle is required for both data words.

If consecutive write transactions in the OutQ are addressed to consecutive I/O addresses within a GSCnX–enabled address space on GSC, and if the initial write transaction(s) in such sequences are 2–word writes, UTurn will coaslesce the write transactions into a single GSC writeV transaction up to 8 words in length. The initial address of the writeV has no alignment restrictions. An 8 word writeV can begin on an odd–word address. UTurn will break a writeV at a page boundary, however. UTurn is not planning to coalsece write sequences that begin with a single word write. A single–word write will be mastered as a writeV with one data cycle. Coalescing is only performed for DIO write transactions whose addresses fall within GSCnX–enabled I/O address space. DIO writes to all other I/O address ranges are performed as write1s and write2s on GSC.

8.6.2.4.4. Additional protocol restrictions

As noted in the GSC2X specification, writeV transactions can be split by a guest (LSL assertion, etc) after the last data cycle is transferred, but a guest cannot assert RETRYL in response to a writeV. Address parity error handling and data parity error signaling are assumed to be performed as for DIO write1s and write2s. In the 2X data case, ERRORL is still asserted for 1 full GSC cycle, even though 2 data words are driven per cycle. The ERRORL delay is still assumed to be 2 GSC cycles after the cycle in which a data parity error occurs.

8.6.3. Applications information

For graphics, heavy use of GSC1.5X and GSC2X modes is expected, taking advantage of the performance improvements made possible by coalescing consecutive writes and by transferring data at a 2X rate. In order to optimize the throughput to graphics devices, UTurn’s arbitration scheme assumes UTurn’s GSC+ master block to be the default GSC+ master, benefiting CPU–initiated transaction traffic (like graphics). Other bus requesters will be granted the bus in round–robin sequence as long as there are no DIO or read return transactions pending inside UTurn and as long as DMA forward progress mechanisms are not active.

Discussion regarding use of RETRYL for graphics data flow control has raised a few serious system performance concerns. As a result, graphics interfaces are not planning to implement RETRYL as a flow control mechanism.

8.7. GSC+ Slave Interface

8.7.1. Overview

UTurn's GSC+ slave interface translates guest–mastered transactions or other activity into appropriate entries in the internal inbound queue. UTurn's GSC+ slave interface proceses 5 types of guest–mastered transactions and 2 classes of hardware events.. The 5 transaction types are read, write, read return, clear, and error transactions. Hardware events include interrupts and errors..

8.7.2. Guest–Mastered Read transactions

Read transactions sourced by a GSC+ guest cause UTurn to create an appropriate read entry in the inbound queue. This entry has several fields that identify the requester and allow for a number of variations. See section {Inbound Queue} in chapter {Inbound (GSC+–to–RUNWAY) Transaction/Data Flow} for details on each inbound queue entry. In the case of sub–word reads, the guest is expected to ignore return bytes it is not interested in. UTurn will always supply 1, 2, 4, or 8 words of read data with all byte lanes containing valid data. No data shifting or other alignment functions are performed.

8.7.3. Guest–Mastered Write Transactions

A write transaction sourced by a GSC+ guest and bound for memory results in an inbound queue entry that is a subset of a read transaction entry (no CONNECTED or PREFETCH indications needed). See section {Inbound Queue} in chapter {Inbound (GSC+–to–RUNWAY) Transaction/Data Flow} for inbound queue entry details. All writes are assumed to be "drop and run" – that is, no acknowledgement of completion of the write in memory is sent to (or expected by) the GSC+ guest. All writes of 2 or fewer words (8 or fewer bytes) will be converted into read–modify–write memory update sequences on the RUNWAY side of UTurn.

8.7.4. DIO Read Return Transactions

Read return transactions sourced by a GSC+ guest provide the data in response to an earlier read request issued by UTurn. These transactions have dedicated GSC+ transaction type encodings, and are matched with request identification information to form an inbound queue entry. See section {Inbound Queue} in chapter {Inbound (GSC+–to–RUNWAY) Transaction/Data Flow} for inbound queue entry details.

8.7.5. Guest–Mastered Clear Transactions

A clear transaction mastered by a GSC+ guest will cause UTurn to return 4 words of data to the mastering guest, similar to a read4 transaction. The between a read4 and a clear transaction is primarily on the Runway side of UTurn. A clear transaction on the GSC bus causes UTurn to read a line from memory, clear a single word in the line (the word addressed by the clear transaction on GSC), and then write the modified cache line back to memory. The requested portion of the data read from memory is also returned to the guest (prior to clearing of the addressed word). In this way, GSC guest devices can participate in a semaphored communication protocol, supported by the GSC bus and UTurn hardware.

8.7.6. Guest–Mastered Error Transactions

A guest–mastered error transaction (TYPE field = 4'b1101 in guest–mastered address cycle) will cause UTurn to log a hard error on its GSC port. This transaction is used by the GSC–to–HPPB bus bridge,

GeckoBOA, to handle certain error cases where GeckoBOA encounters an error on the HPPB (AKA NIO) bus after processing the corresponding GSC transaction. UTurn ignores the address and data for such error transactions (parity is also ignored). UTurn handshakes a guest–mastered error transaction as if it were a write transaction. No inbound queue entry is created in response to an error transaction.

8.7.7. Hardware Events

External events may cause UTurn to generate a transaction entry in the inbound queue. These external events include interrupts and errors. The GSC definition includes both an interrupt line and an error line (INTERRUPTL and ERRORL).

8.7.7.1. INTERRUPTL Assertion

When UTurn senses an INTERRUPTL transition from the asserted to the negated state, it generates an inbound queue entry uniquely encoded to indicate that a GSC interrupt has occurred. This entry is translated into a Runway write_short transaction directed to a processor EIM register. UTurn's Runway inbound block uses information in the EIM_MONARCH_AND_GROUP register (UTurn Specific Register Set, offset 1) when performing this write transaction.

8.7.7.2. Error Cases

In some cases, a GSC+ bus error results in generation of an inbound queue entry. Specifically, when UTurn encounters a data parity error or a timeout on a DIO read transaction, it will log a soft error and will create an inbound queue entry indicating that a directed error transaction should be generated on Runway to inform the data requester of the error. For other GSC+ errors, including an error transaction type mastered by a GSC+ guest, a hard error is logged in the GSC+ port, preventing further guest–mastered transaction activity, but an error–type inbound queue entry is not created. Subsequent DIO reads of GSC+ guest registers result in a path error entry being created in the inbound queue, informing the requester that a hard error prevents fetching the requested data.

8.8. GSC+ Port Error Handling

Section {Error Handling} in chapter {Architectural Requirements} provides high level error detection and logging details for each IOA in UTurn. This section builds on that foundation, identifying a few GSC+ port specific implementation details.

8.8.1. Impact of Error Modes

When the GSC+ port of an IOA is in hard error mode (GSC IO_STATUS[he] = 1), or when the Runway side of an IOA is in hard or fatal error mode (RW IO_STATUS[fe] = 1 or RW IO_STATUS[he] = 1), the operation of the GSC+ port is significantly altered. Transactions on the GSC+ bus are no longer mastered by the IOA. Guests are not allowed to master transactions when an IOA has logged a hard or fatal error. Reads of GSC+ port internal registers are not allowed if the Runway side of an IOA was in hard or fatal error mode at the time the register read was requested. Writes to GSC+ port internal registers queued under a Runway hard or fatal error mode are discarded by the GSC+ port logic. The following table summarizes the impact of the various error modes on GSC+ port operation.

TABLE: Error mode impact on GSC+ port operation

Queue and entry type	Error Mode				
	Runway fatal error (RW IO_STATUS[fe] = 1)		Runway hard error (RW IO_STATUS[he] = 1)		GSC hard error (GSC IO_STATUS[he] = 1)
	Entry queued before fatal error occurred	Entry queued after fatal error occurred	Entry queued before hard error occurred	Entry queued after hard error occurred	
OutQ entries					
UTurn GSC port HPA reg. read	Perform read	Return path_error [1,2]	Perform read	Return path_error [1,2]	Perform read
UTurn GSC port HPA reg. write	Perform write	Discard write	Perform write	Discard write	Perform write
External GSC port read	Perform read on GSC	Return path_error [1,2]	Perform read on GSC	Return path_error [1,2]	Return path_error [1]
External GSC port write	Perform write on GSC	Discard write	Perform write on GSC	Discard write	Discard write
PDH port read	Perform read				
PDH port write	Perform write				
U–turn (RW HPA reg rd/wr)	Perform U–turn (create InQ entry)				
DMA test entry	Create InQ entry for test				
RRQ entries	Return data to GSC+ guest that issued read request				
Retry pending					
Retry trans = read	Return path_error [1]; discard read transaction				
Retry trans = write	Discard write transaction				
External bus requests from guests	No grants issued				
Transactions mastered by guests [3]					
Reads	Assert ERRORL; no InQ entry created				
Writes	Assert READYL but discard data (no InQ entry)				

Notes:
[1] When path_error is returned in these cases, no GSC+ transaction is performed.
[2] Errors that occur on the GSC+ side of an IOA do not affect the logging of a hard or fatal error on the

Runway side of the IOA. If a GSC+ port error occurs, access to GSC+ port HPA registers (including status and error logging registers) is still assured. If a Runway hard or fatal error is logged, GSC+ port status and error logging registers are not accessible.

[3] A Guest can master transactions if a hard or fatal error is encountered while the guest owns GSC+. Guest transactions mastered while an IOA is in hard or fatal error mode are not forwarded to Runway. It is strongly recommended that guests not master any GSC+ transactions after they detect an error condition related to their GSC+ interface.

8.8.2. Address Alignment Violations

Each of UTurn's GSC+ ports expects transactions mastered by GSC+ guests to be aligned to transaction size boundaries. Hence, 8 byte transactions have the least significant word address bit equal to zero; 16 byte transactions must have zeroes in the 2 least significant word address bits, and cache line sized transactions must be aligned to a memory cache line address boundary. Violation of this requirement will cause UTurn to assert ERRORL and log a hard error (GSC+ protocol error). UTurn will delay ERRORL assertion such that a guest will interpret the error as a timeout error.

8.8.3. Pended DMA Read Errors

Pended DMA read transactions have two components: a request component and a response component. The only error that is interesting during the response component is a data parity error. A guest that detects a data parity error during a DMA read response should assert ERRORL two GSC bus cycles after the cycle in which the error occurred. During the DMA read request, and in the time between the request and response components, many possible errors may occur. An address parity or alignment error may be detected by UTurn, or an error on the Runway side of UTurn may occur while the data is being fetched from memory. In any case, the mechanism for UTurn to inform the requesting guest that an error occurred is an errored DMA read return. The errored return consists of 2 or more notify cycles (DRRL and BGL[n] asserted simultaneously), followed by an error cycle in which DRRL, BGL[n], and ERRORL are all asserted. The error cycle may be followed by one or more "standard return" cycles in which only DRRL and BGL[n] are asserted; UTurn inserts 8 such cycles in its current design. After the error and standard return cycles, there may be a restore cycle, in which DRRL, BGL[n], ERRORL, and all other GSC control lines are negated, or there may be an address cycle for a subsequent UTurn–mastered transaction. Note that the AD[], TYPE[], and PARITY signals are meaningless for errored DMA read returns.

It should be noted that if an address cycle error occurs on a pended DMA read, UTurn will not assert ERRORL 2 bus cycles later. Rather, UTurn will drive an errored DMA read return transaction after some time.

8.9. General

8.9.1. Miscellaneous Notes

UTurn's GSC+ slave block has some interaction with UTurn's GSC+ arbiter in that it requires an identification of the current master for pended read return purposes as well as for error tracking. In addition, when RETRYL has been invoked on a UTurn–mastered transaction, UTurn's GSC+ master block may require some communication from the slave block and/or the arbiter to ensure that forward progress guarantees are preserved.

Transactions in which UTurn is a slave will be not be timed in the way that UTurn–mastered transactions are monitored. This is due primarily to the complexity required to keep separate timeout information

for each outstanding transaction that might be buffered inside UTurn. Runway does time its transactions, so timeouts between UTurn and memory will be detected.

8.9.2. Guest–to–Guest Transfers

Interest has been expressed in supporting guest to guest transfers using UTurn as an intermediary. UTurn does not support guest–to–guest transfers. UTurn assumes that it is the slave for all guest–initiated transactions. UTurn does not implement the necessary registers or address decode logic to decide if a transaction is destined for another guest on the same GSC bus segment. It has been suggested that UTurn might be able to act as a reflector for data, forwarding transactions from guest to guest if the address issued falls within an appropriate range of the I/O space. Implementation of this functionality is more than trivial. Guest to guest transfers are not supported in the UTurn design.

9. PDH Support in UTurn

UTurn includes internal logic and a dedicated external port for accessing processor dependent code (PDC) and other registers and functions in processor dependent hardware (PDH) address space. The PDH block inside UTurn is connected to only one of UTurn's IOAs (IOA0). The PDH block consists of several internal registers, an external memory–like port (including address, data, and control lines), and several miscellaneous functions such as a front panel interface, a real time clock, and a transfer of control input.

In Kitty Hawk global 40–bit physical address space, PDH resides in the address range from F0 F000 0000 – F0 F01F FFFF (2 MBytes). Inside UTurn, the 2 MByte PDH space is accessed using a 21–bit, byte–granularity address (the least significant 21 bits of the Runway address in Kitty Hawk). PDH space is partitioned roughly as shown in the following table. Note that addresses and ranges in this table identify the 21–bit PDH address that appears on the PDH address pins for external accesses. To get the equivalent Runway address, prepend the table address with F0 F0; e.g., off–chip fast PDH space is in the Runway address range F0 F0000000 through F0 F015FFFF.

Entity	Location	PDH Address / range	Total size	Timing	Comments
Off–chip "fast" memory components	External	000000 – 15FFFF	1408 KBytes	Fast	In this address range will be flash EPROM containing PDC and IODC, scratch RAM, and misc. individual registers
Real time clock	Internal	160000	4 Bytes	N.A.	Seconds since 1/1/70; battery backed
Semaphore reg.	Internal	160007 (Any address from 160004–160007 works)	4 bits	N.A.	Atomic read and clear LSB when read; next 3 bits are Runway master ID of semaphore owner
Access port data reg. (FPDATA)	Internal	160008	4 Bytes	N.A.	Latched into register, then shifted out using FPCLOCK
Unimplemented UTurn–internal PDH reg. space	Internal	16000C – 17FFFF	(128K – 12) bytes	N.A.	Writes to this range are discarded; reads return zeroes.
Off–chip "slow" memory components and interfaces	External	180000 – 1FFFFF	512 KBytes	Slow	In this address range are the EEPROM (containing stable storage information) and the LCD front panel interface

TABLE: PDH space partitioning

9.1. UTurn–internal PDH Registers

All resources noted as "internal" in the location column of the preceding table are actually implemented inside UTurn. Access to all internal registers should be with single word transactions. The specific PDH space registers implemented in UTurn are documented below.

9.1.1. Real time clock

The real time clock register is a readable and writeable counter that keeps track of wall clock time. The counter is powered by an external battery. The time value is incremented once each second.

The format of the real time clock register is as follows:

0 31 Time in seconds (UNIX assumes since January 1, 1970)

This register's value is maintained through power cycles and resets. Its value is changed only by writing to the register (or by removing the battery).

9.1.2. Semaphore register

The PDH semaphore register is a readable and writeable register containing 4 bits of interesting information. Reads of this register may have the side–effect of modifying the least significant bit of the register. The format of the register is as follows:

0 27 Not implemented (read as zeroes)	28 30 owner	31 semaphore

The *semaphore* bit operates as follows. At power–up, *semaphore* is initialized to a value of 1 (indicating semaphore available). *Semaphore* is set to 1 with any write to the semaphore register, regardless of the value written. Any read of the semaphore register returns the current value of *semaphore*. In addition, if *semaphore* is set when a read occurs, *semaphore* will be cleared, indicating that the semaphore is now owned, after the current value of the bit is returned to the reader.

The *owner* field contains the Runway master ID of the current owner of the PDH semaphore. If the semaphore is currently not owned, *owner* identifies the previous owner of the semaphore. At power–up, this field is initialized to zero. If the semaphore register is read and the semaphore bit is set, the master ID of the reader is stored in the *owner* field after the current value of the field is returned to the reader. Reading the semaphore register when the semaphore bit is cleared (semaphore owned) does not change the value of *owner*. Writing to the semaphore register does not affect the value of *owner*.

The following table summarizes the values of the *owner* and *semaphore* fields after various actions.

Action	*Owner* value	*Semaphore* value	Value returned on read
Power–up	000	1	N.A.
Write to semaphore reg.	unchanged	1	N.A.
Read from semaphore reg., semaphore bit set	RW Master ID of reader	0	{previous_owner_ID, 1}
Read from semaphore reg., semaphore bit clear	unchanged	unchanged (0)	{current_owner_ID, 0}
Otherwise....	unchanged	unchanged	N.A.

Important note: The initial UTurn parts contain a bug that causes the data returned on a read of the PDH semaphore register to be different than what is documented above. Specifically, on reads from the semaphore register when the semaphore bit is set (third row of table, last column), UTurn returns the value {current owner ID, 0} instead of {previous owner ID, 1}. Thus, for early UTurns, a processor reading the semaphore register must check to see that the *owner* value returned by UTurn matches its own ID in order to ensure that it acquired the semaphore.

9.1.3. Front panel data register

The front panel data register is a write–only register. Data to be shifted out on the UTurn's FPDATA pin is written to this register with the following format:

0 11	12 31
Unused	fp_data

According to an early version of the Kitty Hawk DC2–/DC2/DC3 System Board ERS, the fp_data field has its own sub–field definition. UTurn attaches no additional meaning to the bits in the fp_data register. Consult the System Board ERS or contact Ken Pomaranski for fp_data field details (kenp@hprpcd.rose.hp.com).

9.1.4. Unused UTurn–internal PDH space

Writes to unused UTurn–internal PDH space are discarded. Zeroes are returned on reads of unused UTurn–internal PDH space. No error is logged for accesses to unused UTurn–internal PDH space.

9.2. UTurn–external PDH Space

UTurn has an external data port to interface to the features identified as "external" in the location column of the PDH space partitioning table. The data port includes a 21 bit address (output only), an address valid line, an 8 bit data port (bidirectional, no parity), a write enable line, and an output enable line. External timing has fast and slow variants. The timing UTurn uses is based on address, not on external handshake. UTurn does not need to know how the external PDH address space is actually utilized. All PDH accesses that fall outside the internal register address range are simply sourced on the PDH port, with timing determined based on address value.

Details on the timing of the external PDH port, as well as further details regarding the external PDH address map, may currently be found in the DC2–/DC2/DC3 System Board ERS authored by Ken Pomaranski.

9.3. Other PDH Features

9.3.1. Front Panel Interface

For serial access front panel requirements, 2 pins are allocated – FPCLOCK and FPDATA. Values written to the front panel data register are shifted serially out of UTurn on the FPDATA pin as timed by FPCLOCK. Timing details regarding the FPCLOCK/FPDATA interface can be found in the DC2–/DC2/DC3 System Board ERS authored by Ken Pomaranski.

9.3.2. Transfer Of Control Accommodation

To accommodate transfers of control (TOCs), UTurn provides a TOC input pin. The received TOC input signal is forwarded directly to logic in the Runway domain of UTurn's IOA0 for generation of a command reset (write_short) transaction directed to the IO_COMMAND register of the processor indicated by the Runway TOC_MONARCH_CLIENT_ID register. See chapter {Architectural Requirements} for register details.

9.3.3. Real Time Clock Support

For real time clock support, an external clock signal pair is provided along with a separate supply rail input (for battery backed operation).

9.3.4. Test Features

The PDH port address and data pins (PADDR[0:20] and PDATA[0:7]) are used for chip test functions in addition to their normal PDH interface usage. Sets of signals internal to UTurn can be viewed on the PADDR bus when in a viewport mode (mode enabled by setting the vpt bit (bit 31) in the Runway TEST_INFO/CONFIG register). The specific set of signals to be viewed is selected by driving the PDATA bus from an external test source, generally a special test jig. A more detailed description of PDH port test mode usage can be found in the testability chapter of this ERS.

10. UTurn Testability

All testability features are documented in this chapter, including internal logic testing, viewport description and PDC self test.

10.1. Internal logic testing

In general, the testability features of UTurn follows the guidelines outlined in the ”Kitty Hawk ASIC DFT Design Rules”. Any additions or differences from those design rules are discussed below.

Boundary Scan is implemented using the IEEE 1149.1 protocol. A BSDL description of the boundary scan implementation is available.

The outbound queue (OutQ), inbound queue (InQ), read return RAM (RRR), and inbound RAM (InRAM) are all tested using built–in self test (BIST). In association with the BIST, there is a capability to bypass data around the BIST–able structures for test purposes. This combination of features is intended to provided for rapid go/no go testing.

Full internal Scan is implemented for all blocks not served by BIST. The scan flip–flops include double–strobe capability. This feature allows loading of master and slave stages with different values. The at–speed functionality of selected paths can therefore be checked.

The capability exists to use internal scan to apply both address and data to the BIST–able structures. The output data is also accessible via scan. Special provisions are also included to allow the a CPU access to TLB RAM output data for diagnostic purposes.

10.2. UTurn TAP controller

UTurn uses a TAP (test access port) controller. It implements all required 1149.1 JTAG features and a number of additional features that are used in conjuction with chip testing. The TAP controller is leveraged from the design used by the Kitty Hawk memory ASICs.

Please refer to ”Kitty Hawk Memory Sub–System TAP/SAP Design” for a detailed description of the TAP controller. The UTurn TAP controller contains a few modifications relative to the Kitty Hawk memory TAP controller. The following information includes those additions and changes:

10.2.1. UTurn TAP Controller Instructions and Behavior

UTurn implements the following TAP instructions:

Name	Opcode (Hex)	DR selected	capture data
extest	00	boundary	pad
bypass	FF	bypass	’0’
sample/preload	20	boundary	pad
intest	01	boundary	(note 5)
chip test	88	(note 3)	(note 5)
scan_internal	BE	(note 3)	(note 6)
dr00_scan	A0	(note 4)	(note 6)
dr01_scan	A1	(note 4)	(note 6)
dr02_scan	A2	(note 4)	(note 6)
dr03_scan	A3	(note 4)	(note 6)
dr04_scan	A4	(note 4)	(note 6)
dr05_scan	A5	(note 4)	(note 6)
dr06_scan	A6	(note 4)	(note 6)
isample	40	bypass	’0’
esample	41	bypass	’0’
hi_z	60	bypass	’0’
drive_inhibit	61	bypass	’0’
drive_enable	62	bypass	’0’
select_mode	BC	mode	(note 7)
set_mode_bit	DC	mode	(note 7)
clr_mode_bit	CC	mode	(note 7)
idcode	02	idcode	(note 8)

NOTES:

1) Instruction register length is 8 bits.

2) Mode register length is 12 bits.

3) The DR selected for the CHIPTEST and INTERNAL_SCAN instructions depends on the state of the PS (parallel scan enable) bit of the mode register. When PS=0 the Boundary scan register is selected for the DR of the CHIPTEST instruction. When PS=0 the Internal scan register is selected as the DR for the INTERNAL_SCAN instruction. PS=1 should NOT be used during board test/scan for either the CHIPTEST or INTERNAL_SCAN instructions.

4) The DR selected for the DRxx_SCAN instructions also depends on the state of the PS bit of the mode register. When PS=0 a block level internal scan path is selected as follows:

Name	Scan Path	
DR00_SCAN	DR0	
DR01_SCAN	DR1	
DR02_SCAN	DR2	
DR03_SCAN	DR3	
DR04_SCAN	DR4	
DR05_SCAN	DR5	
DR06_SCAN	DR6	PS=1 should NOT be used during board test/scan DRxx_SCAN instructions. If PS=1 the individual internal chains are ported through auxiliary scan–in and scan–out ports.

The PDATA[0:6] are used for scan–in.
The PADDR[0:6] pins are used for scan–out.
The assignments are as follows:

Scan Path	SCAN IN	SCAN OUT
DR0	PDATA[0]	PADDR[0]
DR1	PDATA[1]	PADDR[1]
DR2	PDATA[2]	PADDR[2]
DR3	PDATA[3]	PADDR[3]
DR4	PDATA[4]	PADDR[4]
DR5	PDATA[5]	PADDR[5]
DR6	PDATA[6]	PADDR[6]

5) The INTEST and CHIPTEST instructions capture the state of the system outputs when a system clock is given. The state of the system drive enables are not captured.

6) If PS=1, then the Boundary scan register is selected between TDI and TDO as the DR and the PAD is captured.

7) The parallel–update stage of the Mode register is captured.

8) The 32 bit IDCODE for UTurn is: ”0001” & — 4 bit version (This code for rev. B)
”0001010010101000” & — 16 bit part number
”00001010100” & — 11 bit manufacturer
”1” ; — mandatory LSB

9) All other/undefined opcodes produce the same effect as the BYPASS instruction.

Notes regarding the five system clock pairs:

The ”H” version of the clock corresponds to the internal clock as it is applied to the flops. The only exception is that when TestClockMode is set, then the internal GCLKs correspond to their respective ”L” version clocks. That is, SYNC_K0_L and SYNC_K1_L. The internal clocks should all be kept low during scan.

Bit definitions for the mode register:

DBL_STRB_1 mode_bits[11]

ML_TCK_B_1 mode_bits[10]

Arm_JPAD mode_bits[9] – During Chiptest operation, force PADNIO pads to capture signals from core logic

unused mode_bits[8]

force_hld_fets mode_bits[7] – Forces keepers on GSC pads to drive

gclk_DSBL mode_bits[6] – Disable core gclk. The GSC clocks to the pads are still active. Used to make GSC pads appear transparent during ATG testing.

BIST_EN mode_bits[5]

P_SCAN mode_bits[4]

FLUSH mode_bits[3] – Only works on internal scan chains

PMASK mode_bits[2]

IDDQ_EN mode_bits[1] – Used by the ”J2PADPNIO” pads during IDDQ testing.

TEST_MODE mode_bits[0] – Places GSC clock generator in divide–by–1 mode. Apply at least three clocks after setting or clearing this bit. Note: Bit[0] is closest to TDO.

10.3. Initialization of UTurn for JTAG operation

The first step is to reset the UTurn’s core logic and the TAP controller. At power on of the board the system reset signal ”ARESETL” and the test reset signal ”TRST” must be low. The ”TRST” signal initializes the TAP controller. The ”TRST” is an asynchronous signal, and only needs to be held low for a minimum of 100ns after power is up, to put the TAP controller in its test–logic–reset state. The ”ARESETL” input can either be kept low or allowed to rise after a minimum of 100ns. Application of clocks during the power up sequence is optional.

The purpose of the next step is to prevent the chip core logic from driving the pads and potentially causing external drive fights during opcodes when the core logic has control of the pads.

This tristating of the pads is accomplished by executing the DRIVE_INHIBIT opcode. This opcode sets a flip–flop which prevents the system logic in the chip from driving the pins. (Note that the EXTEST instruction still has the ability to drive the pads while the DRIVE_INHIBIT flip–flop is set.) If this option is to be used and the clocks are not halted then it is NECESSARY to set the ”test–clock–mode ” bit of the mode register before executing the DRIVE_INHIBIT.

The reason is as follows: UTurn uses the Kitty Hawk type inhibit circuit. For more information please refer to ”Proposal for Preventing Bus Clashes and System Reset” by Bulent Dervisoglu, 8/10/92. If drive inhibit has been set and the DRIVE_INHIBIT opcode is not currently in the instruction register, any rising edge of the system clock while the ”RESETL” pin is high will clear the Inhibit Latch. Chip system logic would then be able to drive the pads resulting in DRIVE FIGHTS. The setting of ”test–clock–mode” prevents the system clock from clearing the Inhibit Latch.

Specifically, the sequence looks like this:

A. Load the SELECT_MODE opcode; This selects the 12 bit mode register for the subsequent DR–scan;

B. Scan in the value 001H into the mode register; This sets ”test–clock–mode”;
(Note that the LSB is the first bit applied;)

C. Load the DRIVE_INHIBIT opcode;

It is recommended that the above sequence be used. It provides the highest degree of protection against drive fights caused by errors in the boundary scan test program whiich give control of the pads back to the chip functional logic at an inappropriate time.

If it is desired to re–enable the pads there are two options. The first is to use the DRIVE_ENABLE opcode. The second is to reset UTurn as described in the power up sequence above.

Following are the relevant UTurn opcodes:

DRIVE_INHIBIT 01100001

DRIVE_ENABLE 01100010

SELECT MODE 10111100

HIGH–Z 01100000

10.4. Viewport Description

UTurn has the ability to mux out many internal signals to the PDC Address pins through a combinational path. This allows internal signals to be viewed continuously while the system is running. This can be very useful for debug purposes. When a PDC access is in progress the PDC Address line revert to their real function – being the PDC address. The PDC DATA are used as enables and selects to select which signals are being ”view”ed.

The internal node viewing feature is only enabled if there is no valid address on the PADDR[] pins (that is, if pdh_addr_drv is high) and if either the MSB of the PDATA[] bus is being driven to a 1 and the view_enb bit is set in the Runway TEST_INFO register. Stated in an opposite sense, an actual address is driven on the PADDR[] pins as follows:

```
wire #1 drive_pdh_addr = pdh_addr_drv | ~pdataI[0] | ~reg_view_enb;
/* View port is enabled if pdh_addr_drv=0 AND pdataI[0] = 1 AND reg_view_enb=1  */
/*  View logic signal group selection logic  */
/*      Decode assignments for pdataI[1:2] when used as top level view select:
```

```
      10 – arb
      11 – arb
      00 – IOA0
      01 – IOA1
   case (pdataI[1:2])                // synopsys full_case
     2'b00:    paddrO = ioa0_view;
     2'b01:    paddrO = ioa1_view;
     2'b10:    paddrO[0:20] = {arb_view[0:5], ioa0_view[6:20]};
     2'b11:    paddrO[0:20] = {arb_view[0:5], ioa1_view[6:20]};
     default:  paddrO = ioa0_view;
   endcase
wire    [0:1]   #1      blk_view_sel = pdataI[3:4];
/*      Decode assignments for pdataI[3:4] :
   arb
      xx – blk_view_sel
   ioa
      00 – inbnd
      01 – outbnd
      10 – synchro
      11 – gsc_view

wire    [0:2]   #1      int_view_sel = pdataI[5:7];

/*      Decode  assignments for int_view_sel are made within the individual  lower–level blocks.   The mapping  of
signals for specific values of int_view_sel is shown below.    */



/**************************UTurn_arb****** arb_view[0:5]******************/
/*  blk_view_sel is a don't care – not looked at*/
/*  pdataI[0:4] = 5'b11xxx  */
case (int_view_sel)
     3'b000:  begin
               arb_view[0] = Rstart_arb0_state;
               arb_view[1] = Rtrans_len_in0_state[0];
               arb_view[2] = Rtrans_len_in0_state[1];
               arb_view[3] = Rtrans_len_in0_state[2];
               arb_view[4] = Rdata_return_in0_state;
               arb_view[5] = stop_most_out0_state;
              end
     3'b001:  begin
               arb_view[0] = Rstart_arb1_state;
               arb_view[1] = Rtrans_len_in1_state[0];
               arb_view[2] = Rtrans_len_in1_state[1];
               arb_view[3] = Rtrans_len_in1_state[2];
               arb_view[4] = Rdata_return_in1_state;
               arb_view[5] = stop_most_out1_state;
```

```
                end
3'b010:  begin
          arb_view[0] = ioa0_arb_ckrw;
          arb_view[1] = ioa0_ret_ckrw;
          arb_view[2] = ioa0_len_ckrw;
          arb_view[3] = ioa0_stop_ckrw;
          arb_view[4] = back_off_from_arb0;
          arb_view[5] = atomic_back_off0;
         end
3'b011:  begin
          arb_view[0] = ioa1_arb_ckrw;
          arb_view[1] = ioa1_ret_ckrw;
          arb_view[2] = ioa1_len_ckrw;
          arb_view[3] = ioa1_stop_ckrw;
          arb_view[4] = back_off_from_arb1;
          arb_view[5] = atomic_back_off1;
         end
3'b100:  begin
          arb_view[0] = rr_ptr[0];
          arb_view[1] = rr_ptr[1];
          arb_view[2] = any_ptr[0];
          arb_view[3] = any_ptr[1];
          arb_view[4] = stop_most_ptr[0];
          arb_view[5] = stop_most_ptr[1];
         end
3'b101:  begin
          arb_view[0] = ioa0_select_prev;
          arb_view[1] = ioa1_select_prev;
          arb_view[2] = runway_drv_prev;
          arb_view[3] = ioa_idle_prev;
          arb_view[4] = ioa_deselect_prev;
          arb_view[5] = second_cycle_default_win_last;
         end
3'b110:  begin
          arb_view[0] = reload_trans_ctr0;
          arb_view[1] = reg_pntr0[2];
          arb_view[2] = Rinterface_ready0;
          arb_view[3] = reload_trans_ctr1;
          arb_view[4] = reg_pntr1[2];
          arb_view[5] = Rinterface_ready1;
         end
3'b111:  begin
          arb_view[0] = UTurn_arb_out_twice;
          arb_view[1] = UTurn_arb_in_twice;
          arb_view[2] = any_back_off;
          arb_view[3] = atomic_sequence_over;
```

```
            arb_view[4] = UTurn_wins_any;
            arb_view[5] = clr_any_ctr;
           end
   endcase


/*************************ioa****** ioa_view[0:20]*******************/
      case(blk_view_sel)
       //synopsys parallel_case full_case
       2'b00:    /* select inbnd */
             begin
             in_n_out_view_sel = 1;  /* this selects inbnd instead of outb*/
             ioa_view = r2_view;
             end
       2'b01:    /* select outbnd */
             begin
             ioa_view = r2_view;
             end
       2'b10:    /* select synchro */
             begin
             ioa_view = synchro_view;
             end
       2'b11:    /* select gsc */
             begin
             ioa_view = gsc_view;
             end      endcase


/*************************outbnd_view****** ioa_view[0:20]*******************/
      /*  pdataI[0:4] = 5'b10001(IOA0) or 5'b10101 (IOA1) */
casex (int_view_sel)
       //synopsys parallel_case full_case
       3'b000:   /* select pool */
             begin
             outb_view = pool_view;
             assign #1 pool_view[0:20] = {reg_pool_entry0[`pool_inuse],
                 reg_pool_entry1[`pool_inuse], // pool_inuse = 48
                 reg_pool_entry2[`pool_inuse],
                 reg_pool_entry3[`pool_inuse],
                 reg_pool_entry4[`pool_inuse],
                 reg_pool_entry5[`pool_inuse],
                 reg_pool_entry6[`pool_inuse],
                 reg_pool_entry7[`pool_inuse],
                 reg_pool_entry0[`pool_pref],  // pool_pref = 47
                 reg_pool_entry1[`pool_pref],
                 reg_pool_entry2[`pool_pref],
```

```
            reg_pool_entry3[‘pool_pref],
            reg_pool_entry4[‘pool_pref],
            timeout,
            ld_RRQ_or_conn_pref,
            pref_rdy_4_rrq,
            pref_rdy_4_conn,
            entry_4_rtn[0:2],
            pref_discard};
        end
3’b001:   /* select outctrl */
        begin
        outb_view = outbctrl_view;
        assign #1       outbctrl_view[0:20] =
            {present_state[0:2],
            remember_trans[0:2],
            ld_RRQ,
            ld_RRR,
            ld_OutQ,
            ld_hpareg,
            R_conn_rtn_valid,
            R_ConnRRRAddr,
            R_ConnRRRError,
            r_ok_to_start,
            l_pool_match,
            r_pool_match,
            (r_ad_parityok & r_ctrl_parityok),
            (l_ad_parityok & l_ctrl_parityok),
            sv_trans_dest[0],
            sv_trans_dest[2],
            sv_trans_dest[4]};
        end
3’b010:   /* select data_store_h and ld signals to see when valid */
        begin
        outb_view = {ld_OutQ,ld_RRR,data_store_h[0:18]};
        end
3’b011:   /* select rest of data_store_h */
        begin
        outb_view = {ld_OutQ,ld_RRR,data_store_h[0:11],
                                    data_store_h[19:25]};
        end
3’b11x:   /* select outbccc and RRQ */
        begin
        outb_view = {ld_RRQ,RRQ[0:10],outbccc_view[0:8]};
        assign #1 outbccc_view[0:8] = { num_ok_resp_after[1:4],
                                        delay_till_Npriv, left_just_went_private,
                                        we_have_private, l_delay_ccc, r_delay_ccc};
```

```
            end
        3'b10x:   /* select error stuff */
            begin
            outb_view = {
                  error_enb,
                  timeout,
                  tlb_miss_err,
                  io_address_err,
                  (p_R_ERR_mode_phase | l_R_ERR_mode_phase),
                  l_We_MAS,
                  r_We_MAS,
                  (l_R_ERR_ctrl_par | r_R_ERR_ctrl_par),
                  (l_R_ERR_adr_par | r_R_ERR_adr_par),
                  (l_R_ERR_data_par | r_R_ERR_data_par),
                  (l_R_ERR_broad | r_R_ERR_broad),
                 (p_R_ERR_ill_resp | l_R_ERR_ill_resp | r_R_ERR_ill_resp),
                  (l_R_ERR_path | r_R_ERR_path),
                  (l_R_ERR_unexp_resp | r_R_ERR_unexp_resp),
                  R_ERR_addr_valid,
                  l_R_data_from_memory,
                  r_R_data_from_memory,
                  (p_R_ERR_pool_info_valid | l_R_ERR_pool_info_valid),
                  Rio_status_se,
                  hpa_reg_he,
                  hpa_reg_fe};
            end
        default:          /* select outbccc and RRQ */
            begin
            outb_view = {ld_RRQ,RRQ[0:10],outbccc_view[0:8]};
            end
    endcase


/*************************inbnd_view************************/
    /*  pdataI[0:4] = 5'b10000(IOA0) or 5'b10100 (IOA1) */
    casex (int_view_sel)   // synopsys parallel_case
       3'b000: begin          // most basic debug info
             inbnd_view = { inQ_valid,
                        R2interface_ready,
                        tlb_hit,
                        page_type,           // 2 bits of page type
                        cache_hit,
                        dont_start_rp,
                        write_back_start,
                        pool_full,
                        pool_write,
```

```
                transid_from_pool,
                inQ_done,
                Rstart_arb,
                cache_state,          // 2 bits of cache state
                inQ_nowstate};        // 6 bits of main state
        end
3'bx01: begin          // inQout signals
        inbnd_view = { inQ_valid,
                inQ_done,
                3'b000,               // 3 unused bits
                inQout['entry],       // entry = 0
                3'b000,               // 3 unused bits
                inQout['force_timeout],//force_timeout = 10
                inQout['HPA_read],    // HPA_read = 11
                inQout['TLB_cmd],     // TLB_cmd = 12
                inQout['RTN_zeros],   // RTN_zeros = 13
                inQout['Ktype],       // Ktype = 15:18
                        // 4 bits of GSC or inbnd type
                inQout['Kconnected],  // Kconnected = 49
                inQout['Klock],       // Klock = 50
                inQout['Knext],       // Knext = 51
                inQout['diagnostic]}; // diagnostic = 52
        end
 3'bx10: begin          // tlb and cache signals
        inbnd_view = { load_second_cache,
                cache_status_private,
                cash_read,
                clear_cache,
                cache_write_1,
                cache_write_2,
                bist_rm_obsv_count,
                bad_pool_entry,
                load_first_tlb,
                load_second_tlb,
                tlb_miss_read,
                tlb_purge,
                tlb_RAM_r_w,          // actual RAM write signal
                tlb_addr};            // 8 bits of tlb address
        end
3'bx11: begin          // other inputs (toc, interpt, prefetch ....)
        inbnd_view = { send_toc,
                send_broad_err,
                broad_error,
                send_pfail,
                io_address_detect,
                io_control_hv_mode,   // 2 bits of mode type
```

```
                        stop_most_out,
                        pdir_read_done,
                        prefetch_update,
                        prefetch_read,
                        prefetch_inc,
                        prefetch_overflow,
                        prefetch_match,
                        prefetch_enable,
                        Raddress_sel,        // 3 bits of address sel
                        Rsecond_sel,         // 2 bits of second sel
                        inbRAMaddr_lsb};
            end         endcase
/**********************************gsc_view*****************************/
    /*  pdataI[0:4] = 5'b10011(IOA0) or 5'b10111 (IOA1) */
 case (pdataI[5:7])           // synopsys full_case
     3'b000:       gsc_view = {    view_garb[0:2],
                        view_gmaster[0:8],
                        view_gslave[0:4],
                        k_reset,
                        k_resetO,
                        k_reset_to_RW,
                        powon_reset };
      3'b001:       gsc_view = view_garb;
            wire    [0:20]  #1      view_garb = {   arb_state[0:3],
                                DMAPendActive[0:5],
                                Req[0:5],
                                GuestGrant,
                                GuestActive,
                                gGuestID[0:2]
                              } ;
     3'b010:       gsc_view = view_ginmux;
                  assign #1       view_ginmux = { 1'b0,
                        staged_adI[24:29],
                        StagedDataWr[0:1],
                        staged_typeI[0:3],
                        staged_data[24:31] };
     3'b011:       gsc_view = view_gmisc;
                wire    [0:20]  #1      view_gmisc = {  gError,
                                gInterrupt,
                                gLockSplit_slow,
                                gReady,
                                gRetry,
                                k_readyLO,
                                k_reset,
                                k_resetO,
                                k_reset_to_RW,
```

```
                    powon_reset,
                    pdataI[0],
                    ResetCnt[0:3],  // MSBs of ResetCnt
                    ResetCnt[14:19] // LSBs of ResetCnt
                   } ;
3'b100:      gsc_view = view_gmaster;
           wire    [0:20]  #1      view_gmaster = {master_state[0:8],
                    PossDIOTrans,
                    GSCAddrCycle,
                    gmAdDrv,
                    k_addvLO,
                    k_type_drv,
                    LeftWdOnlyBuf,
                    RtWdOnlyBuf,
                    kmbg,
                    k_RRQ_valid,
                    k_OutQ_valid,
                    AnyErrMode,
                    1'b0
                    };
3'b101:      gsc_view = view_goutdecode;
             wire [0:20] #1  view_goutdecode = {     gIOFlexWr,
                    gIODCDataRd,
                    gIOStatRd,
                    gIOCntrlRd,
                    gIOErrRespRd,
                    gIOErrInfoRd,
                    gIOErrReqRd,
                    gRAMTestRd,
                    gTransTORd,
                    gPendTORd,
                    gConfigRd,
                    gWatchdogTORd,
                    gPerfMask0Rd,
                    gPerfMask1Rd,
                    gPerfComp0Rd,
                    gPerfComp1Rd,
                    gPerfCount0Rd,
                    gPerfCount1Rd,
                    gPerfConfigRd,
                    IntrnlReg,
                    DoWrite
                    };
  3'b110:      gsc_view = view_gslave;
             assign #1 view_gslave = { k_InQ_load,
                 slave_state[0:4],
```

```
                    ksRRRWdAddr[0:2],
                    ksDIORdRtn,
                    CkDMAWrDPar,
                    con_rtn,
                    connected,
                    crrv,
                    ready_wait,
                    BadDIORdRtn,
                    BadDMAWr,
                    DIOPendDone,
                    gsLoadDirErr,
                    ksreadyLO,
                    ksAdDrv };
        3'b111:     gsc_view = pdh_view;
                wire   [0:20]  #1     pdh_view = {pdh_state[0:4],     // Main state mach.
                    ClkCnt[0:5],       // GCLK Cycle ct.
                    OutQBuf[10],       // OutQ PDH bit
                    OutQBuf[12],       // OutQ READ/NWR bit
                    OutQBuf[16:23] };   // OutQ Runway Byte Enb
         default:     gsc_view = view_garb;
endcase


/**********************************synchro_view*****************************/
   /*  pdataI[0:4] = 5'b10010(IOA0) or 5'b10110 (IOA1) */

case ( int_view_sel )            // synopsys full_case  parallel_case
  3'b000:
    begin
      synchro_view = {dp_q_obsv_rcount, dp_q_obsv_wcount, inq_obsv_rcount,
                inq_obsv_wcount, 17'b0};
    end
  3'b001:
    begin
      synchro_view = oQ_view[0:20] ;
    end      assign oQ_view[0:20] = { sync_write_cnt[0:5],
                    sync_read_cnt_1[0:5],
                    in_cmp_cnt[0:5],
                    o_OutQ_valid,
                    k_OutQ_valid,
                    almost_full } ;
    assign oQ_oflow_view[0:4] = oflow_cnt[0:4] ;
  3'b010:
    begin
      synchro_view = { RRQ_view[0:13], oQ_oflow_view[0:4], 2'b00 } ;
    end
```

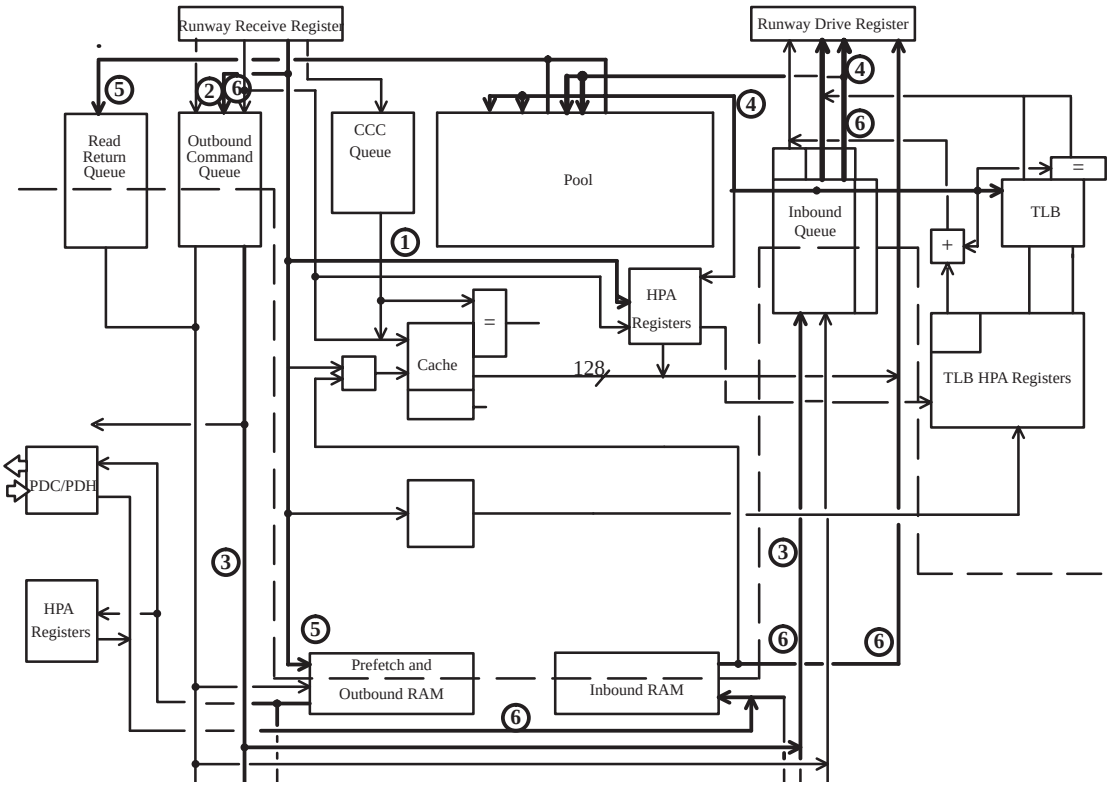
```
  assign RRQ_view[0:13] = { sync_write_cnt[0:3],
                sync_read_cnt_1[0:3],
                in_cmp_cnt[0:3],
                o_RRQ_valid,
                k_RRQ_valid } ;
3'b100:
  begin
    synchro_view = { iQ_view[0:16],rCmdReset,rCmdClear,k_reset_sync,rFlexEnb }
  end
  assign iQ_view[0:16] = { sync_write_cnt[0:3],
                sync_read_cnt_1[0:3],
                in_cmp_cnt[0:3],
                oflow_cnt[0:2],
                inQ_valid,
                k_InQ_avail } ;
```

10.5. PDC Self Test

To some extent, just writing and reading from PDC and UTurn internal registers verifies some degree of functionality. However, additional features have been added to the UTurn design to simulate GSC+ activity. Specifically, GSC+ DMA reads and GSC+ DMA writes can be configured into the queues, according to the mechanisms described below.

10.5.1. GSC+ DMA Reads

UTurn provides programmability features to allow PDC to create the illusion of a GSC+ read. The diagram and sequence of events shown below, explain this functionality. For details on specific HPA Registers and Commands, please reference the Architectural Register Chapter.



1. PDC must write the GSC+ read command to the Runway TEST_ADDRESS and TEST_INFO registers.

2. PDC must issue a TEST_Ld_OutQ command to trigger the Runway slave block to load the Runway Test Register contents into the Outbound Command Queue with both the Uturn and Diagnostic bits set (indicating that this transaction must be routed to the Inbound Queue).

3. The Outbound Command Queue entry will migrate to the head of the queue and from there it will transfer to the Inbound Queue, appearing to the Runway Inbound block like a GSC+ DMA read with the diagnostic bit set.

4. When the Inbound Queue entry has migrated to the head of the queue, the Runway Inbound block will issue a Runway Read_Shar_or_Priv and create a pool entry, again with the Diagnostic bit set.

5. As with all UTurn mastered reads on Runway, the corresponding data return will get stored in the Read Return RAM and a Read Return Queue entry will get formed, again with the Diagnostic bit set.

6. When the Read Return Queue entry has migrated to the head of the queue, the data will be returned on GSC+. Because the Diagnostic bit is set, the data will also be inserted in the Inbound RAM.

7. The processor must wait for the “read” to complete. a 1 microsecond wait after issuance of the TEST_Ld_OutQ command is sufficient. After this time delay, PDC

may issue GSC+ HPA reads of the OUT_RAM registers to obtain the Read Return RAM contents corresponding to this transaction. Data returns for this HPA read will cycle through the Inbound RAM and get issued on Runway in quantities of 64 bits.

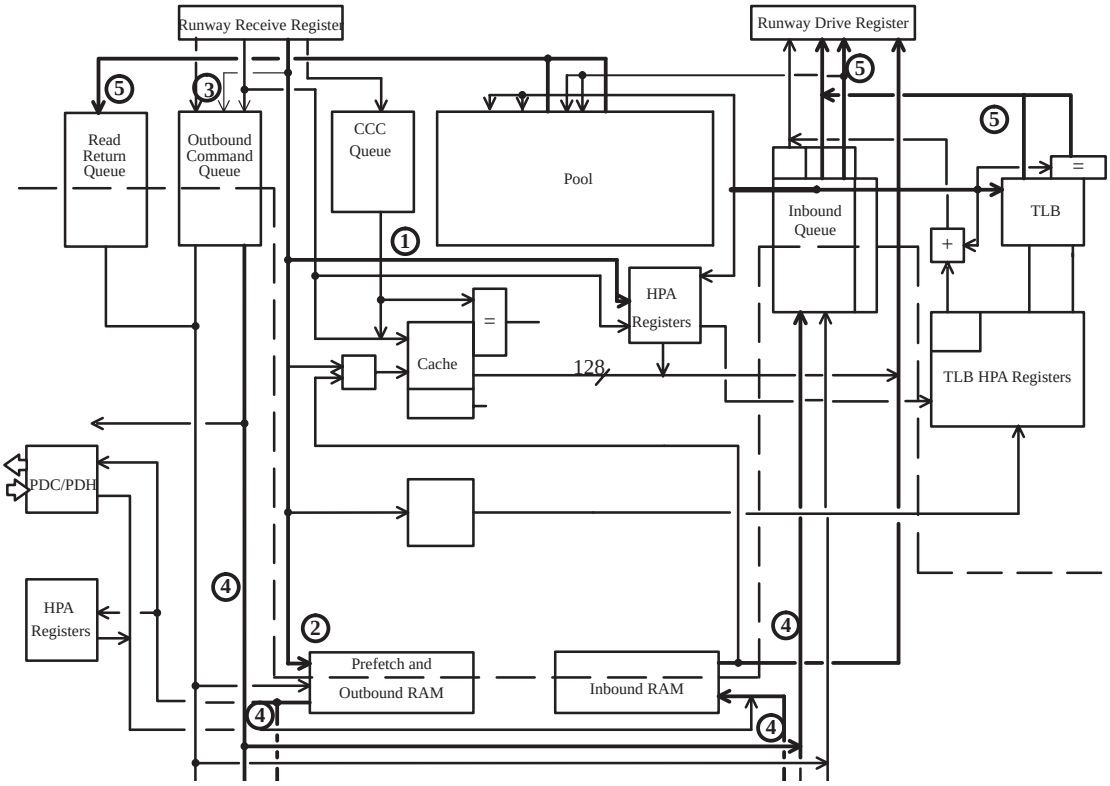
10.5.2. GSC+ DMA Read Returns

This testability feature is simply a subset of the GSC+ DMA Read test described above. It provides PDC with the capability of selecting the Outbound RAM address for a fictional DMA Read Return, thereby allowing test accessibility to all Outbound RAM locations. The following procedure must be followed to perform this test.

1. PDC must write the GSC+ DMA read return transaction information to the Runway TEST_INFO register, selecting the Outbound RAM address.
2. PDC must issue a TEST_Ld_RRQ command to trigger the Runway slave block to load the Runway Test Register contents into the Read Return Queue.
3. The Read Return Queue entry will migrate to the head of the queue and from there it will get issued on GSC+.

10.5.3. GSC+ DMA Writes

UTurn provides programmability features to allow PDC to create the illusion of a GSC+ write. The diagram and sequence of events shown below, explain this functionality. For details on specific HPA Registers and Commands, please reference the Architectural Register Chapter.

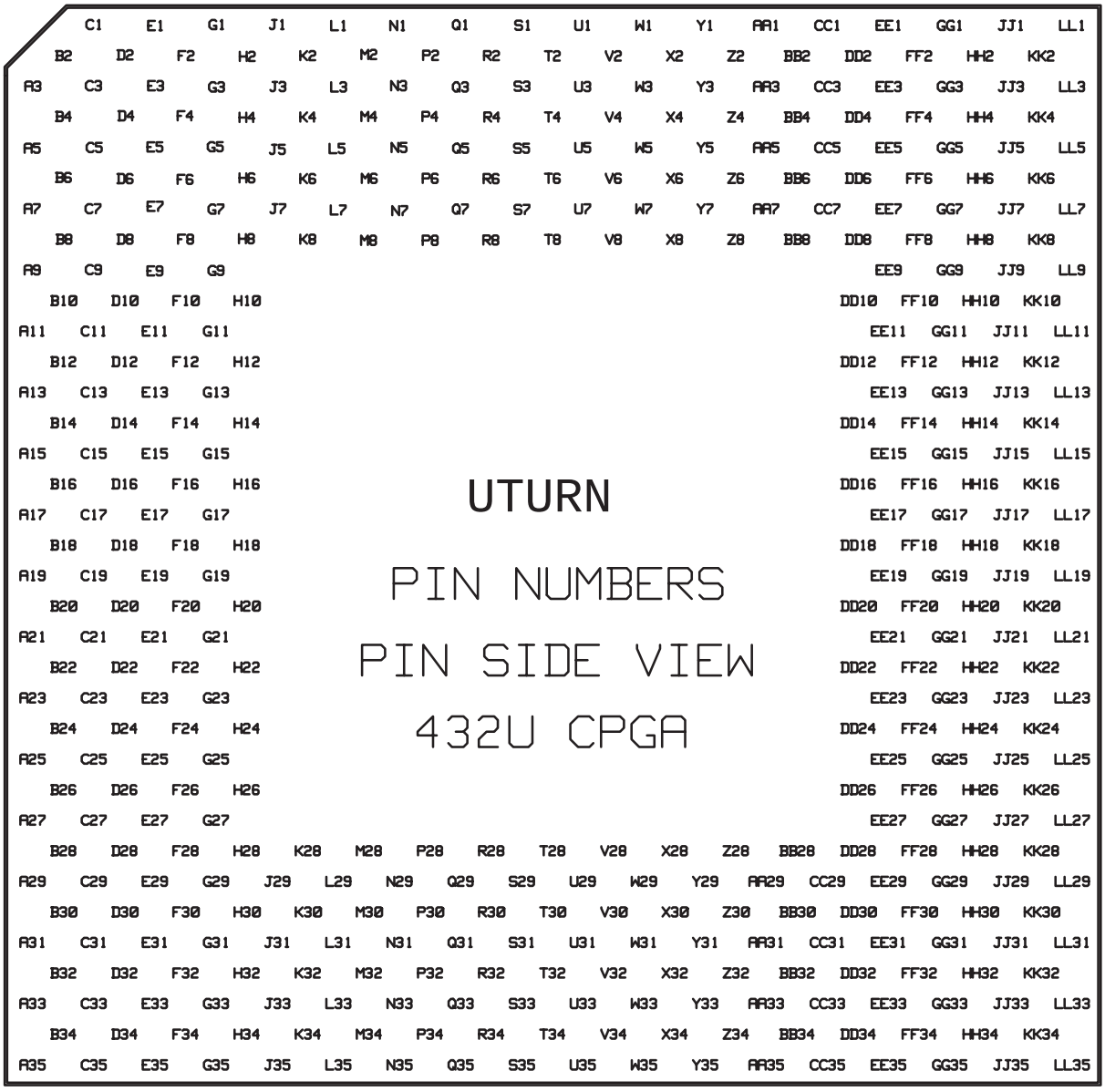


1. PDC must force fictitious data into the Inbound RAM, by following steps 1 through 6 in the above procedure for GSC+ DMA Reads.
2. PDC must write the GSC+ write command to the Runway TEST_ADDRESS and TEST_INFO registers.
3. PDC must issue a TEST_Ld_OutQ command to trigger the Runway slave block to load the Runway Test Register contents into the Outbound Command Queue with both the Uturn and Diagnostic bits set (indicating that this transaction must be routed to the Inbound Queue).
4. The Outbound Command Queue entry will migrate to the head of the queue and from there it will transfer to the Inbound Queue, appearing to the Runway Inbound block like a GSC+ DMA write with the diagnostic bit set.
5. When the Inbound Queue entry has migrated to the head of the queue, the Runway Inbound block will issue a Runway Write16_Purge, Write_Purge, or a Runway Read_Priv followed by a Write_Back depending on the size of the write command and the TLB page type.
6. Now PDC can interrogate Memory to determine if the write occurred properly.

NOTES on DMA Reads/Writes:

1. A fake DMA read/write combination with lock asserted will NOT act the same as a normal GSC transaction combination. The lock will look like it was deasserted and an unlock queue entry will be formed. This will mean that there will be two read_privates done for the read–write combo – the first will be due to the lock–on and the second will be due to the sub–cache–line write.
2. PDC must be very careful when doing the fake GSC DMA writes. PDC should cache all the code before starting this sequence. If not, the data that is written with the DMA write will be somewhat indeterminate. It is best to do a fake DMA read followed by a fake DMA write without any transactions going to UTurn in between.
3. Fake DMA reads that are connected will cause the read to be done and the data to be inserted in the RRR (read return RAM), but the data will not be returned on GSC. For pended reads the data will be returned on GSC.

11. UTurn Pinout



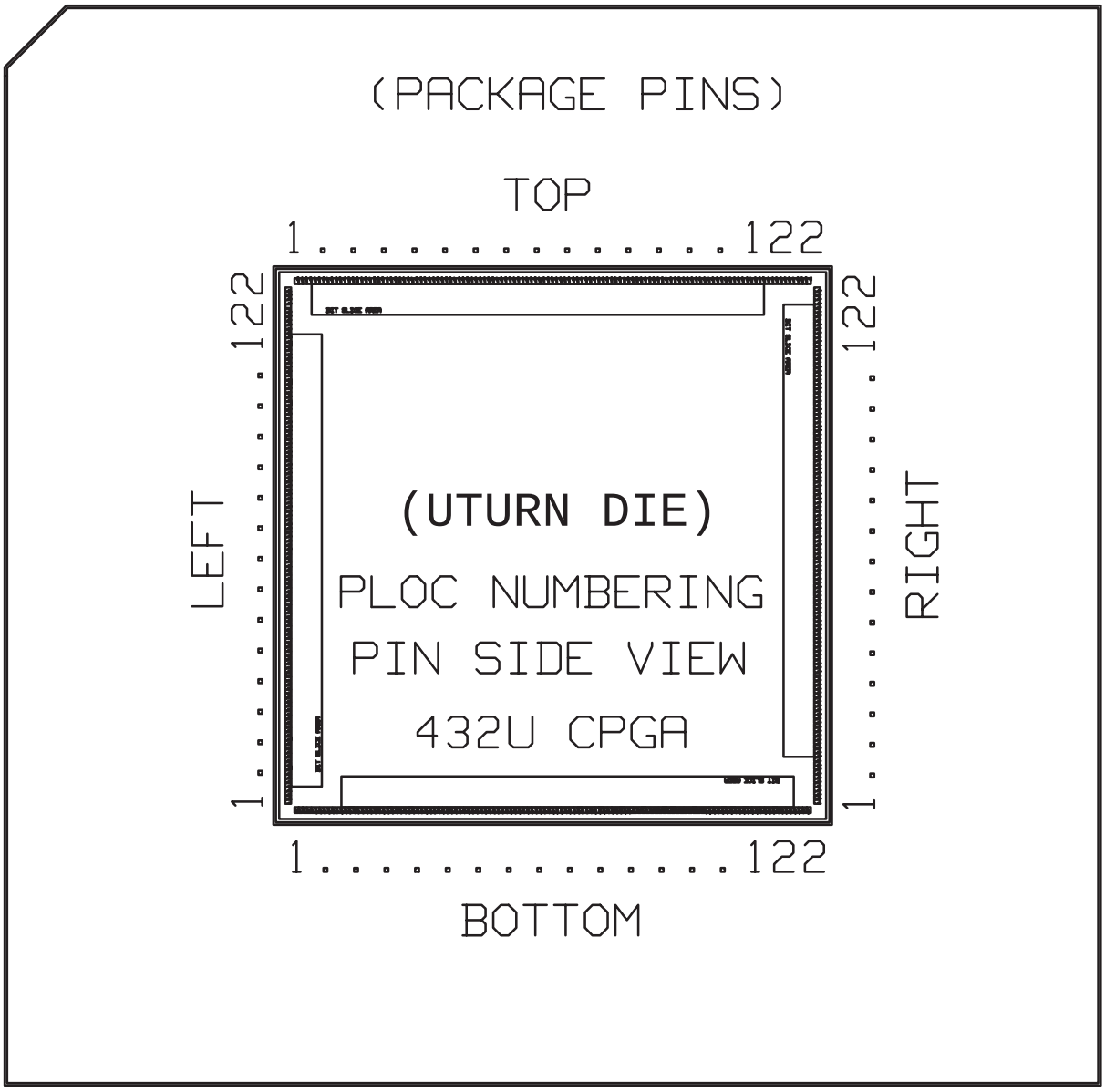

(PACKAGE PINS)
TOP
1.....122
122
1.....
LEFT
(UTURN DIE)
PLOC NUMBERING
PIN SIDE VIEW
432U CPGA
122
.....1
RIGHT
1.....122
BOTTOM

TABLE KEY:

Multiple pin connections are noted by (m*) and notated collectively at the end of the listing.

A ”–” in the ”bond share” column means the PGA bond pad number is unique. A number in this column means that there are multiple bond wires connected from a single large pad on the PGA to indiviual bitslices on the Chip.

SIGNAL TYPES:	I/O	Input/Output
	I	Input
	O	Output
	A	Analog
	I–ECL	Input, standard or DC shifted ECL levels
	VDLXX	VDL with internal VDL/DGND breaker
	GND	core ground } DGND/GND share a common
	DGND	dirty ground } ground plane in the PGA
	VDL	3.3 volt supply (Dirty supply for drivers)
	VDDn	5.0 volt supply (n: 1,2,3,4,5,6,7,8)

m* – Internal multiple pin connections (PGA PIN#’s)

VDD1: D18,F18
VDD2: D6,F8
VDD3: T4,T6
VDD4: DD6,FF4
VDD5: FF18,HH18
VDD6: FF28,HH30,DD26
VDD7: T30,T32
VDD8: F32,H30,K28

VDL: Q1, G3, L3, U3, CC3, EE3, M4, Z4, G5, W5, EE5, K6, P6, A7, E7, GG7, JJ7, T8, C9, FF10, JJ11, D12, HH12, FF14, E15, LL15, KK16, C17, H18, DD18, JJ19, B20, GG21, F22, D24, HH24, F26, JJ27, T28, C29, GG29, X30, BB30, G31, Q31, M32, Z32, J33, S33, EE33, G35, W35,

GND: EE1, LL1, R2, Z2, C3, J3, Q3, S3, W3, Y3, JJ3, F4, K4, R4, L5, F6, M6, X6, Z6, BB6, FF6, HH6, C7, K8, R8, BB8, JJ9, F10, H10, DD10, HH10, GG11, B12, F12, FF12, C13, F14, C15, DD16, JJ17, C19, H20, D20, A21, JJ21, FF22, JJ23, F24, FF24, KK24, C25, E25, D26, H26, FF26, C27, V28, BB28, E29, JJ29, LL29, D30, F30, K30, M30, P30, Z30, FF30, AA31, EE31, V32, BB32, FF32, C33, G33, N33, Q33, U33, AA33, CC33, JJ33, M34, V34, A35, LL35

PGA PAD#	bond share	PGA PIN#	SIGNAL TYPE	PLOC#	SIGNAL NAME	BITSLICE CELL NAME
40	–	m*	VDL	LEFT 1	VDL	U2PADVDLDS
41	40	m*	VDL	LEFT 2	VDL	U2PADVDLDS
42	40	m*	VDL	LEFT 3	VDL	U2PADVDLDS
43	40	m*	VDL	LEFT 4	(NOBONDWIRE)	U2PADVDLDS
44	–	G27	I–ECL	LEFT 5	SYNC_K0_H	/*U2GSCCLK*/
45	40	m*	VDL	LEFT 6	(NOBONDWIRE)	/*U2GSCCLK*/
46	–	G29	I–ECL	LEFT 7	SYNC_K0_L	/*U2GSCCLK*/
47	–	m*	GND	LEFT 8	GND	/*U2GSCCLK*/
48	–	E31	O	LEFT 9	K0_RESET	U2RSTOUT
49	–	H28	I/O	LEFT 10	K0_AD[8]	U2GSCPBS2
50	–	G25	I/O	LEFT 11	K0_AD[9]	U2GSCPBS2
51	–	D32	I/O	LEFT 12	K0_AD[10]	U2GSCPBS2
52	47	m*	DGND	LEFT 13	DGND	U2PADGNDD
53	–	F28	I/O	LEFT 14	K0_AD[11]	U2GSCPBS2
54	–	m*	VDL	LEFT 15	VDL	U2PADVDLD
55	–	H24	I/O	LEFT 16	K0_AD[12]	U2GSCPBS2
56	–	E27	I/O	LEFT 17	K0_AD[13]	U2GSCPBS2
57	54	m*	VDL	LEFT 18	VDL	U2PADVDLD
58	–	C31	I/O	LEFT 19	K0_AD[14]	U2GSCPBS2
59	–	m*	DGND	LEFT 20	DGND	U2PADGNDD
60	–	A33	I/O	LEFT 21	K0_AD[15]	U2GSCPBS2
61	–	B34	I/O	LEFT 22	K0_AD[16]	U2GSCPBS2
62	–	G23	I/O	LEFT 23	K0_AD[17]	U2GSCPBS2
63	–	B32	I/O	LEFT 24	K0_AD[18]	U2GSCPBS2
64	59	m*	DGND	LEFT 25	DGND	U2PADGNDD
65	–	H22	I/O	LEFT 26	K0_AD[19]	U2GSCPBS2
66	–	m*	VDL	LEFT 27	VDL	U2PADVDLD
67	–	D28	I/O	LEFT 28	K0_AD[20]	U2GSCPBS2
68	–	A31	I/O	LEFT 29	K0_AD[21]	U2GSCPBS2
69	66	m*	VDL	LEFT 30	VDL	U2PADVDLD
70	–	A29	I/O	LEFT 31	K0_AD[22]	U2GSCPBS2

71	–	m*	DGND	LEFT 32	DGND	U2PADGNDD
72	–	E23	I/O	LEFT 33	K0_AD[23]	U2GSCPBS2
73	–	B30	I/O	LEFT 34	K0_AD[24]	U2GSCPBS2
74	–	G21	I/O	LEFT 35	K0_AD[25]	U2GSCPBS2
75	–	B28	I/O	LEFT 36	K0_AD[26]	U2GSCPBS2
76	71	m*	DGND	LEFT 37	DGND	U2PADGNDD
77	–	B26	I/O	LEFT 38	K0_AD[27]	U2GSCPBS2
78	–	m*	VDL	LEFT 39	VDL	U2PADVDLD
79	–	C23	I/O	LEFT 40	K0_AD[28]	U2GSCPBS2
80	–	A27	I/O	LEFT 41	K0_AD[29]	U2GSCPBS2
81	78	m*	VDL	LEFT 42	VDL	U2PADVDLD
82	–	E21	I/O	LEFT 43	K0_AD[30]	U2GSCPBS2
83	–	m*	DGND	LEFT 44	DGND	U2PADGNDD
84	–	A25	I/O	LEFT 45	K0_AD[31]	U2GSCPBS2
85	–	F20	I/O	LEFT 46	K0_TYPE[2]	U2GSCTBS2
86	–	A23	I/O	LEFT 47	K0_TYPE[3]	U2GSCTBS2
87	–	D22	I/O	LEFT 48	K0_TYPE[0]	U2GSCTBS2
88	83	m*	DGND	LEFT 49	DGND	U2PADGNDD
89	–	B24	I/O	LEFT 50	K0_TYPE[1]	U2GSCTBS2
90	–	m*	VDL	LEFT 51	VDL	U2PADVDLD
91	–	B22	I	LEFT 52	K0_BRL[0]	U2GSCCBS2
92	–	A19	I	LEFT 53	K0_BRL[1]	U2GSCCBS2
93	90	m*	VDL	LEFT 54	VDL	U2PADVDLD
94	–	E19	I	LEFT 55	K0_BRL[2]	U2GSCCBS2
95	–	C21	I	LEFT 56	K0_BRL[3]	U2GSCCBS2
96	–	G19	I	LEFT 57	K0_BRL[4]	U2GSCCBS2
97	–	m*	DGND	LEFT 58	DGND	U2PADGNDD
98	–	m*	VDD1	LEFT 59	VDD	U2PADVDDIOL
99	97	m*	GND	LEFT 60	GND	U2PADGNDIOL
100	98	m*	VDD1	LEFT 61	VDD	U2PADVDDIOL
101	97	m*	GND	LEFT 62	GND	U2PADGNDIOL
102	98	m*	VDD1	LEFT 63	VDD	U2PADVDDIOL

103	97	m*	GND	LEFT 64	GND	U2PADGNDIOL
104	97	m*	DGND	LEFT 65	DGND	U2PADGNDD
105	–	B18	I	LEFT 66	K0_BRL[5]	U2GSCCBS2
106	–	E17	I/O	LEFT 67	K0_ADDVL	U2GSCCBS2
107	–	B16	I/O	LEFT 68	K0_READYL	U2GSCCBS2
108	–	m*	VDL	LEFT 69	VDL	U2PADVDLD
109	–	D16	I/O	LEFT 70	K0_PARITY	U2GSCPBS2
110	–	A17	I/O	LEFT 71	K0_ERRORL	U2GSCCBS2
111	108	m*	VDL	LEFT 72	VDL	U2PADVDLD
112	–	A15	O	LEFT 73	K0_BGL[0]	U2GSCCBS2
113	–	m*	DGND	LEFT 74	DGND	U2PADGNDD
114	–	A13	O	LEFT 75	K0_BGL[1]	U2GSCCBS2
115	–	B14	O	LEFT 76	K0_BGL[2]	U2GSCCBS2
116	–	A11	O	LEFT 77	K0_BGL[3]	U2GSCCBS2
117	–	F16	O	LEFT 78	K0_BGL[4]	U2GSCCBS2
118	113	m*	DGND	LEFT 79	DGND	U2PADGNDD
119	–	D14	O	LEFT 80	K0_BGL[5]	U2GSCCBS2
120	–	m*	VDL	LEFT 81	VDL	U2PADVDLD
121	–	H16	I	LEFT 82	K0_LSL	U2GSCCBS2
122	–	A9	I	LEFT 83	K0_INTERRUPTL	U2GSCCBS2
123	120	m*	VDLXX	LEFT 84	VDL	PADKHVDLBL
124	–	C11	I/O	LEFT 85	ADDR_DATA[0]	PADIOSSFC
125	–	m*	DGND	LEFT 86	DGND	PADKHDGND
126	–	G15	I/O	LEFT 87	ADDR_DATA[3]	PADIOSSFC
127	–	B10	I/O	LEFT 88	ADDR_DATA[1]	PADIOSSFC
128	–	E13	I/O	LEFT 89	ADDR_DATA[5]	PADIOSSFC
129	–	B8	I/O	LEFT 90	ADDR_DATA[2]	PADIOSSFC
130	125	m*	DGND	LEFT 91	DGND	PADKHDGND
131	–	B6	I/O	LEFT 92	ADDR_DATA[7]	PADIOSSFC
132	–	m*	VDL	LEFT 93	VDL	PADKHVDL
133	–	D10	I/O	LEFT 94	ADDR_DATA[4]	PADIOSSFC
134	–	A5	I/O	LEFT 95	ADDR_DATA[9]	PADIOSSFC

135	132	m*	VDL	LEFT 96	VDL	PADKHVDL
136	–	A3	I/O	LEFT 97	ADDR_DATA[6]	PADIOSSFC
137	–	m*	DGND	LEFT 98	DGND	PADKHDGND
138	–	E11	I/O	LEFT 99	ADDR_DATA[11]	PADIOSSFC
139	–	H14	I/O	LEFT 100	ADDR_DATA[8]	PADIOSSFC
140	–	G13	I/O	LEFT 101	ADDR_DATA[13]	PADIOSSFC
141	–	D8	I/O	LEFT 102	ADDR_DATA[10]	PADIOSSFC
142	137	m*	DGND	LEFT 103	DGND	PADKHDGND
143	–	B4	I/O	LEFT 104	ADDR_DATA[15]	PADIOSSFC
144	–	m*	VDL	LEFT 105	VDL	PADKHVDL
145	–	G9	I/O	LEFT 106	ADDR_DATA[12]	PADIOSSFC
146	–	C5	I/O	LEFT 107	ADDR_DATA[17]	PADIOSSFC
147	144	m*	VDL	LEFT 108	VDL	PADKHVDL
148	–	E9	I/O	LEFT 109	ADDR_DATA[14]	PADIOSSFC
149	–	H12	I/O	LEFT 110	AD_PAR[0]	PADIOSSFC
150	–	G11	I/O	LEFT 111	ADDR_DATA[16]	PADIOSSFC
151	–	m*	DGND	LEFT 112	DGND	PADKHDGNDS
152	–	m*	VDD2	LEFT 113	VDD	PADKHVDDS
153	151	m*	GND	LEFT 114	GND	PADKHCGNDSU
154	152	m*	VDD2	LEFT 115	VDD	PADKHVDDS
155	151	m*	GND	LEFT 116	GND	PADKHCGNDSU
156	152	m*	VDD2	LEFT 117	VDD	PADKHVDDS
157	151	m*	GND	LEFT 118	GND	PADKHCGNDSU
158	152	m*	VDD2	LEFT 119	VDD	PADKHVDDS
159	151	m*	GND	LEFT 120	GND	PADKHCGNDSU
160	151	m*	GND	LEFT 121	GND	PADKHCGNDSU
161	151	m*	GND	LEFT 122	GND	PADKHCGNDSU
240	–	m*	VDL	TOP 1	VDL	PADKHVDLS
241	240	m*	VDL	TOP 2	VDL	PADKHVDLS
242	240	m*	VDL	TOP 3	VDL	PADKHVDLS
243	240	m*	VDL	TOP 4	(NOBONDWIRE)	PADKHVDLS
244	–	J7	I–ECL	TOP 5	SYNC_IOA0_L	PADSYNCM

245	240	m*	VDL	TOP 6	(NOBONDWIRE)	PADKHVDLS
246	–	G7	I–ECL	TOP 7	SYNC_IOA0_H	PADSYNCM
247	–	m*	GND	TOP 8	GND	PADKHCGNDSU
248	–	E5	I/O	TOP 9	ADDR_DATA[19]	PADIOSSFC
249	–	H8	I/O	TOP 10	ADDR_DATA[18]	PADIOSSFC
250	–	L7	I/O	TOP 11	ADDR_DATA[21]	PADIOSSFC
251	–	D4	I/O	TOP 12	ADDR_DATA[20]	PADIOSSFC
252	247	m*	DGND	TOP 13	DGND	PADKHDGND
253	–	H6	I/O	TOP 14	ADDR_DATA[23]	PADIOSSFC
254	–	m*	VDL	TOP 15	VDL	PADKHVDL
255	–	M8	I/O	TOP 16	ADDR_DATA[22]	PADIOSSFC
256	–	J5	I/O	TOP 17	ADDR_DATA[25]	PADIOSSFC
257	254	m*	VDL	TOP 18	VDL	PADKHVDL
258	–	E3	I/O	TOP 19	ADDR_DATA[24]	PADIOSSFC
259	–	m*	DGND	TOP 20	DGND	PADKHDGND
260	–	C1	I/O	TOP 21	ADDR_DATA[27]	PADIOSSFC
261	–	B2	I/O	TOP 22	ADDR_DATA[26]	PADIOSSFC
262	–	N7	I/O	TOP 23	ADDR_DATA[29]	PADIOSSFC
263	–	D2	I/O	TOP 24	ADDR_DATA[28]	PADIOSSFC
264	259	m*	DGND	TOP 25	DGND	PADKHDGND
265	–	P8	I/O	TOP 26	ADDR_DATA[31]	PADIOSSFC
266	–	m*	VDL	TOP 27	VDL	PADKHVDL
267	–	H4	I/O	TOP 28	ADDR_DATA[30]	PADIOSSFC
268	–	E1	I/O	TOP 29	ADDR_DATA[33]	PADIOSSFC
269	266	m*	VDL	TOP 30	VDL	PADKHVDL
270	–	G1	I/O	TOP 31	ADDR_DATA[32]	PADIOSSFC
271	–	m*	DGND	TOP 32	DGND	PADKHDGND
272	–	N5	I/O	TOP 33	ADDR_DATA[35]	PADIOSSFC
273	–	F2	I/O	TOP 34	ADDR_DATA[34]	PADIOSSFC
274	–	Q7	I/O	TOP 35	ADDR_DATA[37]	PADIOSSFC
275	–	H2	I/O	TOP 36	ADDR_DATA[36]	PADIOSSFC
276	271	m*	DGND	TOP 37	DGND	PADKHDGND

277	–	K2	I/O	TOP 38	ADDR_DATA[39]	PADIOSSFC
278	–	m*	VDL	TOP 39	VDL	PADKHVDL
279	–	N3	I/O	TOP 40	ADDR_DATA[38]	PADIOSSFC
280	–	J1	I/O	TOP 41	ADDR_DATA[41]	PADIOSSFC
281	278	m*	VDL	TOP 42	VDL	PADKHVDL
282	–	Q5	I/O	TOP 43	ADDR_DATA[40]	PADIOSSFC
283	–	m*	DGND	TOP 44	DGND	PADKHDGND
284	–	L1	I/O	TOP 45	ADDR_DATA[43]	PADIOSSFC
285	–	R6	I/O	TOP 46	ADDR_DATA[42]	PADIOSSFC
286	–	N1	I/O	TOP 47	ADDR_DATA[45]	PADIOSSFC
287	–	P4	I/O	TOP 48	ADDR_DATA[44]	PADIOSSFC
288	283	m*	DGND	TOP 49	DGND	PADKHDGND
289	–	M2	I/O	TOP 50	ADDR_DATA[47]	PADIOSSFC
290	–	m*	VDL	TOP 51	VDL	PADKHVDL
291	–	P2	I/O	TOP 52	ADDR_DATA[46]	PADIOSSFC
292	–	S1	I/O	TOP 53	AD_PAR[1]	PADIOSSFC
293	290	m*	VDL	TOP 54	(NOBONDWIRE)	PADKHVDLS
294	–	S5	I–ECL	TOP 55	SYNC_RW_L	PADSYNCM
295	–	m*	GND	TOP 56	GND	PADKHCGNDSU
296	–	S7	I–ECL	TOP 57	SYNC_RW_H	PADSYNCM
297	295	m*	GND	TOP 58	GND	PADKHCGNDSU
298	–	m*	VDD3	TOP 59	VDD	PADKHVDDS
299	295	m*	GND	TOP 60	GND	PADKHCGNDSU
300	298	m*	VDD3	TOP 61	VDD	PADKHVDDS
301	295	m*	GND	TOP 62	GND	PADKHCGNDSU
302	298	m*	VDD3	TOP 63	VDD	PADKHVDDS
303	295	m*	GND	TOP 64	GND	PADKHCGNDSU
304	295	m*	DGND	TOP 65	DGND	PADKHDGNDS
305	–	T2	I/O	TOP 66	ADDR_DATA[49]	PADIOSSFC
306	–	U5	I/O	TOP 67	ADDR_DATA[51]	PADIOSSFC
307	–	V2	I/O	TOP 68	ADDR_DATA[48]	PADIOSSFC
308	–	m*	VDL	TOP 69	VDL	PADKHVDL

309	–	V4	I/O	TOP 70	ADDR_DATA[53]	PADIOSSFC
310	–	U1	I/O	TOP 71	ADDR_DATA[50]	PADIOSSFC
311	308	m*	VDL	TOP 72	VDL	PADKHVDL
312	–	W1	I/O	TOP 73	ADDR_DATA[55]	PADIOSSFC
313	–	m*	DGND	TOP 74	DGND	PADKHDGND
314	–	Y1	I/O	TOP 75	ADDR_DATA[52]	PADIOSSFC
315	–	X2	I/O	TOP 76	ADDR_VALID	PADIOSSFC
316	–	AA1	I/O	TOP 77	ADDR_DATA[54]	PADIOSSFC
317	–	V6	I/O	TOP 78	ADDR_DATA[57]	PADIOSSFC
318	313	m*	DGND	TOP 79	DGND	PADKHDGND
319	–	X4	I/O	TOP 80	ADDR_DATA[56]	PADIOSSFC
320	–	m*	VDL	TOP 81	VDL	PADKHVDL
321	–	V8	I/O	TOP 82	ADDR_DATA[59]	PADIOSSFC
322	–	CC1	I/O	TOP 83	ADDR_DATA[58]	PADIOSSFC
323	320	m*	VDL	TOP 84	VDL	PADKHVDL
324	–	AA3	I/O	TOP 85	ADDR_DATA[61]	PADIOSSFC
325	–	m*	DGND	TOP 86	DGND	PADKHDGND
326	–	W7	I/O	TOP 87	ADDR_DATA[60]	PADIOSSFC
327	–	BB2	I/O	TOP 88	ADDR_DATA[63]	PADIOSSFC
328	–	Y5	I/O	TOP 89	ADDR_DATA[62]	PADIOSSFC
329	–	DD2	I	TOP 90	CLIENT_OP[1]	PADIOSSFC
330	325	m*	DGND	TOP 91	DGND	PADKHDGND
331	–	FF2	I/O	TOP 92	DATA_VALID	PADIOSSFC
332	–	m*	VDL	TOP 93	VDL	PADKHVDL
333	–	BB4	I	TOP 94	CLIENT_ID	PADIOSSFC
334	–	GG1	I	TOP 95	RW_EXTRA2	PADIOSSFC
335	332	m*	VDL	TOP 96	VDL	PADKHVDL
336	–	JJ1	I	TOP 97	POWER_ON	PADIOSSFC
337	–	m*	DGND	TOP 98	DGND	PADKHDGND
338	–	AA5	I	TOP 99	RW_EXTRA1	PADIOSSFC
339	–	X8	I	TOP 100	CLIENT_OP[0]	PADIOSSFC
340	–	Y7	I	TOP 101	U2_ARB_IN	PADIOSSFC

341	–	DD4	I	TOP 102	CLIENT_OP[2]	PADIOSSFC
342	337	m*	DGND	TOP 103	DGND	PADKHDGND
343	–	HH2	O	TOP 104	U2_ARB_OUT	PADIOSSFC
344	–	m*	VDL	TOP 105	VDL	PADKHVDL
345	–	CC7	I	TOP 106	STOP_MOST_IN	PADIOSSFC
346	–	GG3	O	TOP 107	STOP_IO_OUT	PADIOSSFC
347	344	m*	VDL	TOP 108	VDL	PADKHVDL
348	–	CC5	I/O	TOP 109	LONG_TRANS	PADIOSSFC
349	–	Z8	I/O	TOP 110	CTL_PAR	PADIOSSFC
350	–	AA7	O	TOP 111	STOP_MOST_OUT	PADIOSSFC
351	–	m*	DGND	TOP 112	DGND	PADKHDGNDS
352	–	m*	VDD4	TOP 113	VDD	PADKHVDDS
353	351	m*	GND	TOP 114	GND	PADKHCGNDSU
354	352	m*	VDD4	TOP 115	VDD	PADKHVDDS
355	351	m*	GND	TOP 116	GND	PADKHCGNDSU
356	352	m*	VDD4	TOP 117	VDD	PADKHVDDS
357	351	m*	GND	TOP 118	GND	PADKHCGNDSU
358	352	m*	VDD4	TOP 119	VDD	PADKHVDDS
359	351	m*	GND	TOP 120	GND	PADKHCGNDSU
360	351	m*	GND	TOP 121	GND	PADKHCGNDSU
361	351	m*	GND	TOP 122	GND	PADKHCGNDSU
440	–	m*	VDL	RIGHT 122	VDL	PADKHVDLS
441	440	m*	VDL	RIGHT 121	VDL	PADKHVDLS
442	440	m*	VDL	RIGHT 120	VDL	PADKHVDLS
443	440	m*	VDL	RIGHT 119	(NOBONDWIRE)	PADKHVDLS
444	–	EE9	I–ECL	RIGHT 118	SYNC_IOA1_L	PADSYNCM
445	440	m*	VDL	RIGHT 117	(NOBONDWIRE)	PADKHVDLS
446	–	EE7	I–ECL	RIGHT 116	SYNC_IOA1_H	PADSYNCM
447	–	m*	GND	RIGHT 115	GND	PADKHCGNDSU
448	–	GG5	I/O	RIGHT 114	MASTER_ID[2]	PADIOSSFC
449	–	DD8	I/O	RIGHT 113	TRANS_ID[0]	PADIOSSFC
450	–	EE11	I/O	RIGHT 112	TRANS_ID[1]	PADIOSSFC

451	–	HH4	I/O	RIGHT 111	MASTER_ID[1]	PADIOSSFC
452	447	m*	DGND	RIGHT 110	DGND	PADKHDGND
453	–	FF8	I/O	RIGHT 109	TRANS_ID[2]	PADIOSSFC
454	–	m*	VDL	RIGHT 108	VDL	PADKHVDL
455	–	DD12	I/O	RIGHT 107	TRANS_ID[3]	PADIOSSFC
456	–	GG9	I/O	RIGHT 106	TRANS_ID[4]	PADIOSSFC
457	454	m*	VDL	RIGHT 105	VDL	PADKHVDL
458	–	JJ5	I/O	RIGHT 104	TRANS_ID[5]	PADIOSSFC
459	–	m*	DGND	RIGHT 103	DGND	PADKHDGND
460	–	LL3	O	RIGHT 102	COH0[0]	PADIOSSFC
461	–	KK2	O	RIGHT 101	COH0[1]	PADIOSSFC
462	–	EE13	O	RIGHT 100	COH1[0]	PADIOSSFC
463	–	KK4	O	RIGHT 99	COH1[1]	PADIOSSFC
464	459	m*	DGND	RIGHT 98	DGND	PADKHDGND
465	–	DD14	I/O	RIGHT 97	MASTER_ID[0]	PADIOSSFC
466	–	m*	VDLXX	RIGHT 96	VDL	PADKHVDLBR
467	–	HH8	I	RIGHT 95	K1_INTERRUPTL	U2GSCCBS2
468	–	LL5	I	RIGHT 94	K1_LSL	U2GSCCBS2
469	466	m*	VDL	RIGHT 93	VDL	U2PADVDLD
470	–	LL7	O	RIGHT 92	K1_BGL[5]	U2GSCCBS2
471	–	m*	DGND	RIGHT 91	DGND	U2PADGNDD
472	–	GG13	O	RIGHT 90	K1_BGL[4]	U2GSCCBS2
473	–	KK6	O	RIGHT 89	K1_BGL[3]	U2GSCCBS2
474	–	EE15	O	RIGHT 88	K1_BGL[2]	U2GSCCBS2
475	–	KK8	O	RIGHT 87	K1_BGL[1]	U2GSCCBS2
476	471	m*	DGND	RIGHT 86	DGND	U2PADGNDD
477	–	KK10	O	RIGHT 85	K1_BGL[0]	U2GSCCBS2
478	–	m*	VDL	RIGHT 84	VDL	U2PADVDLD
479	–	JJ13	I/O	RIGHT 83	K1_ERRORL	U2GSCCBS2
480	–	LL9	I/O	RIGHT 82	K1_PARITY	U2GSCPBS2
481	478	m*	VDL	RIGHT 81	VDL	U2PADVDLD
482	–	GG15	I/O	RIGHT 80	K1_READYL	U2GSCCBS2

483	–	m*	DGND	RIGHT 79	DGND	U2PADGNDD
484	–	LL11	I/O	RIGHT 78	K1_ADDVL	U2GSCCBS2
485	–	FF16	I	RIGHT 77	K1_BRL[5]	U2GSCCBS2
486	–	LL13	I	RIGHT 76	K1_BRL[4]	U2GSCCBS2
487	–	HH14	I	RIGHT 75	K1_BRL[3]	U2GSCCBS2
488	483	m*	DGND	RIGHT 74	DGND	U2PADGNDD
489	–	KK12	I	RIGHT 73	K1_BRL[2]	U2GSCCBS2
490	–	m*	VDL	RIGHT 72	VDL	U2PADVDLD
491	–	KK14	I	RIGHT 71	K1_BRL[1]	U2GSCCBS2
492	–	LL17	I	RIGHT 70	K1_BRL[0]	U2GSCCBS2
493	490	m*	VDL	RIGHT 69	VDL	U2PADVDLD
494	–	GG17	I/O	RIGHT 68	K1_TYPE[1]	U2GSCTBS2
495	–	JJ15	I/O	RIGHT 67	K1_TYPE[0]	U2GSCTBS2
496	–	EE17	I/O	RIGHT 66	K1_TYPE[3]	U2GSCTBS2
497	–	m*	DGND	RIGHT 65	DGND	U2PADGNDD
498	–	m*	VDD5	RIGHT 64	VDD	U2PADVDDIOL
499	497	m*	GND	RIGHT 63	GND	U2PADGNDIOL
500	498	m*	VDD5	RIGHT 62	VDD	U2PADVDDIOL
501	497	m*	GND	RIGHT 61	GND	U2PADGNDIOL
502	498	m*	VDD5	RIGHT 60	VDD	U2PADVDDIOL
503	497	m*	GND	RIGHT 59	GND	U2PADGNDIOL
504	497	m*	DGND	RIGHT 58	DGND	U2PADGNDD
505	–	KK18	I/O	RIGHT 57	K1_TYPE[2]	U2GSCTBS2
506	–	GG19	I/O	RIGHT 56	K1_AD[31]	U2GSCPBS2
507	–	KK20	I/O	RIGHT 55	K1_AD[30]	U2GSCPBS2
508	–	m*	VDL	RIGHT 54	VDL	U2PADVDLD
509	–	HH20	I/O	RIGHT 53	K1_AD[29]	U2GSCPBS2
510	–	LL19	I/O	RIGHT 52	K1_AD[28]	U2GSCPBS2
511	508	m*	VDL	RIGHT 51	VDL	U2PADVDLD
512	–	LL21	I/O	RIGHT 50	K1_AD[27]	U2GSCPBS2
513	–	m*	DGND	RIGHT 49	DGND	U2PADGNDD
514	–	LL23	I/O	RIGHT 48	K1_AD[26]	U2GSCPBS2

515	–	KK22	I/O	RIGHT 47	K1_AD[25]	U2GSCPBS2
516	–	LL25	I/O	RIGHT 46	K1_AD[24]	U2GSCPBS2
517	–	FF20	I/O	RIGHT 45	K1_AD[23]	U2GSCPBS2
518	513	m*	DGND	RIGHT 44	DGND	U2PADGNDD
519	–	HH22	I/O	RIGHT 43	K1_AD[22]	U2GSCPBS2
520	–	m*	VDL	RIGHT 42	VDL	U2PADVDLD
521	–	DD20	I/O	RIGHT 41	K1_AD[21]	U2GSCPBS2
522	–	LL27	I/O	RIGHT 40	K1_AD[20]	U2GSCPBS2
523	520	m*	VDL	RIGHT 39	VDL	U2PADVDLD
524	–	JJ25	I/O	RIGHT 38	K1_AD[19]	U2GSCPBS2
525	–	m*	DGND	RIGHT 37	DGND	U2PADGNDD
526	–	EE21	I/O	RIGHT 36	K1_AD[18]	U2GSCPBS2
527	–	KK26	I/O	RIGHT 35	K1_AD[17]	U2GSCPBS2
528	–	GG23	I/O	RIGHT 34	K1_AD[16]	U2GSCPBS2
529	–	KK28	I/O	RIGHT 33	K1_AD[15]	U2GSCPBS2
530	525	m*	DGND	RIGHT 32	DGND	U2PADGNDD
531	–	KK30	I/O	RIGHT 31	K1_AD[14]	U2GSCPBS2
532	–	m*	VDL	RIGHT 30	VDL	U2PADVDLD
533	–	HH26	I/O	RIGHT 29	K1_AD[13]	U2GSCPBS2
534	–	LL31	I/O	RIGHT 28	K1_AD[12]	U2GSCPBS2
535	532	m*	VDL	RIGHT 27	VDL	U2PADVDLD
536	–	LL33	I/O	RIGHT 26	K1_AD[11]	U2GSCPBS2
537	–	m*	DGND	RIGHT 25	DGND	U2PADGNDD
538	–	GG25	I/O	RIGHT 24	K1_AD[10]	U2GSCPBS2
539	–	DD22	I/O	RIGHT 23	K1_AD[9]	U2GSCPBS2
540	–	EE23	I/O	RIGHT 22	K1_AD[8]	U2GSCPBS2
541	–	HH28	I/O	RIGHT 21	K1_AD[7]	U2GSCPBS2
542	537	m*	DGND	RIGHT 20	DGND	U2PADGNDD
543	–	KK32	I/O	RIGHT 19	K1_AD[6]	U2GSCPBS2
544	–	m*	VDL	RIGHT 18	VDL	U2PADVDLD
545	–	EE27	I/O	RIGHT 17	K1_AD[5]	U2GSCPBS2
546	–	JJ31	I/O	RIGHT 16	K1_AD[4]	U2GSCPBS2

547	544	m*	VDL	RIGHT 15	VDL	U2PADVDLD
548	–	GG27	I/O	RIGHT 14	K1_AD[3]	U2GSCPBS2
549	–	DD24	I/O	RIGHT 13	K1_AD[2]	U2GSCPBS2
550	–	EE25	I/O	RIGHT 12	K1_AD[1]	U2GSCPBS2
551	–	m*	DGND	RIGHT 11	DGND	U2PADGNDDS
552	–	m*	VDD6	RIGHT 10	VDD	U2PADVDDIOLS
553	551	m*	GND	RIGHT 9	GND	U2PADGNDIOLS
554	552	m*	VDD6	RIGHT 8	VDD	U2PADVDDIOLS
555	551	m*	GND	RIGHT 7	GND	U2PADGNDIOLS
556	552	m*	VDD6	RIGHT 6	VDD	U2PADVDDIOLS
557	551	m*	GND	RIGHT 5	GND	U2PADGNDIOLS
558	552	m*	VDD6	RIGHT 4	VDD	U2PADVDDIOLS
559	551	m*	GND	RIGHT 3	GND	U2PADGNDIOLS
560	551	m*	GND	RIGHT 2	GND	U2PADGNDIOLS
561	551	m*	GND	RIGHT 1	GND	U2PADGNDIOLS
640	–	m*	VDL	BOTTOM 122	VDL	U2PADVDLDS
641	640	m*	VDL	BOTTOM 121	VDL	U2PADVDLDS
642	640	m*	VDL	BOTTOM 120	VDL	U2PADVDLDS
643	640	m*	VDL	BOTTOM 119	(NOBONDWIRE)	U2PADVDLDS
644	–	CC29	I–ECL	BOTTOM 118	SYNC_K1_H	/**********/
645	640	m*	VDL	BOTTOM 117	(NOBONDWIRE)	/*U2GSCCLK*/
646	–	EE29	I–ECL	BOTTOM 116	SYNC_K1_L	/* */
647	–	m*	GND	BOTTOM 115	GND	/**********/
648	–	GG31	O	BOTTOM 114	K1_RESET	U2RSTOUT
649	–	DD28	I/O	BOTTOM 113	K1_AD[0]	U2GSCPBS2
650	–	AA29	I	BOTTOM 112	K1_PACKL	U2GSCCBS2
651	–	HH32	I	BOTTOM 111	K1_RETRYL	U2GSCCBS2
652	647	m*	DGND	BOTTOM 110	DGND	U2PADGNDD
653	–	DD30	O	BOTTOM 109	K1_PENDL	U2GSCCBS2
654	–	m*	VDL	BOTTOM 108	VDL	U2PADVDLD
655	–	Z28	O	BOTTOM 107	K1_DRRL	U2GSCCBS2
656	–	CC31	I	BOTTOM 106	K1_XQL	U2GSCCBS2

657	654	m*	VDLXX	BOTTOM 105	VDL	U2PADVDLDBR
658	–	GG33	O	BOTTOM 104	PADDR[20]	UJ2PADPNIO
659	–	m*	DGND	BOTTOM 103	DGND	UJ2PADGNDD
660	–	JJ35	O	BOTTOM 102	PADDR[19]	UJ2PADPNIO
661	–	KK34	O	BOTTOM 101	PADDR[18]	UJ2PADPNIO
662	–	Y29	O	BOTTOM 100	PADDR[17]	UJ2PADPNIO
663	–	HH34	O	BOTTOM 99	PADDR[16]	UJ2PADPNIO
664	659	m*	DGND	BOTTOM 98	DGND	UJ2PADGNDD
665	–	X28	O	BOTTOM 97	PADDR[15]	UJ2PADPNIO
666	–	m*	VDL	BOTTOM 96	VDL	UJ2PADVDDD
667	–	DD32	O	BOTTOM 95	PADDR[14]	UJ2PADPNIO
668	–	GG35	O	BOTTOM 94	PADDR[13]	UJ2PADPNIO
669	666	m*	VDL	BOTTOM 93	VDL	UJ2PADVDDD
670	–	EE35	O	BOTTOM 92	PADDR[12]	UJ2PADPNIO
671	–	m*	DGND	BOTTOM 91	DGND	UJ2PADGNDD
672	–	Y31	O	BOTTOM 90	PADDR[11]	UJ2PADPNIO
673	–	FF34	O	BOTTOM 89	PADDR[10]	UJ2PADPNIO
674	–	W29	O	BOTTOM 88	PADDR[9]	UJ2PADPNIO
675	–	DD34	O	BOTTOM 87	PADDR[8]	UJ2PADPNIO
676	671	m*	DGND	BOTTOM 86	DGND	UJ2PADGNDD
677	–	BB34	O	BOTTOM 85	PADDR[7]	UJ2PADPNIO
678	–	m*	VDL	BOTTOM 84	VDL	UJ2PADVDDD
679	–	Y33	O	BOTTOM 83	PADDR[6]	UJ2PADPNIO
680	–	CC35	O	BOTTOM 82	PADDR[5]	UJ2PADPNIO
681	678	m*	VDL	BOTTOM 81	VDL	UJ2PADVDDD
682	–	W31	O	BOTTOM 80	PADDR[4]	UJ2PADPNIO
683	–	m*	DGND	BOTTOM 79	DGND	UJ2PADGNDD
684	–	AA35	O	BOTTOM 78	PADDR[3]	UJ2PADPNIO
685	–	V30	O	BOTTOM 77	PADDR[2]	UJ2PADPNIO
686	–	Y35	O	BOTTOM 76	PADDR[1]	UJ2PADPNIO
687	–	X32	O	BOTTOM 75	PADDR[0]	UJ2PADPNIO
688	683	m*	DGND	BOTTOM 74	DGND	UJ2PADGNDD

689	–	Z34	O	BOTTOM 73	POEL	UJ2PADPNIO
690	–	m*	VDL	BOTTOM 72	VDL	UJ2PADVDDD
691	–	X34	O	BOTTOM 71	PWEL	UJ2PADPNIO
692	–	U35	O	BOTTOM 70	PADDRVL	UJ2PADPNIO
693	690	m*	VDL	BOTTOM 69	VDL	UJ2PADVDDD
694	–	U31	A	BOTTOM 68	XTAL1	RTC_PAD
695	–	W33	I	BOTTOM 67	RTC_VDD	RTC_VDD
696	–	U29	A	BOTTOM 66	XTAL2	RTC_PAD
697	–	m*	GND	BOTTOM 65	GND	UJ2PADGNDIOL
698	–	m*	VDD7	BOTTOM 64	VDD	UJ2PADVDDIOL
699	697	m*	GND	BOTTOM 63	GND	UJ2PADGNDIOL
700	698	m*	VDD7	BOTTOM 62	VDD	UJ2PADVDDIOL
701	697	m*	GND	BOTTOM 61	GND	UJ2PADGNDIOL
702	698	m*	VDD7	BOTTOM 60	VDD	UJ2PADVDDIOL
703	697	m*	GND	BOTTOM 59	GND	UJ2PADGNDIOL
704	697	m*	DGND	BOTTOM 58	DGND	UJ2PADGNDDX
705	–	T34	I/O	BOTTOM 57	PDATA[2]	UJ2PADPNIOX
706	–	S31	I/O	BOTTOM 56	PDATA[7]	UJ2PADPNIOX
707	–	R34	I/O	BOTTOM 55	PDATA[6]	UJ2PADPNIOX
708	–	m*	VDL	BOTTOM 54	(NOBONDWIRE)	UJ2PADVDDD
709	–	R32	I/O	BOTTOM 53	PDATA[5]	UJ2PADPNIOX
710	–	S35	I/O	BOTTOM 52	PDATA[4]	UJ2PADPNIOX
711	708	m*	VDL	BOTTOM 51	(NOBONDWIRE)	UJ2PADVDDD
712	–	Q35	I/O	BOTTOM 50	PDATA[3]	UJ2PADPNIOX
713	–	m*	DGND	BOTTOM 49	DGND	UJ2PADGNDD
714	–	N35	I/O	BOTTOM 48	PDATA[1]	UJ2PADPNIOX
715	–	P34	I	BOTTOM 47	TRST	UPADPNIOX
716	–	L35	I/O	BOTTOM 46	PDATA[0]	UJ2PADPNIOX
717	–	R30	I	BOTTOM 45	TCK	UPADPNIOX
718	713	m*	DGND	BOTTOM 44	DGND	UJ2PADGNDD
719	–	P32	I	BOTTOM 43	TOCL	UJ2PADPNIOX
720	–	m*	VDL	BOTTOM 42	(NOBONDWIRE)	UJ2PADVDDD

721	–	R28	O	BOTTOM 41	FP_DATA	UJ2PADPNIOX
722	–	J35	I	BOTTOM 40	TMS	UPADPNIOX
723	720	m*	VDL	BOTTOM 39	(NOBONDWIRE)	UJ2PADVDDD
724	–	L33	O	BOTTOM 38	FP_CLK	UJ2PADPNIOX
725	–	m*	DGND	BOTTOM 37	DGND	UJ2PADGNDD
726	–	Q29	I	BOTTOM 36	PFAIL_WARNING_L	UJ2PADPNIOX
727	–	K34	I	BOTTOM 35	ARESETL	UJ2PADPNIOX
728	–	N31	I	BOTTOM 34	TDI	UPADPNIOX
729	–	H34	I	BOTTOM 33	PDC_EXTRA1	UJ2PADPNIOX
730	725	m*	DGND	BOTTOM 32	DGND	UJ2PADGNDD
731	–	F34	O	BOTTOM 31	TDO	UPADPNIOX
732	–	m*	VDLXX	BOTTOM 30	VDL	U2PADVDLDBL
733	–	K32	I	BOTTOM 29	K0_XQL	U2GSCCBS2
734	–	E35	O	BOTTOM 28	K0_DRRL	U2GSCCBS2
735	732	m*	VDL	BOTTOM 27	VDL	U2PADVDLD
736	–	C35	O	BOTTOM 26	K0_PENDL	U2GSCCBS2
737	–	m*	DGND	BOTTOM 25	DGND	U2PADGNDD
738	–	L31	I	BOTTOM 24	K0_RETRYL	U2GSCCBS2
739	–	P28	I	BOTTOM 23	K0_PACKL	U2GSCCBS2
740	–	N29	I/O	BOTTOM 22	K0_AD[0]	U2GSCPBS2
741	–	H32	I/O	BOTTOM 21	K0_AD[1]	U2GSCPBS2
742	737	m*	DGND	BOTTOM 20	DGND	U2PADGNDD
743	–	D34	I/O	BOTTOM 19	K0_AD[2]	U2GSCPBS2
744	–	m*	VDL	BOTTOM 18	VDL	U2PADVDLD
745	–	J29	I/O	BOTTOM 17	K0_AD[3]	U2GSCPBS2
746	–	E33	I/O	BOTTOM 16	K0_AD[4]	U2GSCPBS2
747	744	m*	VDL	BOTTOM 15	VDL	U2PADVDLD
748	–	J31	I/O	BOTTOM 14	K0_AD[5]	U2GSCPBS2
749	–	M28	I/O	BOTTOM 13	K0_AD[6]	U2GSCPBS2
750	–	L29	I/O	BOTTOM 12	K0_AD[7]	U2GSCPBS2
751	–	m*	DGND	BOTTOM 11	DGND	U2PADGNDDS
752	–	m*	VDD8	BOTTOM 10	VDD	U2PADVDDIOLS

753	751	m*	GND	BOTTOM 9	GND	U2PADGNDIOLS
754	752	m*	VDD8	BOTTOM 8	VDD	U2PADVDDIOLS
755	751	m*	GND	BOTTOM 7	GND	U2PADGNDIOLS
756	752	m*	VDD8	BOTTOM 6	VDD	U2PADVDDIOLS
757	751	m*	GND	BOTTOM 5	GND	U2PADGNDIOLS
758	752	m*	VDD8	BOTTOM 4	VDD	U2PADVDDIOLS
759	751	m*	GND	BOTTOM 3	GND	U2PADGNDIOLS
760	751	m*	GND	BOTTOM 2	GND	U2PADGNDIOLS
761	751	m*	GND	BOTTOM 1	GND	U2PADGNDIOLS

12. Electrical and Environmental Specifications

12.1. Chip Specification

Tables 1, 2, and 3 show the various DC and AC characteristics for UTurn. All voltages are references to V_{ss} = Ground = 0V.

Symbol	Parameter	Minimum	Maximum
V_{dd}	DC supply voltage	0 V	7 V
V_{dl}[2]	DC supply voltage	0 V	7 V
$V_{i(Vdd)}$	Input voltage (5V pins)	−0.5 V[1]	V_{dd} + 0.7 V
$V_{i(Vdl)}$	Input voltage (3.3V pins)	−0.5 V[1]	V_{dl} + 0.7 V
P_d	Power dissipated	−	8 W (?)
T_{stg}	Storage temperature	−40° C	125° C

[1] V_i less than −0.5 may forward bias the input clamping diode on UTurn
[2] V_{dl} must always be less than V_{dd} or the ESD protection SCRs will fire and destroy the IC.

Table 1 Absolute maximum ratings

Symbol	Parameter	Minimum	Maximum
V_{dd}	DC supply voltage	4.845 V	5.355 V
V_{dl}	DC supply voltage	3.14 V	3.46 V
T_a	Operating ambient temperature	0° C	70° C

Table 2 Recommended operating conditions and DC Characteristics

Symbol	Parameter	Max	Units
$C_{in\ gsc+}$	Input Capacitance − GSC+ pins	3	pF
$C_{in\ rw}$	Input Capacitance − Runway pins	4	pF
$C_{in\ pdc}$	Input Capacitance − PDC pins	5	pF

Table 3 Signal Pin Input Capacitance

12.2. Individual Pin Specification

UTurn's pins fit into the following categories for timing and electrical specification purposes:

- **Runway Address/Data :** *ADDR_DATA[0:63]*.
- **Runway Control :** *ADDR_VALID*, *DATA_VALID*, *CLIENT_ID*, *CLIENT_OP[0:2]*, U2_ARB_IN, U2_ARB_OUT, STOP_MOST_IN, STOP_MOST_OUT, STOP_IO_OUT, TRANS_ID[0:5], MASTER_ID[0:2], COH0[0:1]. COH1[0:1].

- **Runway Parity :** *AD_PAR[0:1]*.
- **Runway Misc :** *POWER_ON*.
- **GSC+ IOA0 Control :** *K0_ADDVL, K0_READYL, K0_TYPE[0:3], K0_ERRORL, K0_INTERRUPTL, K0_RESETL, K0_BRL[0:5], K0_BGL[0:5], K0_LSL, K0_PACKL, K0_PENDL, K0_XQL*.
- **GSC+ IOA1 Control :** *K1_ADDVL, K1_READYL, K1_TYPE[0:3], K1_ERRORL, K1_INTERRUPTL, K1_RESETL, K1_BRL[0:5], K1_BGL[0:5], K1_LSL, K1_PACKL, K1_PENDL, K1_XQL*.
- **GSC+ Address/Data :** *K0_AD[31:0], K1_AD[31:0].*
- **GSC+ Parity :** *K0_PARITY, K1_PARITY.*
- **PDC I/O's :** *PDATA[0:7]*
- **PDC Outputs :** *PADDR[0:20], PADDRVL, PWEL, POEL*
- **Test Signal Inputs :** TDI, TMS, TRST
- **Test Signal Outputs :** TDO
- **Asynchronous Inputs :** *TOCL,* PFAIL_WARNING_L, ARESETL
- **Chassis Outputs :** *FP_CLK, FP_DATA.*
- **System Clocks :** *SYNC_K0_H, SYNC_K0_L, SYNC_K1_H, SYNC_K1_L, SYNC_IOA0_H, SYNC_IOA0_L, SYNC_IOA1_H, SYNC_IOA1_L, SYNC_RW_H, SYNC_RW_L*

Table 4 shows the required set-up, hold and propagation times for some of the previous groups of UTurn pins.

Symbol	Description	Minimum	Maximum
	Input Requirements		
T_{su1}	Runway I/O's set-up time to SYNCH Rising edge	1.5 ns	–
T_{h1}	Runway I/O's hold time from SYNCH Rising edge	0 ns	–
T_{su2}	GSC Data set-up time to SYNCH falling edge	1.5 ns	–
T_{h2}	GSC Data hold time from SYNCH falling edge	4.5 ns	–
T_{su3}	GSC Control set-up time to SYNCH falling edge	1.5 ns	–
T_{h3}	GSC Control hold time from SYNCH falling edge	4.5 ns	–
T_{su4}	GSC Parity set-up time to SYNCH falling edge	1.5 ns	–
T_{h4}	GSC Parity hold time from SYNCH falling edge	4.5 ns	–
	Output Timing Specification		
T_{p1}	PDC I/O's delay from GSC+ IOA0 SYNCH rising	–	31 ns
T_{p2}	Runway Data delay from SYNCH rising edge	–	1 ns
T_{p3}	Runway Control delay from SYNCH rising edge	–	1 ns
T_{p4}	Runway Parity delay from SYNCH rising edge	–	1 ns
T_{p5}	GSC Data delay from SYNCH rising edge	–	3 ns
T_{p6}	GSC Control delay from SYNCH rising edge	–	3 ns
T_{p7}	GSC Parity delay from SYNCH rising edge	–	3 ns

Table 4 I/O Timing Specification

Table 5 shows some additional electrical pin specifications.

Symbol	Description	Minimum	Maximum
	PDC I/O's		
V_{oh}	high−level output voltage @ I_{oh} = −5 mA	2.4 V	–
V_{ol}	low−level output voltage @ I_{ol} = 30 mA	–	0.5 V
V_{ih}	high−level input voltage	2 V	–
V_{il}	low−level input voltage	–	0.8 V
I_{oh}	high−level output current	−5 mA	–
I_{ol}	low−level output current	30 mA	–
I_{ih}	high−level input current	–	10 μA
I_{il}	low−level input current	–	−10 μA
	PDC asynchronous, Chassis outputs, and Test Output		
V_{oh}	high−level output voltage @ I_{oh} = −5 mA	2.4 V	–
V_{ol}	low−level output voltage @ I_{ol} = 30 mA	–	0.5 V
V_{ih}	high−level input voltage	2 V	–
V_{il}	low−level input voltage	–	0.8 V

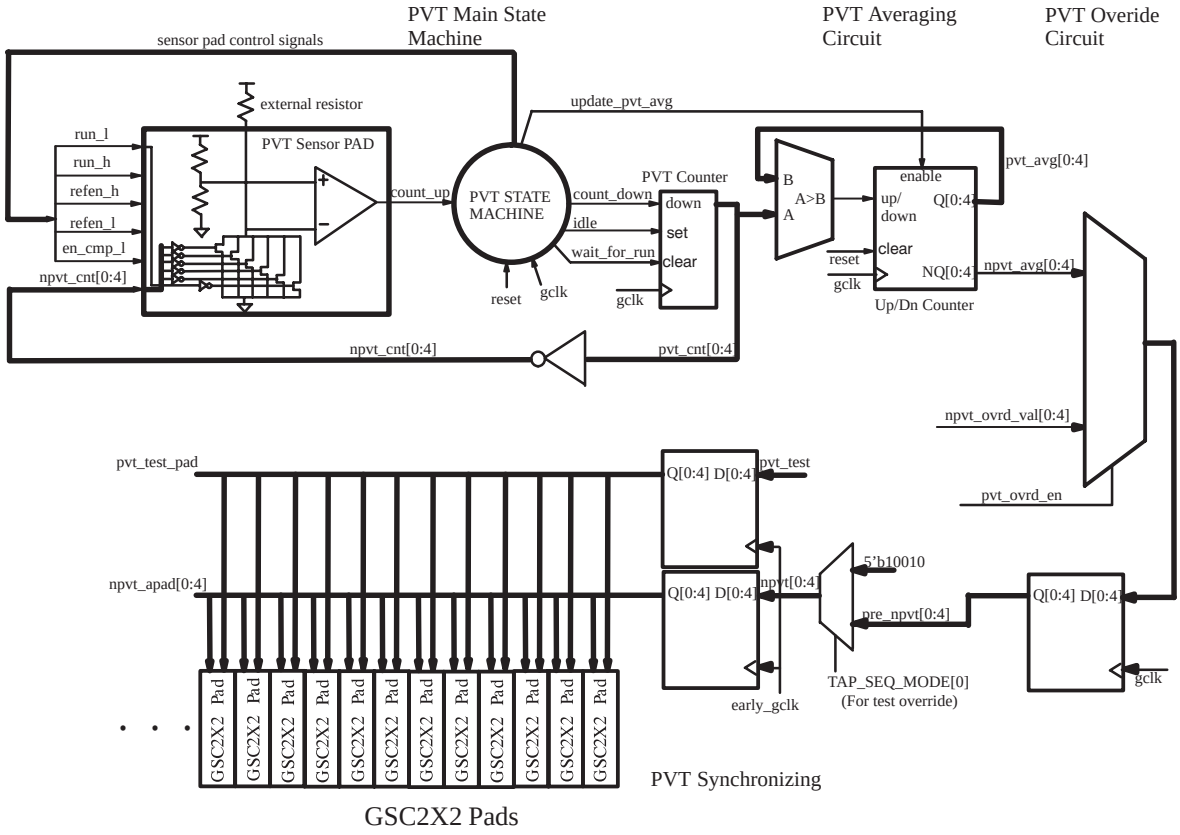
Symbol	Description	Minimum	Maximum
I_{oh}	high−level output current	−5 mA	–
I_{ol}	low−level output current	30 mA	–
I_{ih}	high−level input current	–	10 µA
I_{il}	low−level input current	–	−10 µA
Runway Address/Data, Runway Control, Runway Parity I/O Specs			
V_{oh}	high−level output voltage @ I_{oh} = max	2.9 V	–
V_{ol}	low−level output voltage @ I_{ol} = max	–	640 mV
V_{ih}	high−level input voltage	2.0V	–
V_{il}	low−level input voltage	–	0.8V
I_{oh}	high−level output current @ $V_{oh} = V_{oh\ min}$	27mA	–
I_{ol}	low−level output current @ $V_{ol} = V_{ol\ min}$	36mA	–
I_{ih}	high−level input current	–	10uA
I_{il}	low−level input current	–	− 10uA
GSC Address/Data, GSC Control, GSC Parity I/O Specs			
V_{oh}	high−level output voltage @ I_{oh} = max	2.9 V	–
V_{ol}	low−level output voltage @ I_{ol} = max	–	640 mV
V_{ih}	high−level input voltage	2.0V	–
V_{il}	low−level input voltage	–	0.8V
I_{oh}	high−level output current @ $V_{oh} = V_{oh\ min}$	27mA	–
I_{ol}	low−level output current @ $V_{ol} = V_{ol\ min}$	36mA	–
I_{ih}	high−level input current	–	10uA
I_{il}	low−level input current	–	− 10uA
GSC SyncH, SyncL Clock Input Specs			
V_{ih}	high−level input voltage	3.8V	–
V_{il}	low−level input voltage	–	3.2V
I_{ih}	high−level input current	–	5.0mA
I_{il}	low−level input current	–	5.0mA

Symbol	Description	Minimum	Maximum
SYNC_IOA0, SYNC_IOA1,SYNC_RW Clock Input Specs			
V_{ih}	high−level input voltage	.8 V	–
V_{il}	low−level input voltage	–	0.2 V
I_{ih}	high−level input current	–	−5ma
I_{il}	low−level input current	–	−5ma

Table 5 Individual Pin Electrical Specification

12.3. GSC2X PVT Description

PVT Block Diagram



12.3.1. Overview of PVT Design

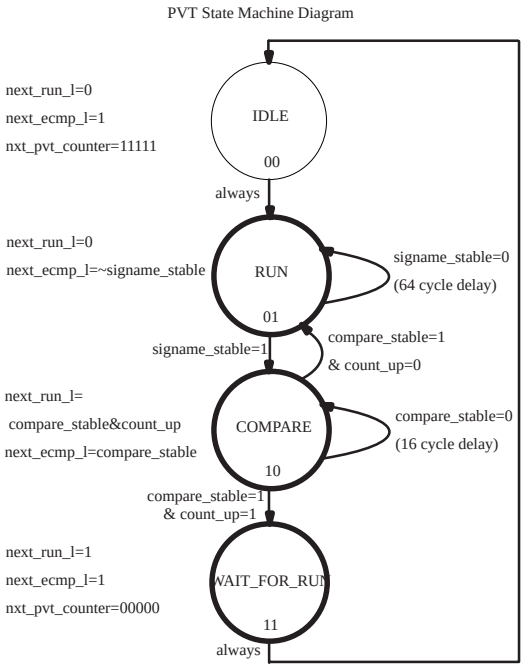
The GSC2X2 pads (a.k.a. pad driver/receiver) are designed to drive a high frequency PC board bus. Past designs have optimized the driver circuits in the pads to drive buses well for nominal conditions and process but have performed marginally at the extremes of process varation and environmental conditions. PVT stands for Process Voltage Temperature compensation. The PVT design is used to maintain the output impedance of the pads over a much wider range of process change and operating conditions. The PVT design consists of three distinct blocks residing in different locations: the PVT sensor Pad, the PVT controller, and the GSC2X2 pads. The PVT sensor pad connects to an external resistor tied to VDD, and the digital output of the PVT sensor Pad and the PVT controller programs the GSC2X2 Pads' output impedance to be approximately one tenth the value of the external resistor. The PVT sensor pad and GSC2X2 pads are located in the pad ring, while the PVT Controller circuit resides in the standard cell core of the chip.The GSC2X2 IO pad is programmable to be either an address/data/type/parity pad or a control pad (the control pad has the added feature of self−timed tristating one cycle after the output deasserts). In addition to this normal functionality, the drive strength of the pad is programmable via a five−bit input port called npvt[0:4]. The inverted binary value of npvt[0:4] controls the effective size of the output drive FETs by parallelling FETs of

binary−weighted sizes. This mechanism effectively provides binary control of the output FET size. For a slow process and high temperature, these bits should all be asserted (0). For a fast process and low temperature these bits should all be be deasserted (1). (When all bits are deasserted (1), there is a single FET that is still turned on of course. This FET may also be turned off for test purposes by asserting the PVT_TEST signal, allowing each parallel FET to be tested in turn.) The PVT sensor Pad is the analog portion of the design. It is implemented in the two pads called PVTPADR and PVTPADG, (Due to the U2 compatibility model, the PVTPADs for ioa0 and ioa1 differ slightly in the implementation of test circuitry). The PVTPADR/G is connected to an external resistor which is pulled up to VDD. The sensor circuit consists of a comparator which has a VDD to GND voltage divider comprised of diffusion resistors as a reference input, and another VDD to GND voltage divider comprised of an external resistor and an internal nfet network as the other input. The nfet network is a 1/10 scaled down replica of the nfet network in the actual driver pads and is composed of six parallel nfets of specific sizes. Five of the six nfets are sized with a binary weighting (e.g. 24, 12, 6, 3, 1.5), and are turned on or off with the npvt[0:4] lines. The sixth nfet is a base value which is always on during normal operation.The final block is the PVT controller, which is responsible for controlling the analog section and updating the npvt[0:4] value. The digital logic is implemented in standard cell as part of the core logic (pvt.v). The PVT state machine is responsible for controlling what is basically an A/D converter comprised of the PVTPAD and a counter in the control logic block.The digital control logic also receives input from an overide register, which provides the ability to overide the analog circuit with a programmable value for the npvt[0:4] value, which is output to the GSC2X2 pads. The calculated value of npvt[0:4] is readable from this same GSC_PVT_OVERRIDE register.

12.3.2. PVT Controller State Machine

The PVT Controller state machine is responsible for creating the five−bit digital signal to control the output impedance of the pads: npvt[0:4]. The signal is created using an A/D converter comprised of the comparator and nfet network in the PVT Sensor PAD and a counter in the pvt standard cell block. The counter and comparator are controlled by the PVT Controller State Machine. The inverted output of the counter is npvt_cnt[0:4], which is sent to PVTPADR/G to control the nfet network. The resistance of the nfet network forms the pulldown portion of a voltage divider, while the external resistor forms the pullup. The resulting voltage seen at this divider is compared against a reference voltage (VDD/2), and if the resulting voltage is too high (count_up=0), the PVT controller state machine will decrement the value of pvt_cnt. This continues until the resistance of the nfet network matches that of the external pullup, resulting in a value of count_up of 1. This results in an npvt_cnt[0:4] value that causes the nfet network in the pads to have an output impedance of roughly 1/10 that of the external resistor.The output of the PVT Controller state machine counter does not go directly to the GSC2X2 pads. The PVT average register is updated each time the state machine calculates a new value of pvt_cnt[0:4] that satisfies the condition of matching the external resistance. The value of the PVT average register pvt_avg[0:4] can only change by plus or minus one value for each new value of pvt_cnt[0:4] that is calculated to match. The average register acts as a low−pass filter to prevent a noise problem from causing a large change in output impedance from time to time. The initial value of pvt_avg is centered to shorten the time required to reach the first correct value at power−on.The PVT Controller State Machine diagram follows. (The state machine was leveraged from the Halcon PVT controller, and some of the naming conventions refer to functions which no longer exist, such as

WAIT_FOR_RUN. Halcon only ran PVT updates during DRAM refresh while UTURN updates constantly.)



1) pvt_counter[0:4] = 11111, assert run_l and run_h outputs (enable PVT Sensor Pad reference and scaled−down pad model).

2) Wait 64 cycles for pad voltage to stabilize (SIGNAME_STABLE=1).

3) Assert en_cmp_l signal to enable comparator.

4) Wait 16 cycles for output of comparator to stabilize.

5) If count_up is not asserted then decrement pvt_counter by 1 and de−assert en_cmp_l. Repeat steps 3 & 4 until count_up is asserted or until pvt_counter reaches a count of 00000.

a) If a value of pvt_counter is found to match the transistor network representing the pad impedance to the external resistor, update the pvt avg register. The pvt_avg value is incremented if the pvt_counter register is greater then the pvt_avg register. The pvt_avg register is decremented if the pvt counter is less then the pvt_average register. The pvt_avg register remains unchanged if the value matches the pvt_counter.

b) If the minimum value of 00000 is reached and count_up is not asserted, use the default value of pvt_ovrd_val[0:4] from the pvt overide register.

12.3.3. PVT Override

The analog circuit of the PVT design can be overridden by setting the PVT_OVERIDE_ENB bit and programing the PVT_OVERIDE field in the GSC_PVT_OVERRIDE register in UTURN. A value of all ones

in the PVT_OVERIDE field sets the IO pad drivers to maximum drive (minimum output impedance) and a value of all zeroes sets the IO pad drivers to minimum drive (maximum output impedance). The current value of the PVT Average register can be also read from the GSC_PVT_OVERRIDE register.

12.3.4. GSC2X2 Pad Driver

Within the realm of the PVT control system the GSC2X2 pad drivers are slaves only. That is, the GSC2X2 pads are not providing any feedback to the PVT control. At any fixed process or temperature the GSC2X2 pad output impedance may be programmed to be one of 32 different values. The value of npvt[0:4] seen at the ports of GSC2X2 will determine which impedance value the pad is programmed to have.The size of the pull−down parallel network of nfets in GSC2X2 are a 10X replica of the pull−down nfet network in the sensor pad (PVTPADR/G) that is connected to the external resisitor. When the PVT control system converges upon a value of npvt[0:4] that matches the impedance of the nfet network in PVTPADR/G to the external resitor, this same value of npvt[0:4] driven to GSC2X2 will produce an impedance 10 times less than the external resistor value. EXAMPLE: Rexternal = 500ohm, will program Routput(GSC2X2) = 50 ohmGSC2X2 has both the ability to pull−down and pull−up the GSC bus connected to UTURN. The PVT control circuit however only closes the loop around a 1/10th replica of the GSC2X2 pull−down fets. Thus, the nfet source followers fets in the pull−up section of the pad driver are scaled relative to the pull−down devices. The range of output impedance of GSC2X2 was chosen such that over the expected variation in both process and temperature the pad driver could be progammmed to be any value between 40 and 55 ohm. This range corresponds to range of 400 to 550 for the external resistor. The value of npvt[0:4] seen at GSC2X2 is updated synchronously every positive phase of CKA driven at the pad. This means that new values of npvt[0:4] must set up before the rising edge of CKA (setup: 0nsec) and hold to the falling edge of CKA(holdtime: 1nsec from falling edge of CKA).

12.3.5. PVT Test

The GSC2X2 pad drivers are all connected in parallel. In order to test the ouputs individually, the PVT bits (npvt[0:4]) must be asserted one at a time, and the outputs driven to show that each FET is working. The “base value” FETs, which are used when all bits of the npvt[0:4] bus are deasserted must be disabled in order to test each FET independently. Asserting the PVT_TEST bit disables the “base value” FETs, allowing test-ability of the parallel drivers.The test for the GSC2X2 pads follows this sequence:The core is set up to drive the GSC2X2 pads out low and then high using scan. The npvt[0:4] bus is deasserted (11111)The pads are tested for driving low then high.The PVT_TEST bit is asserted (turning off the base value drivers)The npvt[0:4] bus is asserted one bit at a time, and the pads are tested for driving low then high.Finally, the npvt[0:4] bus is deasserted (11111), with the PVT_TEST bit still asserted, disabling the pads completely. The pads are then tested as before, and measured to assure that they are NOT driving low then high to assure that there are no bits stuck on.

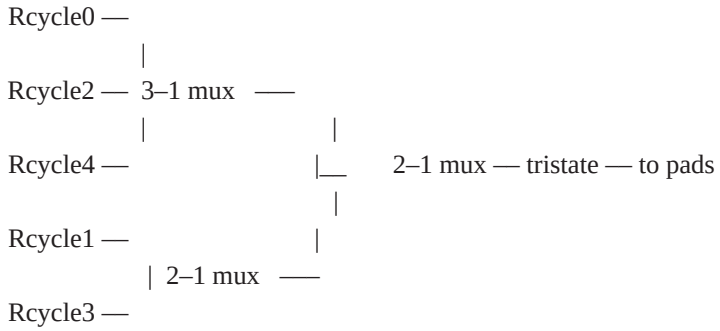
13. UTurn internal timing and physical implementation

This chapter contains information about internal timing, critical signals that require special attention during physical implementation and opportunities to improve the performance of UTurn by removing cycles of latency. This chapter also has details on UTurn's known bugs and anomalous behaviors.

13.1. Multi–cycle paths

UTurn's goal is to meet the clock period for all FF to FF paths in the same clock domain. This will allow universal application of the double strobe timing technique. In U2, there were a number of paths to and from custom modules (or pads) that were multicycle paths. The U2 paths are listed below.

- Power_onI(falling) to FFs.
  There is an asynchronous path from this pad input falling to all versions of reset, this creates lots of timing violations. However since we are being reset this should be ok. The rising edge of Power_onI is synchronized to the various clock domains and there are no timing violations.
- R2clk(TLB address) to TLB RAM.
  The TLB RAM access for reads only takes one cycle to get the data out of the RAM, and then one cycle to get the valid data to the inQctrl and inbDpath. Thus reads take two cycles. inQctrl generates two signals called inQ_valid_delayed and inQ_valid_delayed_two to make sure the data is not used until valid. inQctrl then relays this to the inbDpath via control signals. TLB RAM writes take two cycles to get the data hold time correct. In the first cycle the read_writebar line is pulled low to write, and data is drive to RAM. In the second cycle read_writebar is returned to read state, and data CONTINUES to be driven to get correct hold time.
- R2clk FFs to iopad(ad_parO[0:1]).
  These paths have multiple cycles to go through the parity tree, only the final mux and tri–state driver are on the critical path. This path is similar the the critical path for the adO[0:63] going from the ioa to the pads(see next item).
- R2clk FFs to iopads.
  These FFs are set up before Rstart_arb is signaled. There is a good picture of this with timing in chapter 6 in the section: TWO to One Interface . The muxes are structured as follows:



The muxes are controlled such that first level of muxes is set up the previous cycle. The critical path is only from the select of the last 2–1 mux through the tri–state to the pads. The select for the last 2–1 mux is reg_pntr[5] – 0 selects an odd word, 1 selects an even word.

- OutQ to gclk FFs.
  These are all 2 cycle paths except for Data_out[27] to gmaster_state machine. Data_out[27] path must be 4ns or less, the rest of OutQ has 20ns(gclk period) + 4ns
- gclk to PDATA pads.
  The timing for gclk to PDC Data pins is not critical, only gclk to GSC bus pads.
- RRQ entries to gclk FFs
  These are not real timing paths due do the the synchronizer and the valid bit. The data is written to the RRQ entry several clocks before it is read from the RRQ. This (may) shows up in veritime when looking at timing paths from iopad(R2clk) to gclk storage or in timing paths from R2clk FFs to gclk FFs.

13.2. Other timing/routing information

The following is a list of U2 timing/physical routing items that could be confusing and need to be kept in mind. The UTurn design will need to evaluate how to handle each of these items.

- Squashing k_InQ_load early.
  There is a special gate to squash k_InQ_load signal going to the InQ datapath very early in the cycle after the InQ is loaded(k_InQ_load active). This is to speed up the falling edge since that was much more critical – to prevent writing to incorrect queue locations.
- inQout to hpa_reg_out is near the limit of R2clk period.
  This is really a multicycle(>=2 cycles) path but to allow universal double strobe met the R2clk period.
- Narrow power signals were added through the top routing channel to divide channel – this should eliminate any possibility of inductance problems.
- There were many driver size changes to the netlist after synthesis (hand in place optimization) – these changes are not reflected in the constraint file or the verilog.
- The Runway arbitration block (u2_arb) was a very challenging block. One of the most critical paths is to the runway_drive signal valid. The diming analysis tools did not predict this timing path correctly because of the parallel drive fights seen by runway_drive due to weak drive feedback inverters in the Runway pads.
- The placement of R2clk relative to ckrw is VERY important.
  The longest path from the pads(ckrw rising) to the ioa R2clk FFs is about 4.5ns. This implies that:

  (external skew between ckrw and R2clk) +
  Minimum R2clk tree insertion > 4.5ns

  The number of signals that must go from R2clk FFs to ckrw domain are very limited. They

are:

1) Rstart_arb – from IOA to U2_arb
2) Trans_len[0:2] – from IOA to U2_arb
3) Rcoh[0:1] – from IOA to ckrw FF within IOA
4) Rdata_return_in – from IOA to U2_arb
5) stop_most_out – from IOA to U2_arb

Currently the FFs in the IOA are all clocked off the R2clk at the end of the clock tree. The output of the FF goes directly to the ckrw FF (except for Rcoh – there is a 2–1 mux). This implies that:

(external skew between ckrw and R2clk) +
Maximum R2clk tree insertion + FF delay + Mux delay < 8.33ns

To make the above requirement easier to meet in UTuirn, the list of above signals could be clocked from an earlier version of R2clk (no clock tree), removing about 2ns from the above equation. The Rcoh FFs were clocked on a delayed version of ckrw since the timing to Rcoh R2clk FFs was already quite critical.

13.2.1. Routing details

This is a list of signals that must be routed carefully. Usually this means a low resistance path and/or wider metal for electromigration.

- R2clk0,R2clk1 – outside and inside IOA – from pad to IOA – to first level of clock buffers.
- nR2clk0,nR2clk1 – outside and inside IOA – from pad to IOA to Datapaths
- ckrw – outside and inside IOA – from pad to IOA
- very_early_r2clk2, very_early_r2clk1 – inside IOA – EM
- runway_drv1 and runway_drv2 – from arb to pads. All 3 output ports of the SPBFK3 must be connected to wide metal (signal needs very low resistance)
- ioa0_select , ioa1_select – from arb to inside IOA. All 3 output ports of the SPBFK3 must be connected to wide metal – this needs to be VERY low resistance
- from left side and right side runway pads to IOA – low resistance on signals from pad to core
- Power_onI – from the POWER_ON pad to the point where the signal divides to go to both IOAs. There is a large C on this signal.
- from k1_AD[31:21] pads to core – this was only necessary because in U2 the port order was reversed on the IOA
- GSC clock signals – k0_ckb and k1_ckb
- Tap signals – due to the very large load on TAP signals used by the runway pads a buffer was placed near the arb block to reduce the RC problems.
- All Double strobe signals should be treated like clocks – pre–route in M3 at the source

- R2clk_DP_N to outbnd Datapath – this clock has more than 5pF load – it needs to be routed in M3 or wide M1,M2 for electromigration
- Force NSPBFJs (tri–state drivers in ioa) to be closer to top of IOA with ports directly above them.
- The addr_dataI and addr_dataO buses were interspersed and routed with wider metal and wider spacing, reducing coupling impacts (since the buses switch at different times within the runway bus period).
- The eight signals from FFs that cross from R2clk to ckrw domains should be kept very short. For most of the signals this means placing the FFs close to the arb block. Rcoh FFs should be in a location close the RCOH pads.

Other misc routing details

- The k_ad pin order was reversed from the pad order on the IOA. This was due to a connection at the top level from k_adI[0:31] to k0_adI[31:0]. This bus swapping was caused by GSC bus notation that is not consistent with HPPA convention.
- The pin order of the addr_dataI and addr_dataO buses is opposite from U2. We used the IOA furthest from the pad to determine the order, keeping the addr_dataI and addr_dataO buses above the outbnd Datapath and inbnd Datapath respectively. Reversing the pin ordering will reduce the top level capacitances of these signals.

There is a set of regions that was defined to help the placer. This was done for several reasons. The most important reason was to make sure all R2clk and ckrw clock domain stuff was above the datapaths and all gclk domain stuff was below the datapaths. The rest of the region definition was to help the placer put things into lesser congested areas, place spare gates in appropriate places, etc.

13.3. Signals that clock tree synthesis was used on

The following is a list of signals that clock tree synthesis was used to buffer the signal within the IOA. R2clk and BGclk are the most critical for maintaining clock skew and insertion delay. The total clock skew goal for these signals is ~300ps. This guarantees no race conditions between FFs because the minimum time from CLK to Q on the FFs is just over 300ps. The name in () is the name that occured at the base of the tree after clk tree synthesis.

- R2clk (R2clk_bin) – 2 levels of buffering, 1st level is NSPBFK, 2nd level is NSPBFJ – ~2.4ns insertion
- BGclk (gclkX) – 2 levels of NSPBFJ's – ~2ns insertion
- Rreset (RresetX) – 1 level of NSPBFJ

The following all have 2 levels of NSPBFJ's:

- TAP_ML_TCK_Bioa1 (IOA_ML1)
- TAP_DBL_STRBioa1 (IOA_DS1)
- TAP_INT_SHIFTioa1 (IOA_ACLK1)
- TAP_INT_NSHIFTioa1 (IOA_BCLK1)
- TAP_ML_TCK_Bioa2 (IOA_ML2)
- TAP_DBL_STRBioa2 (IOA_DS2)

- TAP_INT_SHIFTioa2 (IOA_ACLK2)
- TAP_INT_NSHIFTioa2 (IOA_BCLK2)

13.4. Timing opportunities

There are 3 potential places to pull out an R2clk cycle of latency. They are:

1) TLB RAM currently receives an inverted (early/late?) version of R2clk. This made address setup easier, but read data comes out later. The TLB RAM could receive a normal R2clk, and the generation of tlb_hit could then be done in the same cycle as read data coming out of RAM. This would require significant changes to inQctrl (see multicycle paths above), and the TLB RAM BIST logic. It would allow inQctrl to branch out of ”IDLE” one cycle earlier for EVERY transaction (however see 2 below also).

2) The inQctrl state machine was designed to be pipelined to hide some of the TLB RAM access to tlb_hit latency. When we combined the inbound RAM with the inbound queue, we killed this pipeline effect. Since there is a staging register for inQout, even if the queue is FULL (or has multiple transactions in it) the inQ_valid that comes from sychro.v must be de–asserted for the time it takes to load the staging register. So the pipeline never appears to be full due to the added latency of the staging register. The obvious fix here is to separate the inbound RAM and queue so that the staging register can be eliminated. This actually has two effects. First, the latency of the staging register is eliminated (one cycle!). Second, the synchronizer pipe now appears to stay full (when multiple transactions are in the queue), and the inQctrl pipelining will actually do some good (one more cycle savings when multiple transactions are in the queue – though it does depend on the type of transactions in the queue). Both 1 and 2 together are NOT completely additive improvements. The pipelining in inQctrl counts on the latency of TLB RAM accesses, and attempts to hide one cycle worth. If the TLB RAM data to tlb_hit latency is reduced, then inQctrl would have to be re–written to take advantage of the ’new’ pipelining delays. Both 1 and 2 together do provide an incremental advantage (bv: I think; I haven’t gone through this completely, just looking at my notes, and thinking on the fly).

3) Allow reading and writing to the InQ at the same time. (Hani H. to finish details)

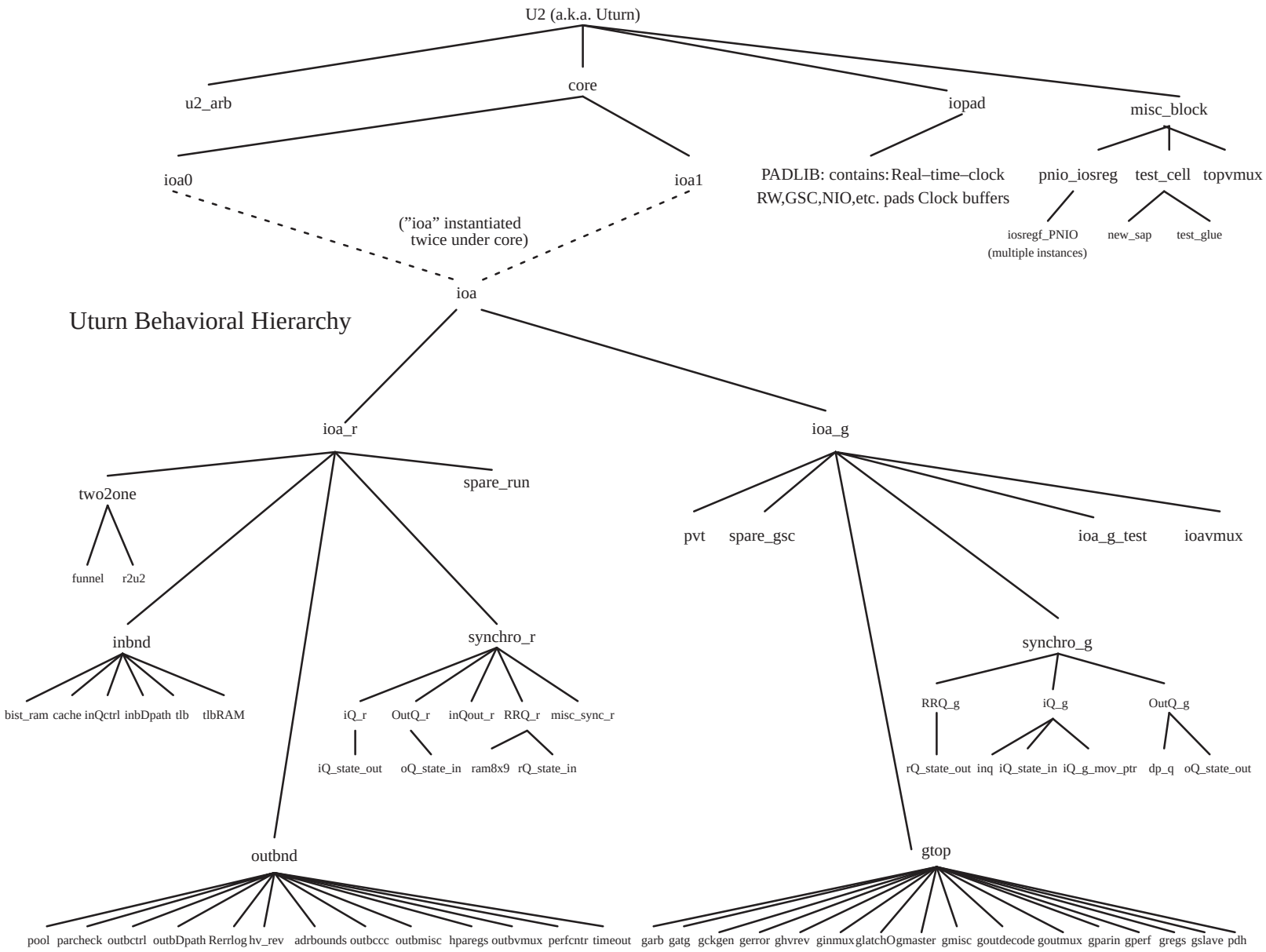
13.5. UTurn Implementation Details

13.5.1. Behavioral Verilog Hierarchy

On the following page is the hierarchical diagram of Uturn. This hiearchy is valid for the behavioral files only. For the netlist, we flatten everything under ioa_g into a single level, and also we flatten ioa_r into a single level. The “core” block is removed. This creates a chip with only the blocks iopad, u2_arb, ioa0_g, ioa1_g, ioa0_r, ioa1_r, and misc_block for the netlist. In our design methodology, all verilog files are found in directories with the same name, and all are at the same UNIX directory level. In other words, if one wants to find the block “u2_arb,” one should look under vlog/u2_arb/u2_arb.v. The block “pdh”, at a completely different **logical** hierarchy, is still found in the UNIX directory vlog/pdh/pdh.v. Because there is no hierarchy information imparted by the UNIX directories, only by the verilog instantiations, this diagram is a helpful guide.

13.5.2. TRST used in Real Time Clock circuit

The TRST signal, a reset signal for the Test Access Port (TAP), is also used as an input to the real time clock (RTC) to prevent false writes to the RTC counter/register during power–up. TRST is a low–true


U2 (a.k.a. Uturn)
u2_arb
core
iopad
misc_block
ioa0
ioa1
PADLIB: contains: Real–time–clock
RW,GSC,NIO,etc. pads Clock buffers
pnio_iosreg
test_cell
topvmux
iosregf_PNIO
(multiple instances)
new_sap
test_glue
("ioa" instantiated
twice under core)
ioa
Uturn Behavioral Hierarchy
ioa_r
ioa_g
two2one
spare_run
pvt
spare_gsc
ioa_g_test
ioavmux
funnel
r2u2
inbnd
synchro_r
synchro_g
bist_ram
cache
inQctrl
inbDpath
tlb
tlbRAM
iQ_r
OutQ_r
inQout_r
RRQ_r
misc_sync_r
RRQ_g
iQ_g
OutQ_g
iQ_state_out
oQ_state_in
ram8x9
rQ_state_in
rQ_state_out
inq
iQ_state_in
iQ_g_mov_ptr
dp_q
oQ_state_out
outbnd
gtop
pool
parcheck
outbctrl
outbDpath
Rerrlog
hv_rev
adrbounds
outbccc
outbmisc
hparegs
outbvmux
perfcntr
timeout
garb
gatg
gckgen
gerror
ghvrev
ginmux
glatchO
gmaster
gmisc
goutdecode
goutmux
gparin
gperf
gregs
gslave
pdh

asynchronous reset signal, and should be low during the period of time that power rails are coming up and clocks are starting up. TRST should be negated (driven high) to UTurn only after all external power rails are within tolerance and after external clocks have been oscillating within expected voltage and frequency limits for several clock cycles. Internal to UTurn, the received version of TRST is connected to the RTC's chip select input such that if TRST is high (negated), accesses of the RTC counter/register are enabled. It should be noted, then, that TRST must not be tied low external to UTurn. If TRST does not go high, accesses to the RTC value will be impossible. In Kitty Hawk systems, TRST is actually tied to the ARESETL signal external to UTurn. ARESETL comes from the power supply, and is driven high after all voltages are in tolerance (and well after clocks have begun oscillating).

13.6. UTurn Known Bugs and Anomalies

13.6.1. Dual ERRORLs at reduced Runway:GSC clock frequency ratios (SABE bug S188)

When a GSC guest device masters a connected read transaction on GSC and, in the process of fetching the read data, UTurn encounters an error in the Runway clock domain, UTurn is supposed to assert ERRORL to the guest, informing it that the transaction cannot be completed successfully. As frequency ratios between the Runway ckrw clock and the GSC GCLK decrease below 3:1, UTurn will assert ERRORL twice in such error circumstances. Devices in existence today as well as all devices that comply with the GSC specification should not have any trouble with this dual ERRORL assertion; the second ERRORL is ignored. The fix for this bug is in the gerror block, which should use a signal from gslave called conn_rtn_valid instead of the currently used signal k_conn_rtn_valid. Signal conn_rtn_valid is generated in the gslave block but is not currently ported out of that block.

13.7. Next Time

There are always things in a design that could have been done differently to improve the design from one perspective or another. The UTurn design is no exception. In this section are documented the things we'd consider changing in a "Son of UTurn" design.

13.7.1. Cache line writes to I/O space

For graphics performance improvement, the single most effective extension to the UTurn design would be to accommodate a cache–line sized write to I/O space. Obviously we'd need a processor that could generate such a transaction (no such PA processor currently exists). Absent this hurdle, UTurn could be modified to accept a cache line sized write to I/O space with its existing queue structure with several key modifications. First, the write transaction would have to be broken into3 or 4 OutQ entries. In the 4 entry option, the line–sized write would be made to look like 4 separate coalesceable write_short transactions. This may not be feasible from a timing perspective, but if it could be accomplished, no GSC logic would have to be changed in UTurn. An alternative scheme for accommodating line–sized I/O writes would store the line–sized write in 3 specially–formatted OutQ entries. The first entry being the "header" entry, possibly containing 2 words of data, and the next 2 entries would be data–only entries. With this 3–entry option, the GSC logic would need to be modified to accommodate the new OutQ entry type(s).

In addition to the OutQ entry organization issues, the OutQ full indicator (which translates to STOP_IO on Runway) would likely have to be changed such that a full OutQ was indicated earlier, given the possibility that 4 OutQ entries could be filled as a result of a single Runway transaction.

13.7.2. DMA Forward Progress improvements

UTurn implements a DMA forward progress mechanism to ensure that guests have a chance to master transactions on GSC in the face of long streams of processor–mastered I/O write traffic (as might be generated for graphics display operations). The UTurn mechanism counts OutQ entries acknowledged, allowing a guest to be granted GSC bus ownership after each 16 OutQ entries processed. Given architectural directions planned for SPPA–based graphics, a better scheme would be to count bus cycles instead of OutQ entries processed. In a future version of UTurn, it would be wise to incorporate a register that could be configured with the number of GSC clocks to wait after a guest bus request is issued until OutQ entry processing is suspended, enabling a guest to be granted ownership of the GSC bus. The register would be located in the GSC bus specific register set (suggest register offset 6), and the OutQStreamBreak logic in block garb would be changed to count GCLKs.