A2 Processor

A2-L2 Interface Description

Preliminary V1.0

February 5, 2010

© Copyright International Business Machines Corporation 2010

All Rights Reserved
Printed in the United States of America June 9, 2020

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both.
IBM IBM Logo
CoreConnect PowerISA
PowerISA logo PowerISA Architecture
RISCTrace RISCWatch

Other company, product, and service names may be trademarks or service marks of others.

The information contained in this document is subject to change or withdrawal at any time without notice and is being provided on an "AS IS" basis without warranty or indemnity of any kind, whether express or implied, including without limitation, the implied warranties of non-infringement, merchantability, or fitness for a particular purpose. Any products, services, or programs discussed in this document are sold or licensed under IBM's standard terms and conditions, copies of which may be obtained from your local IBM representative. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties.

Without limiting the generality of the foregoing, any performance data contained in this document was determined in a specific or controlled environment and not submitted to any formal IBM test. Therefore, the results obtained in other operating environments may vary significantly. Under no circumstances will IBM be liable for any damages whatsoever arising out of or resulting from any use of the document or the information contained herein.

While the information contained herein is believed to be accurate, such information is preliminary, and should not be relied upon for accuracy or completeness, and no representations or warranties of accuracy or completeness are made.

Note: This document contains information on products in the design, sampling and/or initial production phases of development. This information is subject to change without notice. Verify with your IBM field applications engineer that you have the latest version of this document before finalizing a design.

While the information contained herein is believed to be accurate, such information is preliminary, and should not be relied upon for accuracy or completeness, and no representations or warranties of accuracy or completeness are made.

Note: This document contains information on products in the sampling and/or initial production phases of development. This information is subject to change without notice. Verify with your IBM field applications engineer that you have the latest version of this document before finalizing a design.

IBM Microelectronics Division
1580 Route 52, Bldg. 504
Hopewell Junction, NY 12533-6351

The IBM home page can be found at http://www.ibm.com

The IBM Microelectronics Division home page can be found at http://www.ibm.com/chips1.1

1. A2-L2 Interface Description

1.1 Interface Naming Conventions

1. The mnemonic prefix for the L2 Nest (which includes the 4 CIUs, CXBAR, 4 L2s, BXBAR, BIU): “an”
2. The mnemonic prefix for the A2 core: “ac”
3. Prefix signals with source (driver) mnemonic, followed by an underscore, the destination mnemonic, and a second underscore.
4. Ports for an entity would be “ac_an_” and “an_ac_”.
5. When a port is connected to a net, the instance number is appended after the core prefix. For example, if a source/destination net is from/to a single one of like objects, specify it numerically (i.e. “ac1_an_”, “an_ac1_”, etc.)

1.2 Quick Links to A2Core-L2 Interface Signal Description Tables

The following table provides a set of quick links to signal description tables for those who are already familiar with the interface names and functions.

Table 1-1. Quick Links to A2Core-L2 Interface Signal Description Tables

Cross-Ref: Select to "Jump There"	Description
Table 3-3	Alphabetical Signal List [TBD when I/O Complete]
Table 1-2	Core-L2 Request Interface Description
Table 1-4	Core-L2 Store Data Interface Description
Table 1-6	L2-Core Reload Interface Description
Table 1-9	L2-Core Back-Invalidate / Snooped TLBI / CPX Status Interface Description
Table 1-10	L2-Core Reservation Interface
Table 1-11	L2-Core Sync Interface
Table 1-12	L2-Core ICBI Interface
Table 1-13	L2-Core Interrupt Presentation Interface
Table 1-14	Core-L2 Power Management Interface
Table 1-15	Core-L2 MIsc Interface Signals

1.3 Command Request Interface Description

Table 1-2 Core-L2 Request Interface Description

.

Table 1-2. Core-L2 Request Interface Description

Src	Signal Suffix	bits	cyc	Signal Description(x)
Non Type-Specific Signals		70		
core	req_pwr_token	1	r-1	**Request Power Token**. This signal indicates that a request is coming or may be coming next cycle. It is a speculative indicator that helps reduce power consumption on the command request interface.
core	req	1	r	**Request**. Indicates a valid request. This signal is a one-cycle pulse and may only be active if the core has a credit available for the request type being made (store credit or load credit).

core	req_ra	22:63	r	**Request's Real Address**. 42-bit Real Address for the request.
				For tlbivax ttype, the req_ra 42-bit field is used as follows, to pass the necessary info out to the PowerBus:
				• 22:26 - TID(3:7) (translation identifier; i.e. targeted process ID)
				• 27:51 - EPN (i.e. RB(27:51) from the tlbivax instruction)
				• 52 - TS (translation space)
				• 53 - TID(2) (translation identifier; i.e. targeted process ID)
				• 54:55 - attributes
				• 56:63 - TID(8:15) (translation identifier; i.e. targeted process ID)
				For IPI (Inter-Processor Interrupt): Carries message of IPI operation as follows:
				• 32:36 - message/interrupt type for msgsnd
				  • 00000: Type 0 - Doorbell Interrupt
				  • 00001: Type 1 - Doorbell Critical Interrupt
				  • 00010: Type 2 - Guest Processor Doorbell Interrupt
				  • 00011: Type 3 - Guest Processor Doorbell Critical Interrupt
				  • 00100: Type 4 - Guest Processor Doorbell Machine Check Interrupt.
				  • 00101-11111: undefined
				• 37:63 - message packet for msgsnd
				  • 37 - Broadcast flag
				  • =0: compare message PIRTAG to the PIR to determine if interrupt is generated
				  • =1: the interrupt is generated regardless of the value of the PIRTAG and PIR
				  • 38:41 - (reserved)
				  • 42:49 - LPID tag to be compared to the LPID register.
				  • 50:63 - PIRTAG to be compared to the PIR
				For mtspr_trace ttype, the req_ra 42-bit field is used as follows:
				• 30:31 - Core ID
				• 32:33 - Thread ID
				• 34:43 - Marker type
				• 44 - 0
				• 45 - Marker valid
				• 46 - Start trigger
				• 47 - Stop trigger
				• 48 - Pause trigger
				• 49:63 - reserved (unused)

core	req_ttype	0:5	r	**Transaction Type**, or command, associated with the request. Bit(0)=1 indicates that this request is a store-type command and needs to be written into the STQ. See Table 1-3 req_ttype(0:5) Encoding (as of v0.35) on page 9.

Key
r: cycle of request
rg: cycle of store gather indication
rt: cycle of request token (independent of request cycle)
Note (1): 32 byte l=1 loads will return reload data in two data beats

Table 1-2. Core-L2 Request Interface Description

Src	Signal Suffix	bits	cyc	Signal Description(x)
core	req_thread	0:2	r	**Thread**. Bits 0:1 - Encoded Thread ID of the requesting core. When bit 2 is a 1, it indicates that the request is associated with a DITC message. This allows DITC messages to operate independant of sync instructions.
core	req_wimg_w	1	r	**Write-through**. Request is for an address marked “Write-through”. The A2 core has a write through L1 cache.
core	req_wimg_i	1	r	**Inhibited**. Request is for an address marked “cache-Inhibited”.
core	req_wimg_m	1	r	**Memory coherence required**. Request is for an address marked “Memory coherence required”.
core	req_wimg_g	1	r	**Guarded**. Request is for an address marked “Guarded”.
core	req_endian	1	r	**Endian Mode bit** [unused in the AT Node L2 implementation] • E=0: Big Endian • E=1: Little Endian
core	req_user_defined	4	r	**User-defined bits**. U0 - U3 carry information that the user defines for these bits. The AT Node L2 defines U0 and U1 for accessing the RMT (Replacement Management Table). These bits come from the PTE.
core	req_spare_ctrl_a0	4		Spare control bits from core to L2.
L2	req_spare_ctrl_a1	4		Spare control bits from L2 to core.
Load-Specific Signals		8		
core	req_ld_core_tag	0:4	r	**Load Core Tag**. Destination unit & resource within unit (i.e. whichMQ). Used for load-ttypes only. The AT L2 design makes use of req_ld_core_tag(1:4). 00000 - 00111: Load Miss Queue Entry 0-7 01000 - 01011: I-Fetch Miss Queue, Thread 0-3 01100 - 01101: MMU Load (reserved for hardware table walker) 01110 - 01111: Unused 10000 - 10111: Load Miss Queue Entry 8-15 (for A2 with 16 total Miss Queues) 11000 - 11111: Unused
core	req_ld_xfr_len	0:2	r	**Transfer Length**. Transfer Length for non-cacheable Load Request. Indicates the number of bytes being requested for a non-cacheable load request. These have no meaning for all other requests. When used in Prism, the A2 core will not issue a 32 byle load. The A2 core also places the transfer length for non-cacheable Store requests in this field. This feature is a remnant of the similar treatment of requests. 000 reserved; 100 4 bytes 001 1 byte; 101 8 bytes 010 2 bytes; 110 16 bytes 011 (reserved - 3 bytes); 111 32 bytes(1)
L2	req_ld_pop	1	rt	**Load Pop**. One load token is being issued to the core. This token may be used for any request that is not to be placed in the STQ. The A2 core can handle up to 8 load-type credits. The actual number of credits it will handle is initialized in the A2 Core Config Ring. For the AT L2, the A2 core is initialized to begin operation with 4 credits for load-type commands. For each load-type command sent to the L2, that count is decremented. Each time this signal is asserted by the L2, that count is incremented.

Key
r: cycle of request
rg: cycle of store gather indication
rt: cycle of request token (independent of request cycle)
Note (1): 32 byte l=1 loads will return reload data in two data beats

Table 1-2. Core-L2 Request Interface Description

Src	Signal Suffix	bits	cyc	Signal Description(x)
Store-Specific Signals		5		
L2	req_st_gather	1	rg	**Store Gather**. One store queue token is being issued to the core because a store request was gathered with a previous store request. This token may be used for any request that is to be placed in the STQ. The A2 core can handle up to 32 store-type credits. The actual number of credits it will handle is initialized in the A2 Core Config Ring. For the AT L2, the A2 core is initialized to begin operation with 32 credits for store-type commands. Each time a store-type command is sent to the L2, that count is decremented. Each time this signal is asserted by the L2, that count is incremented.
L2	req_st_pop	1	rt	**Store Pop**. One store queue token is being issued to the core because a store request was processed and removed from the STQ. This may be used for any request that is to be placed in the STQ. The A2 core can handle up to 32 store-type credits. The actual number of credits it will handle is initialized in the A2 Core Config Ring. For the AT L2, the A2 core is initialized to begin operation with 32 credits for store-type commands. Each time a store-type command is sent to the L2, that count is decremented. Each time this signal is asserted by the L2, that count is incremented.
L2	req_st_pop_thrd	0:2	rt	**Store Pop Thread**. When a DITC command is removed from the store queue (via req_st_pop), this field indicates which thread's command was removed. Bit 2 of this field should be set to 0 for all other store commands that are not DITC.
Total Interface Signals		83	(=70+8+5)	

Key
r: cycle of request
rg: cycle of store gather indication
rt: cycle of request token (independent of request cycle)
Note (1): 32 byte I=1 loads will return reload data in two data beats

1.3.1 Request TTYPES

Table 1-3 shows the encoded request TTYPE definitions.

Table 1-3. req_ttype(0:5) Encoding (as of v0.35)

abc / def	000	001	011	010	110	111	101	100
000	inst fetch	(reserved) inst prefetch	-	mmu_read	-	dcbt (L2 only)[1]	dcbtst (L2 only)[1]	icbt (L2 only)[1]
001	load	lwarx/ldarx	llwarx/ldarx w/ mutex hint	-	(reserved) dcbtt	dcbt (L1/L2)	dcbtst (L1/L2)	-
011	-	-	-	-	-	dcbtls (L1/L2)	dcbtstls (L1/L2)	-
010	-	-	-	-	-	dcbtls (L2 only)[1]	dcbtstls (L2 only)[1]	icbtls (L2 only)[1]
110	-	-	(reserved) ptesync	mbar	dcbf(l)	dcbf(g)	dcbst	l1_load_hit
111	-	(reserved) pte_update	tlbi_complete	tlbsync	icbi	dcbi	(reserved) tlbi	tlbivax
101	-	stwcx./stdcx.	hwsync	lwsync	ici	dci	msgsnd	mtspr_trace
100	store	dcbz	-	ditc	icswx	icswx.	dcblc[2]	icblc[2]

Notes:
- (1): For L2 only touches, the L2 will return data to the core and the core will throw the data away. Also, for L2 only touches, the L2 will not set the L1 inclusive bits. (L2 may have a config bit that overrides and sets the L1 inclusive bits.)
- (2): icblc/dcblc will clear the lock in the L2 without knowing whether that line was instruction/data, when the L2 has a unified I/D Cache.

1.4 Store Data Interface Description

Table 1-4 Core-L2 Store Data Interface Description

.

Table 1-4. Core-L2 Store Data Interface Description

Src	Signal Suffix	bits	cyc	Signal Description
Store Data Interface Signals		289		
core	st_data_pwr_token	1	r-1	**Store Data Power Token**. A request is coming or may be coming next cycle which involves a store ttype that sends data. This power token is a subset of req_pwr_token and is used to enable the registers that will capture the st_data and st_byte_enbl signals off of the interface only when necessary. This signal is speculative: the next cycle may or may not have a valid data beat.
core	st_byte_enbl	0:31	r	**Store Data Byte Enables**. These byte enables qualify which bytes of the corresponding 32-Byte store data bus, st_data(0:255), are to be written. st_byte_enbl(0) qualifies st_data(0:7). For an L2 that uses only a subset of this data, the data and byte enables that are used should begin with st_data(0:7), qualified by st_byte_enbl(0). When the store interface is 16 Bytes wide, as defined by the XUCR L2 Store Interface Mode bit, use st_data(0:127), qualified by st_byte_enbl(0:15).

core	st_data	0:255	r	**Store Data**. 32B of store data to be written, as qualified on a byte basis by the corresponding store data byte enable bits, st_byte_enbl(0:31). st_data(0:7) is qualified by st_byte_enbl(0). For an L2 that uses only a subset of this data, the data and byte enables that are used should begin with st_data(0:7), qualified by st_byte_enbl(0). When the store interface is 16 Bytes wide, as defined by the XUCR L2 Store Interface Mode bit, use st_data(0:127), qualified by st_byte_enbl(0:15).
				For ICSWX:
				• 00:31 - ccw w/ byte_enbl(0:3)
				  • 0:7 - MSR bits
				  • 8:15 - 00 || CT
				  • 16:31 - CD
				• 32:39 - LPID (no byte_enbl)
				• 40:55 - '00',PID (no byte_enbl)
				For DITC:
				• 00:07 - 00000000 (reserved for MSR byte) (no byte_enbl)
				• 08:09 - reserved (no byte_enbl)
				• 10:15 - CT = 1 (no byte enabl)
				• 16:20 - 00000
				• 21:23 - Destination: Node ID (PBus coherency node, not AT node)
				• 24:26 - Destination: AT ID (AT instance number)
				• 27:28 - Destination: core ID
				• 29:30 - Destination: thread ID
				• 31 - 0
				• 32:39 - 00000000 (reserved for LPID) (no byte_enbl)
				• 40:55 - 0000000000000000 (reserved for PID) (no byte_enbl)
				For TLBIVAX: see Table 3-1 on page 52 for st_data usage details.

Key r: cycle of store request				

1.5 Interface Signal usage based on TTYPE

The following table indicates which of the A2-L2 Interface signals associated with command requests are used for each ttype. The following quideline

Key to Understanding the table:

- Rows are associated with the various TTYPES. Each ttype is linked into the ttype table.
- A2-L2 Interface fields are represented in the columns. Each field is linked to its signal definition.
- Cell colors and text are used to designate a variety of conditions:
  - Green cells indicate that the field is used for this ttype.
    - Text in a green field indicates signal conditions expected across the interface for that ttype.
    - Text in a green field specifies which portion of the field is used for this ttype. Not all ttypes use all bits of a multi-bit field, though they may all be supplied.
    - Text in a green field may also specify other field conditions that may cause multi-bit usage to vary. For example, if I=1 for a “load” ttype, then req_ra(22:63) are used, but if I=0 for a “load” ttype, only req_ra(22:58) are used.
  - Red cells indicate that the field is unused for this ttype.
  - Magenta cells indicate that I do not yet know whether or not the field is delivered/used for this ttype.
    - Text in a magenta field specifies the value that is expected to be sent on this field for this ttype.
  - Yellow cells indicate that the field is used under a specified condition.
  - White cells indicate that I have not yet considered the field for this ttype.
- The second column from the right is a sort aid (left-right in each row, then down the rows of the ttype table) for FrameMaker source file table.

1.5.1 A2 Core Interface Signals Sent for a Given TTYPE

For the AT Node L2, req_wimg_w, req_wimg_m, and req_endian signals are unused, the icbi ttype is nop’ed, and we will never see the l1_load_hit ttype due to the A2 Core’s Mode bit in the XUCR.

Table 1-5. Interface Signal Usage based on TTYPE

ttype	req_ttype (0:5) (abcdef)	req_ra	req_thread	req_wimg_w	req_wimg_i	req_wimg_m	req_wimg_g	req_endian	req_user_defined	req_spare_ctrl_a0	req_ld_core_tag	req_ld_xfr_len	st_byte_enbl	st_data	l-r sort	Notes
dcbf(g)	110111														38	
dcbf(l)	110110														37	
dcbi	111111														46	
dcblc2	100101				=0										63	
dcbst	110101														39	
dcbt (L1/L2)	001111				=0										14	
dcbt (L2 only)1	000111				=0										6	
dcbtls (L1/L2)	011111				=0										22	
dcbtls (L2 only)1	010111				=0										30	
dcbtst (L1/L2)	001101				=0										15	
dcbtst (L2 only)1	000101				=0										7	

Table 1-5. Interface Signal Usage based on TTYPE

ttype	req_ttype (0:5) (abcdef)	req_ra	req_thread	req_wimg_w	req_wimg_i	req_wimg_m	req_wimg_g	req_endian	req_user_defined	req_spare_ctrl_a0	req_ld_core_tag	req_ld_xfr_len	st_byte_enbl	st_data	l-r sort	Notes
dcbtstls (L1/L2)	011101				=0										23	
dcbtstls (L2 only)1	010101				=0										31	
dcbtt	001110				=0										13	
dcbz	100001	22:57			=0										58	
dci	101111														54	
ditc	100010	22:57											=0	0:55	60	st_data usage: •00:07 - 0s (no byte_enbl) •08:09 - reserved (no byte_enbl) •10:15 - CT = 1 (no byte enabl) •16:31 - FRC=dest(no byte_enbl) •32:39 - 0s (no byte_enbl) •40:55 - 0s (no byte_enbl)
hwsync	101011														51	
icbi (nop on AtNodeL2)	111110	22:57													45	
icblc2	100100	22:57			=0										64	
icbt (L2 only)1	000100	22:57			=0										8	
icbtls (L2 only)1	010100	22:57			=0										32	
ici	101110														53	
icswx	100110	22:57											0:3	0:55	61	st_data usage: •00:31 - data w/ byte_enbl(0:3) •32:39 - LPID (no byte_enbl) •40:55 - '00',PID (no byte_enbl)
icswx.	100111	22:57											0:3	0:55	62	st_data usage: •00:31 - data w/ byte_enbl(0:3) •32:39 - LPID (no byte_enbl) •40:55 - '00',PID (no byte_enbl)
inst fetch	000000	22:57													1	
l1_load_hit	110100	22:63			=0										40	Presentation of ttype is under A2 Core XUCR Mode control, depending on attached L2. Either appears as a normal store-ttype (with pwr_token, req_vld, requiring req_st_pop), or it does not appear at all on the interface.
llwarx/ldarx w/ mutex hint	001011	22:63													11	
load	001000	22-63										l=1			9	
lwarx/ldarx	001001	22:63													10	
lwsync	101010														52	
mbar	110010														36	
mmu_read	000010											l=1			4	l=0 or 1 controlled by s/w, never mixing l=0 w/ l=1. mmu_read w/ l=1 should request xfr_len = 16 bytes. Thread is a don't care.
msgsnd	101101	32:63													55	

Table 1-5. Interface Signal Usage based on TTYPE

ttype	req_ttype (0:5) (abcdef)	req_ra	req_thread	req_wimg_w	req_wimg_i	req_wimg_m	req_wimg_g	req_endian	req_user_defined	req_spare_ctrl_a0	req_ld_core_tag	req_ld_xfr_len	st_byte_enbl	st_data	l-r sort	Notes
mtspr_trace	101100	32:63													56	req_ra usage: • 30:31 - Core ID • 32:33 - Thread ID • 34:43 - Marker type • 44 - 0 • 45 - Marker valid • 46 - Start trigger • 47 - Stop trigger • 48 - Pause trigger • 49:63 - reserved (unused)
store	100000	22:63										l=1			57	A2 core sends xfr length for l=1 stores (byproduct of similar handling of l=1 loads)
stwcx./stdcx.	101001	22”63													50	
tlbi_complete	111011														43	
tlbivax	111100	tlb 22:63												32:47	48	st_data(32:47): see Table 3-1 on page 52 for st_data usage details.
tlbsync	111010														44	

1.6 Reload Interface Description

Table 1-6 L2-Core Reload Interface Description

Table 1-6. L2-Core Reload Interface Description

Src	Signal Suffix	bits	cyc	Signal Description
Reload Interface Signals		143		
L2	reld_data_coming	1	d-3	**Reload Data Coming**. Indicates that reload data is coming in three cycles. This signal is required for L2 designs that return data in consecutive cycles, but can be tied to a logic '0' for designs that return data in every other cycle. For L2 designs that return data in consecutive cycles, this signal should be asserted three cycles ahead of the first of two paired data beats. If more than two data beats will be presented consecutively, this signal should be asserted once for the first set of two (paired) data beats and once for the second set of two (paired) data beats. Each assertion should be three cycles ahead of the first beat of the paired set of data beats. This signal allows the A2 core to insert an issue bubble for the second reload data beat to avoid flushing the processor pipe. This signal is not required to be asserted as described above for DITC return data. For non-cacheable (I=1) reloads of one or two beats, this signal should be asserted three cycles ahead of the first (and possibly only) data beat transfer.
L2	reld_data_vld	1	d-2	**Reload Data Valid**. Indicates that reload data is coming in two cycles. This signal qualifies the other reload interface signals sent in this cycle: reld_ditc, reld_core_tag, reld_crit_qw, and reld_qw. If reld_data_vld is not active, then the qualified signals should be ignored.
L2	reld_ditc	1	d-2	**Reload Direct Inter-Thread Communication**. Indicates that the reload data is associated with a DITC transfer instead of a standard load-ttype reload. This signal is qualified by reld_data_vld and determines the interpretation of the reld_core_tag bus. DITC reload data transfers are always 64-Byte transfers that follow the same consecutive cycle or every-other-cycle behavior as standard load-ttype reloads for the attached L2.
L2	reld_core_tag	0:4	d-2	**Reload Core Tag**. Identifies the target of the reload data transfer. The AT L2 design only makes use of req_ld_core_tag(1:4). This bus is qualified by reld_data_vld and it is interpreted further by whether or not the reld_ditc bit is asserted, as follows: • If the reld_ditc bit is on in the presence of reld_data_vld, then reld_core_tag(3:4) is an encoded field that identifies the target thread for the 64-Byte DITC data transfer, and reld_core_tag(0:2) are unused. • If the reld_ditc bit is off in the presence of reld_data_vld, then reld_core_tag(0:4) identifies the destination unit & resource within the unit (for example, which Miss Queue) of a standard reload data transfer. In this case, the reld_core_tag(0:4) field will match the value that was presented on req_ld_core_tag(0:4) when the reload request was presented. See req_ld_core_tag definition for the specific bit encoding.
Key d: cycle of reload data transfer				

Table 1-6. L2-Core Reload Interface Description

Src	Signal Suffix	bits	cyc	Signal Description
A2	reld_ditc_pop	0:3	-	**Direct Inter-Thread Communication inbox Pop**. Indicates (one bit per thread) that the A2 Core has freed up a slot in the DITC inbox for the thread or threads indicated. The A2 Core has four inbox entries for each thread. The L2 keeps track of how many DITC transfers it sends to each thread with a per-thread inbox entry counter, which it decrements with each DITC sent and increments with each “pop” it receives from the A2 Core. When the L2 detects that the inbox for a thread is full, it cannot send another DITC to that thread until a “pop” signal is received for that thread.
L2	reld_crit_qw	1	d-2	**Reload Critical Quadword**. Indicates that the reload data quadword that is going to be presented in two cycles is the critical quadword of this reload transfer. This signal is expected to be asserted whether the reload is for an l=0 (four data beat) or l=1 (one or two data beat) reload transfer. However, this signal is not used for DITC transfers.
L2	reld_qw	57:59	d-2	**Reload data Quadword identifier**. Indicates which quadword (16B) is going to be presented on the reload data bus in two cycles. This signal isrequired for both data and ditc reloads. Bit 57 is only used when the core is configured for a 128 byte L1 D-cache line size. When the L1 D-cache line size is 64 bytes, this pin can be tied to zero.
L2	reld_l1_dump	1	d-2	**Reload L1 Dump.** Indicates that the reload data for this cache line should not be cached in the L1 and only forwarded to the register of the miss causing thread. (This signal could be tied to 0 if it is not needed by a particular L2 implementation). This signal should be activated for each and every transfer of the reload that is to be dumped.
L2	reld_data	0:127	d	**Reload Data**. One quadword (16B) of Reload data (one data beat). The quadword is identified by “rld_qw” two cycles earlier. Cache line reloads require four data beats for a total of 64 Bytes. Non-cacheable reloads of 16 bytes or less will be contained within one quadword and will not cross a quadword boundary. Non-cacheable reloads of 32 bytes will return data in two data beats and will not cross an octword boundary. DITC reload data transfers are always 64-Byte transfers.
L2	reld_ecc_err	1	d+1	**Reload data ECC Error**. This signal is valid one cycle after the corresponding reload data beat to which it pertains. This signal indicates that the reload data beat sent in the previous cycle has an ECC error. Data from a data beat with this ECC error indication cannot be used and the line cannot be designated as “valid” in the L1 cache. If the data beat that contains the target data for the load that generated the cache line read arrives without an ECC error, the data for the clean data beat may be forwarded for use, but if the other data beat contains an ECC error the line cannot be designated as “valid” in the L1 cache. The L2 will resend the entire line of data at some future time.
L2	reld_ecc_err_ue	1	d+1	**Reload data ECC Uncorrectable Error**. This signal is valid one cycle after the corresponding reload data beat. It indicates that the reload data beat sent in the previous cycle has an Uncorrectable ECC Error (UE). The L2 will set a maskable checkstop bit. The core should NOT put this line into the cache, but it should forward the data and use it so the Miss Queue or the thread will not hang.
Key d: cycle of reload data transfer				

1.6.1 L2 Reload Data Return Order

Each reload data beat returns a quadword (QW) of data. The L2 indicates the quadword address of the reload data beat two cycles before it presents the reload data using reld_qw (and reld_crit_qw), qualified by reld_data_vld.

- Cacheable reloads of 64 bytes are transferred in four reload data beats.
- DITC transfers are 64-byte transfers and are transferred in four reload data beats.
- Non-cacheable reloads of 16 bytes or less are transferred in one reload data beat.
- Non-cacheable reloads of 32 bytes are transferred in two reload data beats.

The A2 core can handle any order of reload data return, provided that when more than one quadword is transferred, each paired set of 16-Byte quadwords received in order must be from the same 32-Byte octword. Providing the critical quadword first is not mandatory, but it is desirable, as it reduces load latency and enables the A2 core pipeline to make forward progress.

For a cacheable reload transfer, an L2 should attempt to provide the critical quadword data in the first data beat, followed by the remaining data beats, as identified by the encoded reld_qw value. If the memory array that contains the critical quadword is unable to be accessed before other quadwords of the cacheline, then the reload transfer may be returned without the critical quadword first. In such a situation, the L2 should return the critical quadword as early as the memory access and transfer protocols allow. The ordering possibilities are reflected in Table 1-7.

For a DITC transfer, the reld_crit_qw signal has no meaning, as there is no critical quadword.

For a non-cacheable reload transfer of 16 bytes or less, only a single reload data beat is returned. Since it is the only quadword returned, it is treated as the critical quadword.

For a non-cacheable reload transfer of 32 bytes, two reload data beats are returned. In this case, the L2 should attempt to provide the critical quadword in the first data beat, followed by the second data beat. The critical quadword is identified by the target address of the load generating the reload.

Table 1-7. Permissable 64-Byte Reload Data Delivery (by QW)

1st beat	2nd beat	3rd beat	4th beat
0	1	2	3
		3	2
1	0	3	2
		2	3
2	3	0	1
		1	0
3	2	1	0
		0	1

Table 1-8. 64-Byte Reload Data Delivery Used by At Node L2 (by QW)

1st beat	2nd beat	3rd beat	4th beat
0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

1.6.2 L2 Reload Data Return Sequencing

The A2 core supports two modes of reload data transfer from an L2. The mode is set via a configuration register based upon the reload data transfer method supported by the attached L2. The two modes are:

1. Reload data returned every other cycle
2. Reload data returned in back-to-back, or consecutive, cycles

Interlacing of reload data is allowed, with certain restrictions. Interlacing will be addressed in each of the next two sections.

Additional rules that govern the sequencing of a cacheable reload data transfer for each mode follow in the next sections.

1.6.2.1 Reload Data Return Rules for an L2 that Returns Data Every Other Cycle

1. The A2 core does not require the use of reld_data_coming for an L2 that returns data in every other cycle; the signal may be tied to '0'.
2. The A2 core allows the L2 to send data every other cycle until all four quadwords of the cacheline are delivered, requiring four cycles of reld_data, with a gap (empty cycle) in between each piece of data of the same transfer.

   Figure 1-1 shows a standard cacheable reload transfer of four data transfer cycles sent every other cycle. Notice that reld_data_coming is tied to '0', as it is not required for an L2 that returns data every other cycle. In this transfer, the critical quadword, "rld-a0", (reld_crit_qw='1', reld_qw='00') is sent first.

Figure 1-1. Cacheable Reload Data Return by an Every-Other-Cycle L2 (fastest)

Signal	1	2	3	4	5	6	7	8	9	10	11	12
reld_data_coming	tied '0'											
reld_data_vld		1		1		1		1				
reld_ditc		0		0		0		0				
reld_core_tag(0:4)		rld-a		rld-a		rld-a		rld-a				
reld_crit_qw		1		0		0		0				
reld_qw(0:1)		00		01		10		11				
reld_data(0:127)				rld-a0		rld-a1		rld-a2		rld-a3		
reld_ecc_err					$0_{(a0)}$		$0_{(a1)}$		$0_{(a2)}$		$0_{(a3)}$	
reld_ecc_err_ue					$0_{(a0)}$		$0_{(a1)}$		$0_{(a2)}$		$0_{(a3)}$	

3. The A2 core allows the L2 to send data in two sets of two quadwords each, where the two sets may be separated by a variable number of cycles whose lower limit is a single cycle of gap, which is required for an L2 that returns data in every other cycle. The number of cycles between the two sets can vary widely, depending upon the L2's array access methodology and other factors. Within a set of paired quadwords, only a single cycle gap is allowed.

   Figure 1-2 shows a cacheable reload transfer of four data transfer cycles sent every other cycle, but this time it is sent in two sets of paired quadwords with a gap of three cycles between sets. In this transfer, the critical quadword, "rld-a0", (reld_crit_qw='1', reld_qw='00') is sent first, followed by a gap cycle and then

its paired quadword, “rld-a1”. After that, there is a three-cycle separation between the first and second set of paired quadwords, followed by “rld-a2” and its paired quadword, “rld-a3”.

Figure 1-2. Cacheable Reload Data Return by an Every-Other-Cycle L2 (w/ 3 cycle gap)

Signal	1	2	3	4	5	6	7	8	9	10	11	12
reld_data_coming	tied ‘0’											
reld_data_vld	1		1				1		1			
reld_ditc	0		0				0		0			
reld_core_tag(0:4)	rld-a		rld-a				rld-a		rld-a			
reld_crit_qw	1		0				0		0			
reld_qw(0:1)	00		01				10		11			
reld_data(0:127)			rld-a0		rld-a1	◄ 3 cycles ►			rld-a2		rld-a3	
reld_ecc_err				$0_{(a0)}$		$0_{(a1)}$				$0_{(a2)}$		$0_{(a3)}$
reld_ecc_err_ue				$0_{(a0)}$		$0_{(a1)}$				$0_{(a2)}$		$0_{(a3)}$

4. The A2 core allows the L2 to interlace data for two different reload transfers, as long as the rules above remain intact for each of the two reload data transfers, as identified by the unique reld_core_tag’s of the two reload transfers.

   For every-other-cycle L2s, interlacing is allowed to occur in the gaps that occur between the first and second quadwords of a paired set. Interlacing always requires that the paired quadword of a set is present either before it or after it, in an every-other cycle fashion. Further, if there is a gap of a large number of cycles between a first and second set of a reload transfer, multiple reloads for other operations (interlaced or not) are allowed to be transferred in the gap, as long as each set’s paired quadwords appear in an every-other-cycle fashion.

   Figure 1-3 shows two cacheable reload transfers, each of which is returned in two sets of paired quadwords. The first reload, “rld-a” shown in red below, is exactly the same reload as that shown in *Figure 1-2* on page 18. The second reload, “rld-b” shown in blue below, is shown to demonstrate interlacing of reload transfers. “rld-b” is shown returning data in every other cycle without interruption (except for interlacing) entirely within the bounds of “rld-a”. This is not necessary, but done this way for simplicity of the example. “rld-a” is a return of reload data in which the critical quadword, “rld-a0” (reld_qw=’00’), is returned first, as seen by the controls of cycle 1 and the associated data transfer two cycles later, in cycle 3. The controls for its paired quadword “rld-a1”, are shown in cycle 3, along with its data two cycles later in cycle 5. Interlaced with this reload is the first set of a second reload, “rld-b” shown in blue below. (From here on in this discussion, I will refer to the control cycles when referring to a transfer, knowing that the associated data transfer for each occurs two cycles later.) The critical quadword for the second transfer is “rld-b3” (reld_qw=’11’), which is returned first, in cycle 2, followed in cycle 4 by its paired quadword “rld-b2”. At this point, the first reload was unable to provide return controls in cycle 5, as it may have had to wait for array access, but the second reload was able to continue uninterrupted and actually complete before the first reload. The signals for the second set of the second reload are then returned in cycle 6, as seen by the controls to return “rld-b0”, along with the controls for its paired quadword “rld-b1” in cycle 8. INterlaced with this, the signals for the second set of the first reload are returned in cycle 7, as seen by the controls to return “rld-a2”, along with the controls for its paired quadword “rld-a3” in cycle 9. The data

transfers for each quadword are sent two cycles after the controls, and the corresponding ECC error indications are sent one cycle after each data quadword.

Figure 1-3. Cacheable Reload Data Return Interlacing by an Every-Other-Cycle L2

Signal	1	2	3	4	5	6	7	8	9	10	11	12
reld_data_coming	tied '0'											
reld_data_vld	1	1	1	1		1	1	1	1			
reld_ditc	0	0	0	0		0	0	0	0			
reld_core_tag(0:4)	rld-a	rld-b	rld-a	rld-b		rld-b	rld-a	rld-b	rld-a			
reld_crit_qw	1	1	0	0		0	0	0	0			
reld_qw(0:1)	00	11	01	10		00	10	01	11			
reld_data(0:127)			rld-a0	rld-b3	rld-a1	rld-b2		rld-b0	rld-a2	rld-b1	rld-a3	
reld_ecc_err				$0_{(a0)}$	$0_{(b3)}$	$0_{(a1)}$	$0_{(b2)}$		$0_{(b0)}$	$0_{(a2)}$	$0_{(b1)}$	$0_{(a3)}$
reld_ecc_err_ue				$0_{(a0)}$	$0_{(b3)}$	$0_{(a1)}$	$0_{(b2)}$		$0_{(b0)}$	$0_{(a2)}$	$0_{(b1)}$	$0_{(a3)}$

1.6.2.2 Reload Data Return Rules for an L2 that Returns Data in Back-to-Back, or Consecutive, Cycles

1. The A2 core requires the use of reld_data_coming for an L2 that returns data in back-to-back cycles; the signal must precede (by three cycles) the presentation of the first data beat of each set of paired quadwords. If the two sets of paired quadwords of a cacheline reload transfer are presented without a gap between the two sets, then the reld_data _coming signal is asserted in cycle (d-3) for the first set of reld_data that is presented in cycle (d) and (d+1), and again is asserted in cycle (d-1) for reld_data that is presented in cycle (d+2) and (d+3). This scenario is presented below, in text and in Figure 1-4.
2. The A2 core allows the L2 to send data every cycle until all four quadwords of the cacheline are delivered, requiring four cycles of reld_data, with no gaps in between the four data beats.

   Figure 1-4 shows a standard cacheable reload transfer sent in four consecutive cycles. Notice that reld_data_coming is asserted twice, once for the first set of paired quadwords and once for the second set. In this transfer, the critical quadword, “reld-a0” (reld_crit_qw=’1’, reld_qw=’00’) is sent first.

Figure 1-4. Cacheable Reload Data Return by a Back-to-Back L2 in Four Consecutive Cycles

Signal	1	2	3	4	5	6	7	8	9	10	11	12
reld_data_coming	$1_{(rld\text{-}a01)}$		$1_{(rld\text{-}a23)}$									
reld_data_vld		1	1	1	1							
reld_ditc		0	0	0	0							
reld_core_tag(0:4)		rld-a	rld-a	rld-a	rld-a							
reld_crit_qw		1	0	0	0							
reld_qw(0:1)		00	01	10	11							
reld_data(0:127)				rld-a0	rld-a1	rld-a2	rld-a3					
reld_ecc_err					$0_{(a0)}$	$0_{(a1)}$	$0_{(a2)}$	$0_{(a3)}$				
reld_ecc_err_ue					$0_{(a0)}$	$0_{(a1)}$	$0_{(a2)}$	$0_{(a3)}$				

3. The A2 core allows the L2 to send data in two sets of two quadwords each, where the two sets may be separated by a variable number of cycles whose lower limit is zero, which means the L2 returns data in four consecutive cycles. The number of cycles between the two sets can vary widely, depending upon the L2’s array access methodology and other factors.

   Figure 1-5 shows a cacheable reload transfer sent in two sets of paired quadwords with a four-cycle separation between the two sets of paired quadwords. Again, reld_data_coming is asserted twice, once for the first set of paired quadwords and once for the second set. In this transfer, the critical quadword, “rld_a0” (reld_crit_qw=’1’, reld_qw=’00’) is sent first.

Figure 1-5. Cacheable Reload Data Return by a Back-to-Back L2 in Two Sets of Paired Quadwords

Signal	1	2	3	4	5	6	7	8	9	10	11	12
reld_data_coming	$1_{(rld\text{-}a01)}$						$1_{(rld\text{-}a23)}$					
reld_data_vld		1	1					1	1			
reld_ditc		0	0					0	0			
reld_core_tag(0:4)		rld-a	rld-a					rld-a	rld-a			
reld_crit_qw		1	0					0	0			
reld_qw(0:1)		00	01					10	11			
reld_data(0:127)				rld-a0	rld-a1	4 cycle separation				rld-a2	rld-a3	
reld_ecc_err					$0_{(a0)}$	$0_{(a1)}$					$0_{(a2)}$	$0_{(a3)}$
reld_ecc_err_ue					$0_{(a0)}$	$0_{(a1)}$					$0_{(a2)}$	$0_{(a3)}$

4. The A2 core allows interlacing of reload data for two distinct reload transfers between sets of paired quadwords. You are not allowed to break in between paired quadwords of a set. It is possible, however, to have the first set of paired quadwords for reld_core_tag=Y, followed by a set (or more) of paired quadwords for a different reld_core_tag, followed eventually by the second set of paired quadwords for reld_core_tag=Y.

   For back-to-back L2s, interlacing is allowed to occur on a “set” basis, in the gaps that occur between the first and second sets of a reload transfer (cacheable or DITC). Interlacing is not allowed between the paired quadwords of a set. The paired quadwords of any set must always be returned back-to-back from such an L2. If there is a gap of a large number of cycles between a first and second set of a reload transfer, multiple reloads for other operations (interlaced or not) are allowed to be transferred in the gap, as long as each set’s paired quadwords appear in a back-to-back fashion.

   Figure 1-6 shows two cacheable reload transfers, each of which is returned in two sets of paired quadwords. The first reload, “rld-a” shown in red below, is a return of reload data in which the critical quadword is “rld-a0” (reld_qw=’00’), which is returned first, followed immediately by its paired quadword “rld-a1”. Then the first set of the second reload, “rld-b” shown in blue below, is returned. Notice that the critical quadword for this transfer is “rld-b3” (reld_qw=’11’), which is returned first, followed immediately by its paired quadword “rld-b2”. The second set of the first reload is then returned, as seen by the controls and data to return “rld-a2” and its paired quadword “rld-a3”. Lastly, the second set of the second reload is then returned, as seen by the controls and data to return “rld-b0” and its paired quadword “rld-b1”.

Figure 1-6. Cacheable Reload Data Return Interlacing by a Back-to-Back L2

Signal	1	2	3	4	5	6	7	8	9	10	11	12
reld_data_coming	$1_{(rld\text{-}a01)}$		$1_{(rld\text{-}b23)}$		$1_{(rld\text{-}a23)}$		$1_{(rld\text{-}b01)}$					
reld_data_vld		1	1	1	1	1	1	1	1			
reld_ditc		0	0	0	0	0	0	0	0			
reld_core_tag(0:4)		rld-a	rld-a	rld-b	rld-b	rld-a	rld-a	rld-b	rld-b			
reld_crit_qw		1	0	1	0	0	0	0	0			
reld_qw(0:1)		00	01	11	10	10	11	00	01			
reld_data(0:127)				rld-a0	rld-a1	rld-b3	rld-b2	rld-a2	rld-a3	rld-b0	rld-b1	
reld_ecc_err					$0_{(a0)}$	$0_{(a1)}$	$0_{(b3)}$	$0_{(b2)}$	$0_{(a2)}$	$0_{(a3)}$	$0_{(b0)}$	$0_{(b1)}$
reld_ecc_err_ue					$0_{(a0)}$	$0_{(a1)}$	$0_{(b3)}$	$0_{(b2)}$	$0_{(a2)}$	$0_{(a3)}$	$0_{(b0)}$	$0_{(b1)}$

1.7 Back-Invalidate / Snooped TLBI / CPX Status Interface Description

Table 1-9 L2-Core Back-Invalidate / Snooped TLBI / CPX Status Interface Description

Table 1-9. L2-Core Back-Invalidate / Snooped TLBI / CPX Status Interface Description

Src	Signal Suffix	bits	cyc	Signal Description
Back-Invalidate Interface Signals		69		
L2	back_inv	1	b-1	**Back Invalidate Valid**. Indicates that a back invalidate, a tlbi snoop, or CPX status is being presented to the target identified by back_inv_target this cycle. The back_inv_addr bits will carry the information pertinent to the operation in the next cycle, along with the back_inv_lbit and back_inv_lpar_id(0:9) for tlbi snoops operations.
L2	back_inv_target	0:4	b-1	**Back Invalidate Target**. Indicates target of back invalidate. • bit 0: I-side (used for back-invalidate) • bit 1: D-side (used for back-invalidate) • bit 2: MMU (used for tlbi snoop) • bit 3: CPX (used for coprocessor indirect status) • bit 4: IPI (Inter-Processor Interrupt - used for message passing)
Key b: cycle of back invalidate address/tlbi snoop/cpx status presentation				

Table 1-9. L2-Core Back-Invalidate / Snooped TLBI / CPX Status Interface Description

Src	Signal Suffix	bits	cyc	Signal Description

L2	back_inv_addr	42	b	**Back Invalidate Address**.
				For I-side or D-side: 36-bit Back Invalidate address:
				• 22:57 - indicates the 64-Byte L1 cache line to invalidate
				For MMU (tlbi snoop), the 42-bit address definition is as follows:
				• 22:26 - TID(3:7) (translation identifier; i.e. targeted process ID)
				• 27:51 - EPN (i.e. RB(27:51) from the tlbivax instruction)
				• 52 - TS (translation space)
				• 53 - TID(2) (translation identifier; i.e. targeted process ID)
				• 54:55 - attributes
				• 56:63 - TID(8:15) (translation identifier; i.e. targeted process ID)
				For CPX: Carries icswx. status of CPX operation as follows:]
				• 58:60 - 3-bit condition code response
				• 62:63 - 2-bit encoded thread ID
				For IPI (Inter-Processor Interrupt): Carries message of IPI operation as follows:
				• 32:36 - message/interrupt type for msgsnd
				  • 00000: Type 0 - Doorbell Interrupt
				  • 00001: Type 1 - Doorbell Critical Interrupt
				  • 00010: Type 2 - Guest Processor Doorbell Interrupt
				  • 00011: Type 3 - Guest Processor Doorbell Critical Interrupt
				  • 00100: Type 4 - Guest Processor Doorbell Machine Check Interrupt.
				  • 00010-11111: undefined
				• 37:63 - message packet for msgsnd
				  • 37 - Broadcast flag
				  • =0: compare message PIRTAG to the PIR and compare message LPID tag to LPID to determine if interrupt is generated
				  • =1: compare message LPID tag to LPID to determine if the interrupt is generated regardless of the value of the PIRTAG and PIR
				  • 38:41 - (reserved)
				  • 42:49 - LPID tag to be compared to the LPID register. If the LPID tag = 0 it matches all values in the LPID register.
				  • 50:63 - PIRTAG to be compared to the PIR

L2	back_inv_ind	1	b	**Indirect Entry**. Indicates Indirect Entry attribute for tlbi snoop.
L2	back_inv_gs	1	b	**Guest Space**. Indicates Guest Space attribute for tlbi snoop.
L2	back_inv_lbit	1	b	**Large Page**. Indicates Large Page attribute for tlbi snoop.
L2	back_inv_lpar_id	0:7	b	**LPAR ID**. Indicates LPAR ID for a tlbi snoop. The L2 passes all tlbi snoops to the core. The L2 does not filter tlbi snoops based on the lpar_id.
L2	back_inv_local	1	b	**Local tlbi snoop**. Indicates that the tlbi snoop being sent on the back-inv interface is the result of a locally-generated tlbivax or erativax instruction from the attached A2 Core. The attached core uses this signal to decide if it should clear a thread issue stall that may be pending due to these instructions.
core	back_inv_reject	1	b+2	**Back Invalidate Reject**. Indicates rejection of LPAR ID for tlbi snoop. The core/MMU (when operating in ERAT-only mode, or when MMUCR1[TLBI_REJ]=1) will send the reject if the tlbi snoop isn't meant for this core's LPAR_ID. If the core is operating in MMU mode, or if the tlbi snoop is meant for this core's LPAR_ID, the core will send a subsequent tlbi_complete (a store ttype) to the L2.

Key
b: cycle of back invalidate address/tlbi snoop/cpx status presentation

1.8 Reservation Interface Description

Table 1-10 L2-Core Reservation Interface

Table 1-10. L2-Core Reservation Interface

Src	Signal Suffix	bits	cyc	Signal Description(x)
Reservation Interface Signals		8		
L2	stcx_complete	0:3	s	**STWCX./STDCX. Complete**. Indicates that status is being presented for a stwcx./stdcx. operation for one or more threads, on a per thread basis. Qualifies the corresponding bit of the stcx_pass signals.
L2	stcx_pass	0:3	s	**STWCX./STDCX. Pass**. Indicates stwcx./stdcx. status for one or more threads, on a per thread basis, when qualified with the assertion of the corresponding stcx_complete bit.
L2	reservation_vld	0:3	-	**Reservation Valid**. Indicates the status of the reservation valid bit on a per-thread basis. This bit is used in a fast wake-up mechanism from the *wait* state. A thread may enter the *wait* state after using a lwarx/ldarx to set a reservation, which causes this bit to be asserted. In an effort to gain the thread's attention for service, another thread or off-node source performs a store to the reservation address. This causes the reservation to clear, which produces a deassertion of this signal that indicates that the thread should leave the *wait* state and resume processing.
Key s: cycle of stwcx./stdcx. status presentation				

1.9 SYNC Interface Description

Table 1-11 L2-Core Sync Interface

Table 1-11. L2-Core Sync Interface

Src	Signal Suffix	bits	cyc	Signal Description
Sync Interface Signals		4		
L2	sync_ack	0:3	sa	**SYNC Acknowledge**. Indicates the acknowledgment of a synchronizing command that requires one on a per thread basis. Indicates that the thread may begin issuing commands again. In the case of a back-invalidate followed by a sync_ack, the L2 will delay the sync_ack for 3 cycles in order to give the core suf'ficient cycles to complete the back invalidation. Used for: • HWSYNC • TLBSYNC (when tlbsync_ack mode = '1') Not used for: • LWSYNC • MBAR • TLBSYNC (when tlbsync_ack mode = '0')
Key sa: cycle of sync_ack presentation				

1.10 ICBI Interface Description

When the icbi ack enable bit is set in the IUCR register, the icbi instructions need to be acknowledged by the L2 before the barrier will be released.

Table 1-12 L2-Core ICBI Interface

Table 1-12. L2-Core ICBI Interface

Src	Signal Suffix	bits	cyc	Signal Description
ICBI Interface Signals		3		
L2	icbi_ack	1	i	**ICBI Acknowledge**. Indicates the acknowledgment of an icbi instruction back to the core, for the thread encoded in icbi_ack_thread. Icbi is a barrier op. The core relies on the L2 to release the barrier op for that thread by indicating icbi_ack.
L2	icbi_ack_thread	0:1	i	**ICBI Thread**. Encoded thread ID for the icbi command which is being acknowledged.
Key i: cycle of icbi_ack presentation				

1.11 Interrupt Presentation Interface Description

Table 1-13 L2-Core Interrupt Presentation Interface

Table 1-13. L2-Core Interrupt Presentation Interface

Src	Signal Suffix	bits	cyc	Signal Description
Interrupt Presentation Interface Signals		4		
L2	ext_interrupt	0:3	-	**External Interrupt Request.** Indicates (one per thread) that an external interrupt request is waiting to be serviced by the indicated thread.
L2	crit_interrupt	0:3	-	**Critical Interrupt Request.** Indicates (one per thread) that a criticall interrupt request is waiting to be serviced by the indicated thread.
L2	perf_interrupt	0:3	-	**Performance Monitor Interrupt Request.** Indicates (one per thread) that a performance monitorinterrupt request is waiting to be serviced by the indicated thread.

1.12 Power Management Interface Description

Table 1-14 Core-L2 Power Management Interface

Table 1-14. Core-L2 Power Management Interface

Src	Signal Suffix	bits	cyc	Signal Description(x)
Power Management Interface Signals		2		
core	power_managed	1		**Power Managed**. Indicates that all of the A2 core's threads is in one of the power management states: PM_Sleep or PM_RVW. This signal is brought out of the core as an additional indicator of power management status.
core	rvwinkle_mode	1		**Rip Van Winkle Mode**.Indicates that each of the A2 core's threads is in the PM_RVW power management state. The assertion of this signal from the A2 core indicates that the core will not send any further requests to the L2 cache until a wakeup event occurs.
L2	sleep_en	0:3		**Sleep Enable**. Indicates that execution can continue following a wait instruction with WC=2. There is 1 bit per thread. When this signal is 0, the appropriate thread will resume the instruction execution that was halted due to the wait instruction. For applications that do not implement this function, the pins can be tied to 0.
Key				

1.13 Pervasive/Misc Interface Description

Table 1-15. Core-L2 MIsc Interface Signals

Src	Signal Suffix	bits	cyc	Signal Description
Pervasive Interface Signals				
L2	FLH2L2_GATE	1	-	This signal is tied to a register that is set by MMIO. In the core it will be gated onto the XUCR0 bit that controls the forwarding of L1 load hits to the L2.
Key “cyc” identifier: description				

2. Pervasive Interface Signals

Table 2-1. A2 Core Pervasive Interface Signal List			
Signal Name	**Size**	**In/Out**	**Description**
Reset Signals:			
an_ac_reset_1_complete	1	IN	Reset type status to DBSR[MRR]
an_ac_reset_2_complete	1	IN	Reset type status to DBSR[MRR]
an_ac_reset_3_complete	1	IN	Reset type status to DBSR[MRR]
ac_an_reset_1_request	1	OUT	reset_1 type request from DBCR0 or WDT
ac_an_reset_2_request	1	OUT	reset_2 type request from DBCR0 or WDT
ac_an_reset_3_request	1	OUT	reset_3 type request from DBCR0 or WDT
Errors and Interrupts:			
ac_an_special_attn	4	OUT	Thread specific special attention signals to chiplet FIR
ac_an_recov_err	3	OUT	Recoverable error signals to chiplet FIR
ac_an_checkstop	3	OUT	Checkstop error signals to chiplet FIR
ac_an_local_checkstop	3	OUT	Local Checkstop error signals to chiplet FIR (not implemented).
an_ac_checkstop	1	IN	To FIR; fences core on checkstop
an_ac_external_mchk	4	IN	External machine check interrupt requests.
an_ac_malf_alert	1	IN	To FIR; indicates malfunction alert in system (not used).
SCOM Interface:			
an_ac_scom_cch	1	IN	SCOM control channel input
an_ac_scom_dch	1	IN	SCOM data channel input
an_ac_scom_sat_id	4	IN	SCOM satellite address configuration
ac_an_scom_cch	1	OUT	SCOM control channel output
ac_an_scom_dch	1	OUT	SCOM data channel output
ID Interface:			
an_ac_coreid_dc	8	IN	Used to set core ID information in PIR
Hang Pulses:			
an_ac_hang_pulse	4	IN	Hang pulse to core livelock buster
Time Base:			
an_ac_tb_update_enable	1	IN	Enable incrementing of timer facilities; assists with timebase synchronization
an_ac_tb_update_pulse	1	IN	Externally generated update pulse for incrementing timer facilities.

Table 2-1. A2 Core Pervasive Interface Signal List			
Signal Name	**Size**	**In/Out**	**Description**
Power Management:			
an_ac_pm_thread_stop	4	IN	Stop control for threads 0 to 3
ac_an_pm_thread_running	4	OUT	Stop/running state from threads 0 to 3
ac_an_power_managed	1	OUT	All core threads are in a power-savings mode
ac_an_rvwinkle_mode	1	OUT	All core threads are in rvwinkle mode
PSRO Sensor:			
an_ac_psro_enable_dc	3	IN	Enable control to PSRO sensor
ac_an_psro_ringsig	1	OUT	PSRO sensor output to chiplet level monitor.
Debug and Trace:			
ac_an_debug_bus	88	OUT	MUXed debug bus from core units to external trace logic
ac_an_trace_trigger	16	OUT	MUXed trigger signals from core units to external trace logic
ac_an_debug_trigger	4	OUT	Debug event trigger output; used to stop all cores on chiplet.
an_ac_debug_stop	1	IN	External debug control signal used to stop all threads. Must be enabled via PCCR0[DBG_STOP_EN]
ac_an_event_bus	8	OUT	MUXed event bus from core units to external PMU
ac_an_fu_bypass_events	4	OUT	FU event mux outputs that bypass the PC unit core event mux
ac_an_iu_bypass_events	8	OUT	IU event mux outputs that bypass the PC unit core event mux
ac_an_mm_bypass_events	8	OUT	MMU event mux outputs that bypass the PC unit core event mux
Test Signals:			
an_ac_scan_diag_dc	1	IN	Used for scan ring debug in test. Connected to XOR in scan ring chains. The XORs should be taken care of by Pervasive components (LCBCNTRLs, arrays, graybox). This signal should be connected to these components as needed.
an_ac_abist_mode_dc	1	IN	Set before and active during an ABIST test. Used by ABIST engine to generate AC version used by arrays and muxed logic.
an_ac_abist_start_test	1	IN	Starts the ABIST engine
ac_an_abist_done	1	OUT	ABIST done signal from core ABIST engines
an_ac_lbist_en_dc	1	IN	Set before and active during an LBIST test. Used to fence logic that would lead to unstable LBIST test signatures (e.g. asynchronous circuit outputs)
an_ac_lbist_ac_mode_dc	1	IN	Set before and active during an AC LBIST test. Used to fence logic that would lead to unstable LBIST test signatures (e.g. DC paths)
an_ac_lbist_ary_wrt_thru_dc	1	IN	Forces act_dis_dc when in ABIST write though mode
an_ac_lbist_ip_dc	1	IN	indicates LBIST testing is in process
an_ac_gsd_test_enable_dc	1	IN	Enable control for LSSD/GSD testing

Table 2-1. A2 Core Pervasive Interface Signal List			
Signal Name	**Size**	**In/Out**	**Description**
an_ac_gsd_test_acmode_dc	1	IN	Sets LSSD/GSD testing to AC mode
an_ac_atpg_en_dc	1	IN	ABIST control for arrays with multiple write ports
Clock Controls:			
an_ac_rtim_sl_thold_7	1	IN	thold control input for rtime ring latches. To PC clock controls
an_ac_func_sl_thold_7	1	IN	thold control input for func ring latches. To PC clock controls
an_ac_func_nsl_thold_7	1	IN	thold control input for func non-scan latches. To PC clock controls
an_ac_ary_nsl_thold_7	1	IN	thold control input for arrays. To PC clock controls
an_ac_sg_7	1	IN	SG control for scannable latches. To PC clock controls
an_ac_fce_7	1	IN	FCE control for non-scan latches. To PC clock controls
an_ac_scan_type_dc	9	IN	Selects which scan chain that will be scanned
an_ac_ccflush_dc	1	IN	Flush signal to staging latches
an_ac_ccenable_dc	1	IN	Clock control enable
Graybox Interface:			
an_ac_scan_dis_dc_b	1	IN	Fences scan stump outputs when not in scan mode
an_ac_gptr_scan_in	1	IN	scan input to core gptr rings
an_ac_time_scan_in	1	IN	scan input to core time rings
an_ac_repr_scan_in	1	IN	scan input to core repr rings
an_ac_abst_scan_in	8	IN	scan input to core abst rings
an_ac_func_scan_in	44	IN	scan input to core func rings
an_ac_bcfg_scan_in	5	IN	scan input to core bcfg rings
an_ac_dcfg_scan_in	3	IN	scan input to core dcfg rings
an_ac_rtim_scan_in	1	IN	scan input to core rtim rings
ac_an_gptr_scan_out	1	OUT	gptr ring scan output
ac_an_time_scan_out	1	OUT	time ring scan output
ac_an_repr_scan_out	1	OUT	repr ring scan output
ac_an_abst_scan_out	8	OUT	abst ring scan output
ac_an_func_scan_out	44	OUT	func ring scan output
ac_an_bcfg_scan_out	5	OUT	bcfg ring scan output
ac_an_dcfg_scan_out	3	OUT	dcfg ring scan output
ac_an_rtim_scan_out	1	OUT	rtim ring scan output

3. AT-Node A2-L2 Interface Protocol

3.1 General

- Shared L1 Dcache for all four A2 threads
- Shared L1 Icache for all four A2 threads
- L1 and L2 cacheline size is 64B
- A2 Core - L2 Interfaces
  - One A2-L2 Command Request Interface per A2 Core
    - Includes support for cacheable and non-cacheable loads and stores
    - Credit-based interface in which A2 core has separate load-ttype credits and store-ttype credits
  - One Store Data Interface per A2 Core
  - One Reload Interface per A2 Core
  - L2 can send reload data to the four A2 Cores simultaneously
  - One interface per A2 Core to handle the Back-Invalidate, TLBI-Snoop, ICSWX. status and IPI (Inter-Processor Interrupt) functions
    - Back-Invalidate interface is shared between the four independent functions
  - One Reservation Interface per A2 Core
  - One Sync Interface per A2 Core
  - One Pervasive Interface per A2 Core
- The AT Node L2 has a 4-entry CIU Load Queue and a 32-entry Store Gathering Queue per A2 Core
  - Interface and queues are used for cacheable and non-cacheable operations, and follow the same data paths in the L2, with the difference being that the non cacheable operations are not written to the L2 Cache.
- 2MB L2 with four 512KB slices shared across four A2 Cores
- Each slice has 12 requests per slice

3.2 Command Request Interface

- Single request interface for both loads and stores
  - Includes support for both cacheable and non-cacheable Loads and Stores
  - Since there are no Store Command Queues and Store data buffers in the A2 Core Interface logic, stores have priority over loads if they are both available to be requested (with corresponding credit available).
- Credit-based interface in which A2 core maintains separate load-ttype credits and store-ttype credits
  - A2 core is initialized to the maximum number of load-ttype credits and store-ttype credits supported by the L2
    - A2 core supports up to 8 load-ttype credits
    - A2 core supports up to 32 store-ttype credits
  - Corresponding credit count is decremented for each load-ttype or store-ttype request sent from the A2 core to the L2
  - Corresponding credit count is incremented for each “pop” or “gather” (store-ttype only) received from the L2
    - Load Pop (req_ld_pop) from L2 indicates one additional load-ttype credit is available
    - Store Pop (req_st_pop) from L2 indicates one additional store-ttype credit is available
    - Store Gather (req_st_gather) from L2 indicates one additional store-ttype credit is available
    - May get all three in the same cycle
  - A2 Core can send a command request to the L2 every cycle, if credits are available
- Request preceded by power token (req_pwr_token) by one cycle
  - May be speculative, but more accuracy = more power saved
- Store data request also preceded by store data power token (st_data_pwr_token) by one cycle
  - Occurs in same cycle as request power token
  - Asserted for store-ttypes that provide store data
  - Allows power gating of store data (up to 32 Bytes) and store byte enable (up to 32 bits) registers
- Valid bit, “req”, sent with request, due to speculative nature of req_pwr_token
  - Other Command Request signals are ignored unless the Valid Request bit, “req”, is set.
- No cancel of requests
  - All Command Requests are allowed to complete. If the A2 Core does not need the reload data, it throws it away.
- Debug Mode
  - In debug mode, there will be a configuration bit in the XUCR that, when set, indicates that the A2 Core can send one and only one load or store (but not both) when the A2 core has 1 store credit and 1 load credit. This bit is intended to be used to provide a single-step feature - or a close approximation of it - as a debug mode.

- Core_tag = 4 bits (encoded)
  - Passed back to A2 Core (via reld_core_tag) with reload data valid indication (reld_data_vld)
  - Core_tag is not used for back-invalidates. Back-invalidates are sent to i-side or d-side based on ttype
  - A2 Core will not reuse core_tag until reload data is back without error, or with “UE” (Uncorrectable ECC Error) indication
- TTYPE = 6 bits (encoded)
  - One bit of ttype, req_ttype(0), identifies destination (STQ vs LDQ)
  - ttypes are defined in Table 1-3 req_ttype(0:5) Encoding (as of v0.35) on page 9.
- RA = 42 bits
- WIMG (Write-through, Inhibited, Memory coherency required, Guarded), E (Endian mode), and U0-U3 (User-defined bits) sent with req
  - L2 does not do anything with W and M of WIMG, E. U2-U3
- Thread ID = 2 bits (encoded)
- DCBF Local or Global
  - Presented as two separate ttypes, dcbf(l) and dcbf(g)
- A2 Core Command Sequence Rules
  1. A2 Core will not send a second D-side load request if there is an outstanding load in the L2 for same cacheline address (regardless of thread).
  2. A2 Core will not send a second I-side ifetch request if there is an outstanding I-side ifetch in the L2 for same cacheline address (regardless of thread).
  3. However, the A2 Core may send one D-side load and one I-Side ifetch request to the same cacheline address. There is no ordering enforced between D-side and I-side loads.
  4. A2 Core will not send store to L2 if there is outstanding load in L2 for same cacheline address (regardless of thread)
  5. A2 Core may send load to L2 as early as 1 cycle following store to same cacheline address
     - L2 ensures that load gets the store’s data [regardless of thread]
  6. A2 Core will only send one I=1, G=1 load per thread to the A2-L2 Interface and will not send another I=1, G=1 load or store on that thread until reload data from the first I=1, G=1 load has returned. These I=1, G=1 requests must be ordered.
  7. A2 Core may send multiple I=1, G=0 loads to different 64-Byte granules (non-cacheable address terminology).
  8. A2 Core will not send a second I=1, G=0 load to the same 64-Byte granule as an outstanding I=1, G=0 load. I=1, G=0 loads to the same 64-Byte granule must be ordered.
- Load Transfer Length
  - Cacheable
    - I=0 Load always transfers 64-Byte cacheline
    - I=0 I-Fetch always transfers a 64-Byte cacheline
  - Non-cacheable
    - I=1 Load Transfer Length (req_ld_xfr_len) = 3 bits (encoded)

        - 1 byte, 2 byte, 4 byte, 8 byte, 16 byte, and 32 byte for non-cacheable load
        - 32 byte I=1 load is transferred over reload interface in two 16-Byte (quadword) data beats
      - I=1 I-Fetch always transfers 16 bytes - sent as 16 bytes in request, and returns 16 bytes from L2 to A2 Core.
- Store Transfer Length
  - I=0 or I=1 Store transfers 1 byte, 2 bytes, 4 bytes, 8 bytes, 16 bytes, or 32 bytes as indicated by the byte enables that accompany store data
- Store data
  - Sent with request
  - Sent with byte_enables, which indicate the bytes to store
  - Memory aligned
  - Quadword width (16B) or octword width (32B), under control of a mode bit in the A2 core
  - In 16-Byte Mode
    - I=0 or I=1 stores - will only have 1, 2, 4, 8, or 16 Bytes
    - No store crosses a 16-Byte boundary
  - In 32-Byte Mode
    - I=0 or I=1 stores - will only have 1, 2, 4, 8, 16, or 32 Bytes
    - 32 Byte stores are always operand aligned
    - A store may cross a 16-Byte boundary within the 32 Byte interface
    - No store crosses a 32-Byte boundary
  - ICSWX/ICSWX. use st_data field to carry the following:
    - st_data(0:31) - FRC, CT, MSR, etc., qualified by st_byte_en(0:3)
    - st_data(32:47) - ‘b00, PID(0:13), st_byte_en(4:5) not asserted
- No interface request (load or store) of 32 bytes will cross a 32-Byte (octword) boundary
- In 16-Byte mode only, no interface request (load or store) of 16 Bytes or less will cross a 16-Byte (quadword) boundary
  - If code is written such that an instruction would violate this rule, then microcode, trap handler, and exception handlers are used to handle these situations and present the requests in conformity with this rule. See the A2 Core User Manual for more details.

3.3 Reload Data Interface

- Reload control signals sent 3 cycles before data
  - Reload data Coming (required for back-to-back reload delivery L2 designs)
- Reload control signals sent 2 cycles before data
  - Data valid
  - DITC transfer indicator
  - Critical Quadword Identifier
  - Quadword identifier (RA[58:59])
  - Core_tag (5 bits)
- Reload data bus width is 16 Bytes (quadword)
- Data error
  - Sent 1 cycle after data beat (quadword)
  - There are two ECC error indications associated with a data beat (1 cycle after the beat is sent)
  - If an ECC error (reld_ecc_err) is indicated for any beat of a cache line:
    - A2 Core does not set the valid bit in the L1 Cache.
    - A2 Core writes the regfile if the data needed is error free, even if an ECC error is found on other data in the line.
    - L2 sends all four beats of original reload even if an error is encountered on an earlier beat.
    - L2 automatically resends the entire line some time later.
      - If an ECC error can be corrected, the corrected data will come across without an error indication upon resend.
      - If an ECC error cannot be corrected, the data will come across with an uncorrectable ECC error (UE) indication upon resend.
  - The original four beats of reload data and corresponding error/non-error indications are not allowed to overlap with the core_tag (sent w/ data valid, QW identifier) for any beat of the reload resend of that data.
    - If the A2 Core sees an ECC error for any of the 4 beats of data, with the last beat of data for the original four beats arriving in cycle X and the last ECC error/non-error indication arriving in cycle X+1, then the A2 Core will not see the core_tag for any beat of the reload resend before cycle X+2.
  - If an uncorrectable ECC error (reld_ecc_err_ue) is indicated for a data beat:
    - A2 Core writes the regfile if the data needed has an uncorrectable ECC error (UE), in order to make forward progress. In this case, a maskable checkstop will be set by both the A2 Core and L2.
    - A2 Core does not set the valid bit in the L1 Cache.
    - L2 sends all four beats of resend reload data even if an error is encountered on an early beat.
- Reload transfer rules
  - Every I=1 reload has 1 data beat, except for I=1 Load with 32B transfer length, which has two data beats

        - No 32B I=1 read for I-side; 16B only on I-side
    - Every I=0 reload has four 16-Byte (quadword) data beats
    - A mode bit dictates whether data for multiple-beat transfers must be sent in consecutive cycles, or in every other cycle. This mode is based upon the L2 cache data delivery needs. The L2 must transfer all reload data according to this mode. If the mode bit is not set for consecutive cycles, the following rules govern the sending of data and usage of the gaps in “every other cycle” mode:
        - Data beats for a reload operation have a gap of at most one cycle for a given octword transfer
        - Data beats for a reload operation could arrive as two data beats (paired quadwords of an octword) followed by a varying number of cycles until the remaining two data beats (paired quadwords of the other octword of the cacheline) arrive.
        - The gap may be filled with reload data for another reload transfer (i.e. two reloads may be interlaced)
- I=1 read data is memory-aligned
- A2 requests, but does not require critical quadword first
- The two quadwords of an octword must always be sent as a set of paired data beats

3.4 Back-Invalidate Interface

- One interface per A2 Core to handle the Back-Invalidate, TLBI-Snoop, and CPX functions
- The interface is shared between the three independent functions, but only one function will be using the interface on a given cycle. There is no overlap of the functions in a given cycle.
- A2 Core is designed to handle a new back-invalidate every cycle
- Back-Invalidate
  - In the case of a back-invalidate followed by a SYNC ACK, the A2 Core requires that the L2 delay the sync_ack for three cycles in order to give the core sufficient cycles to complete the back invalidation.
  - Back-invalidate bus target is specified in a 5-bit target field
    - [0] I-side (L1 cache back-invalidate)
    - [1] D-side (L1 cache back-invalidate)
    - [2] MMU (TLBI snoop)
    - [3] CPX (CoProcessor indirect status)
    - [4] IPI (Inter-Processor Interrupt message transfer)
  - Back-Invalidate bus target may have both I-side and D-side targets asserted concurrently.
  - Back-Invalidate bus targets for MMU, CPX, and IPI are mutually exclusive with each other and with the I-side/D-side targets.
- TLBI Snoop
  - A tlbi_reject signal will be sent from the A2 core to the L2 as a response to the snoop for any of the following conditions:
    - the core is operating in ERAT-only mode (CCR2[NOTLB]=1) and the tlbi snoop LPAR_ID does not match this core's LPIDR[LPID] value
    - the core is operating in MMU mode (CCR2[NOTLB]=0), this core's MMUCR1[TLBI_REJ]=1, and the tlbi snoop LPAR_ID does not match this core's LPIDR[LPID] value
  - A tlbi_complete command request will be sent from the A2 core to the L2 as a response to the snoop after the invalidation of the entry has been completed for any of the following conditions:
    - the core is operating in MMU mode (CCR2[NOTLB]=0) and this core's MMUCR1[TLBI_REJ]=0
    - the core is operating in ERAT-only mode (CCR2[NOTLB]=1) and the tlbi snoop LPAR_ID matches this core's LPIDR[LPID] value

3.5 LWARX/LDARX

- *lwarx*/*ldarx* bypasses L1 cache (i.e. data is not used if it hits in the L1)
- If *lwarx*/*ldarx* hits L1, then core invalidates line automatically, therefore, the L2 does NOT need to send back-invalidate for *lwarx*/*ldarx*
- *lwarx*/*ldarx* address is specifed to the 64-byte cache line. The reservation granule is the 64-Byte cache-line.
- Core will not send any newer instructions following *lwarx*/*ldarx* from the same thread to L2 until *lwarx*/*ldarx* is completed
- L2 tracks one reservation per thread
- Reservation is set before core receives reload data
- reservation_vld signal (used for fast wake-up from wait state) must be visible at the A2 before lwarx data is returned

3.6 STWCX./STDCX.

- Core automatically invalidates L1 if *stwcx./stdcx.* hits L1
- Core blocks issue behind a *stwcx./stdcx.* for that thread, but not for other threads, until it receives *stwcx./stdcx.* completion status
- L2 tracks one reservation per thread
- Reservation is cleared by a matching store reservation granule address from a different thread
- Reservation is cleared by certain snoop types to the same cache line from another processor
- Reservation is cleared by any *stwcx./stdcx.* (pass or fail) from this thread
- Reservation is cleared before *stwcx./stdcx.* completion sent to core
- *stwcx./stdcx.* pass condition
  - Pass if thread reservation is set and reservation granule address matches *stwcx./stdcx.* address at time of completion
- Completion handshake from L2 to core (8 bits):
  - stwcx_complete bit per thread (4 bits)
  - stwcx_pass bit per thread (4 bits)
- L2 may complete more than one *stwcx./stdcx.* per cycle
- Core may receive more than one *stwcx./stdcx.* completion per cycle
- L2 sends back-invalidate to other cores if *stwcx./stdcx.* pass, but not if fail

3.7 DCBT*

- Universally applicable to DCBT*
  - Request comes to L2 with load-ttype
  - L2 orders it after all older stores for same cacheline address
  - L1/L2 versions return reload data to core and set inclusive bit
  - L2-only version returns reload data to core, but does not set inclusive bit
    - Core throws away data for L2-only versions of the instructions
    - With the return of data, the core can now reuse the core_tag associated with the DCBT*
- [AT Node L2 Book IV] Unique Instruction Characteristics [AT Node implementation details]
  - DCBTST same as DCBT*, but additionally gets the line in exclusive state

3.8 DCBZ

- Treated as store
  - Request comes to L2 with store ttype
  - L2 orders it after all older stores for same cacheline address
  - If any other cores inclusive, they are back-invalidated
- Core invalidates L1 if hit (different than normal store)
- Data of all 0x00s written to L2 cache
- Line marked modified in L2 directory
- No dcbz32

3.9 DCBF

- Treated as store
  - Request comes to L2 with store ttype
  - L2 orders it after all older stores for same cacheline address
- L2 back invalidates all cores that have (L1-D or L1-I inclusive) == 1
- Additionally, if hit L1, local core invalidates self
- DCBF (L=0 or L=1) is sent to the L2
  - L=0 is Global dcbf and affects all processors (all levels of cache)
  - L=1 is Local dcbf and affects any processor sharing this cache
    - Implementation dependent (shared vs private cache)
  - Request ttype indicates Local (L=1) or Global (L=0) dcbf as dcbf(l) or dcbf(g), respectively.
- DCBF (L=3) invalidates L1 cache only and not lower levels of cache. This is not sent to the L2.

3.10 DCBST

- Treated as store
  - Request comes to L2 with store ttype
  - L2 orders it after all older stores for same cacheline address
- L2 does NOT back invalidate any cores
- L2 performs CLEAN function

3.11 ICBI

- Treated as store
  - Request comes to L2 with store-ttype
- L2 nops ICBI [At Node L2 implementation]
- The core will invalidate the I cache before sending the ICBI.
- When an L2 uses a cache coherency scheme that uses I-side inclusive bits to back-invalidate the L1 I-cache when a store is performed on an address, then an ICBI request received from the A2 core may be nop'ed in the L2 and the block need not be invalidated.
- L2 sends "store gather" or "store_pop" indication back to the core to free up a store-ttype credit
- [Does core send an icbi even if an outstanding load to this address exists? (No. This would normally be a st-hit-ld hazard that would not be sent.) Can a subsequent load have a ld-hit-st comparison against an icbi while it exists in the L2, or are comparisons disabled for icbi (and any other nop'd ttypes) from CREQ? (For AT Node L2 implementation, we will not set the valid bit, so we will not have a ld-hit-st to the icbi or any other nop'ed requests.)]

3.12 HWSYNC

1. HWSYNC will issue to XU when all outstanding commands for that thread have completed on the L2 interface. For that thread, this means that:
   a. all older requests have been sent on the A2-L2 interface, and
   b. all older data loads (including data cache block touches) have their reload data returned without ECC error or with the UE (uncorrectable ECC error) indication
2. IU blocks issue behind HWSYNC for that thread
3. HWSYNC flows down to XU store queue and the store command L2 interface
4. L2 sends sync_ack when HWSYNC completes
5. XU sends a memory barrier complete signal back to the IU
6. IU resumes issue for that thread with instructions following the HWSYNC

3.13 LWSYNC, MBAR

1. LWSYNC, MBAR will issue to XU when all outstanding commands for that thread have completed on the L2 interface. For that thread, this means that:
   a. all older requests have been sent on the A2-L2 interface, and
   b. all older data loads (including data cache block touches) have their reload data returned without ECC error or with the UE (uncorrectable ECC error) indication.
2. IU blocks issue behind LWSYNC, MBAR for that thread
3. LWSYNC, MBAR flows down to XU store queue and the store command L2 interface
4. XU sends LWBSYNC out to the L2 interface and sends a memory barrier complete signal back to the IU
5. IU resumes issue for that thread with instructions following the LWSYNC, MBAR
6. L2 puts LWSYNC, MBAR into store queue
7. L2 retires the LWSYNC, MBAR when it gets to the bottom of store queue

3.14 TLBSYNC

The tlbsync instruction operates in one of two modes, as set by a configuration bit set in the A2 Core. The L2 can be designed to operate in either of the two modes, or it can be designed to operate in both modes, with its own corresponding configuration bit in the L2 that will be set to match the behavior programmed in the A2 Core's configuration bit, as necessitated by the system design.

- The behavior of the tlbsync in one mode will mimic that of the lwsync instruction. In this mode, issuing of instructions behind the tlbsync can resume without waiting for a sync_ack signal from the L2.
- The behavior of the tlbsync in the other mode waits for a sync_ack from the L2 before issuing of instructions behind the tlbsync can resume.

You can see that if the A2 Core's configuration bit is set up to wait for a sync_ack but the L2 is not, then that core will hang waiting for a sync_ack that will never be sent.

For the AT Node, these mode bits need to be set such that the L2 will always execute the tlbsync on the powerbus and send the sync_ack back to the core.

Mode 0 Behavior: tlbsync w/o sync_ack required (lwsync-like)

1. IU issues TLBSYNC to XU when all outstanding commands for that thread have completed on the L2 interface. For that thread, this means that:
   a. all older requests have been sent on the A2-L2 interface, and
   b. all older data loads (including data cache block touches) have their reload data returned without ECC error or with the UE (uncorrectable ECC error) indication.
2. IU blocks issue behind TLBSYNC for that thread
3. TLBSYNC flows down to XU store queue and the store command L2 interface
4. XU sends TLBSYNC out to the L2 interface and sends a memory barrier complete signal back to the IU
5. IU resumes issue for that thread with instructions following the TLBSYNC
6. L2 puts TLBSYNC into store queue
7. L2 retires the TLBSYNC when it gets to the bottom of store queue [STQ sends to L2U slice when no more stores pending. L2U slice retires it.]

Mode 1 Behavior: tlbsync requiring sync_ack (hwsync-like)

With the proposal of using tlbsync as ptesync, A2 core will wait for a sync_ack back from system, either L2 or something else that is not the case for a regular tlbsync instruction.

1. IU issues TLBSYNC to XU when all outstanding commands for that thread have completed on the L2 interface. For that thread, this means that:
   a. all older requests have been sent on the A2-L2 interface, and
   b. all older data loads (including data cache block touches) have their reload data returned without ECC error or with the UE (uncorrectable ECC error) indication.
2. IU blocks issue behind TLBSYNC for that thread
3. TLBSYNC flows down to XU store queue and the store command L2 interface
4. XU sends TLBSYNC out to the L2 interface
5. L2 puts TLBSYNC into store queue
6. L2 broadcasts it on PowerBus as a ptesync bus command when it gets to the bottom of store queue
7. L2 waits for a non-retry combined response

8. L2 sends sync_ack to A2 Core (XU)
9. XU sends a memory barrier complete signal back to the IU
10. IU resumes issue for that thread with instructions following the TLBSYNC

3.15 DITC Messages

1. When the A2 core needs to send a DITC message, it will first issue 4 16B stores (with ttype = b”100000”) to a special address that was previously configured (one 64 byte address block will be configured per thread). These stores will have req_thread bit 2 set to a 1 to indicate that they are stores for a DITC. Since these stores will be from unique threads, they will not be affected by sync instructions.
2. The A2 core will then issue the DITC store (ttype = b”100010”) with thread bit 2 set to a 1.
3. When these stores are removed from the store queue, req_st_pop and req_st_pop_thrd(0:2) will be signaled to the A2 core to indicate their removal. Bit 2 of req_st_pop_thrd is used to indicate that the command being removed is the DITC store (ttype = b”100010”). Bits 0 and 1 of req_st_pop_thrd will point to the thread ID of the DITC. When the 4 16B stores are removed from the store queue, req_st_pop_thrd(2) will be set to 0 even though req_thread bit 2 was set on their request. On the st_pop, bit 2 is used to indicate that the DITC has been taken from the store queue and that message "buffer" has been transferred out of the store queue.

3.16 TLBIVAX Op

Generally speaking, a page is not invalidated at issue time, but at completion time. So there is a need to complete whatever translated prior to the invalidation. It is assumed that only one thread per core can source a TLBIVAX or ERATIVAX instruction at one time (i.e. a core level software lock is required). TLBIVAX instructions can only be issued when a source core is operating in MMU mode (i.e. CCR2[NOTLB]=0), and ERATIVAX instructions can only be issued when a core is operating in ERAT-only mode (i. e. CCR2[NOTLB]=1). Both the TLBIVAX and ERATIVAX instructions result in a downbound tlbivax request ttype operation on the core to L2 interface.

The sequence below only applies to the global TLBIVAX and ERATIVAX instructions. The local TLBILX and ERATILX instructions do not result in a request being sent on the A2-L2 interface.

1. Thread x issues a TLBIVAX or ERATIVAX instruction and will be stalled until after a local tlbi snoop is received (i.e. with the back_inv_local indicating this tlbi snoop was the result of an instruction on this core). The issuing thread re-issues only after the tlbi_complete has been sent from the core to the L2 (in the case of accepting the local snoop), or only after the local snoop is rejected. The other threads may continue issuing, unless prohibited from doing so due to other reasons. NOTE: The case of a local snoop rejection is viewed as an abnormal event and can only occur if (1) software running on thread A issues an ERATIVAX, and sometime later software running on thread B changes the LPIDR[LPID] value before the local snoop from thread A is received, or (2) software running on thread A issues a TLBIVAX while MMUCR1[TLBI_REJ]=1 on the issuing core, and the MAS5[SLPID] value used for the TLBIVAX does not match the LPIDR[LPID] value of the issuing core when the local snoop is received.
2. XU sends tlbivax request op to L2 when it gets to the bottom of XU store queue
3. L2 puts it into its store queue
4. The tlbivax request op trickles down to bottom of L2 store queue
5. The L2 moves the tlbivax op to an RC machine for processing.
6. The tlbivax op is broadcast to Bus and L2 sends tlbi snoop from the Bus back to the executing core and others via the back-invalidate interface. The attached L2 needs to see a tlbi_complete request or a snoop reject from the core after sending the tlbi snoop, whether it was the originating core or a snooped core.
7. Core determines whether or not it will reject the tlbi snoop or if will reply with a tlbi_complete to the L2:
   a. A tlbi_reject signal will be sent from the A2 core to the L2 as a response to the snoop for any of the following conditions:
      - the core is operating in ERAT-only mode (CCR2[NOTLB]=1) and the tlbi snoop LPAR_ID does not match this core's LPIDR[LPID] value
      - the core MMUCR1[TLBI_REJ]=1 and the tlbi snoop LPAR_ID does not match this core's LPIDR[LPID] value
   b. A tlbi_complete command request will be sent from the A2 core to the L2 as a response to the snoop after the invalidation of the entry has been completed for any of the following conditions:
      - the core is operating in MMU mode (CCR2[NOTLB]=0) and MMUCR1[TLBI_REJ]=0
      - the core is operating in ERAT-only mode (CCR2[NOTLB]=1) and the tlbi snoop LPAR_ID matches this core's LPIDR[LPID] value
8. If rejecting a tlbi snoop that originated from one of this core's threads (as determined by the L2 sourced back_inv_local signal) the originating thread will resume issuing. If rejecting a tlbi snoop that did not originate from this core, the originating thread continues to be stalled.

9. If issuing a tlbi_complete back to the L2, the core stalls all issue (all threads) and waits for pending ops, which might have been translated with the page that is targeted by tlbi snoop, to drain (i.e. stores go to L2 and data home for loads).
10. Core (MMU) invalidates TLB, and waits for D-ERAT and I-ERAT entry invalidation completion
11. Core sends tlbi_complete to L2 (at this point there are no pending loads in core but outstanding stores can be in L2 store queue)
12. Core removes issue stall for all threads, except any originating thread that still has an outstanding tlbivax or erativax instruction generated stall when this tlbi snoop was sourced from a remote core. A source thread stall release requires a locally sourced tlbi snoop (as indicated by the back_inv_local signal to the core).
13. L2 puts tlbi_complete into its store queue
14. The tlbi_complete trickles down to the bottom of L2 store queue (at this point there are no older stores in queue)
15. L2 resets tlbi snoop entry. Operation is complete.

The st_data field is used to carry information for a tlbivax request op (resulting from a tlbivax or erativax instruction) to the L2 for use on the nest PowerBus. The st_byte_enbl signals are not to be used for this st_data usage, as the information is targeted for the PBus, not the L2 cache. The st_data usage for tlbivax is depicted below in Table 4-1.

Table 3-1. st_data field description for tlbivax

st_data	Field Definition	Description
0:31	reserved	unused bits
32:39	pppp_pppp	LPAR_ID (Logical Partition Identifier)
40:44	r_rrrr	reserved for future expansion
45	IND	Indirect Entry bit
46	GS	Guest State bit
47	L	Large page size bit
48:255	reserved	unused bits

3.17 Reservation Handling

The reservation granule is 64B.

A reservation is lost when

1. The thread holding the reservation executes another lwarx, or ldarx: this clears the first reservation and establishes a new one.
2. The thread holding the reservation executes any stwcx., or stdcx., regardless of whether the specified address matches the address specified by the lwarx, or ldarx that established the reservation, and regardless of whether the storage operand lengths of the two instructions are the same and regardless if the stwcx. or stdcx. passes or fails.
3. Some other thread executes a Store, dcbz, or dcbzep that specifies a location in the same reservation granule.
4. Some mechanism other than a thread modifies a storage location in the same reservation granule. (examples: dma_write, Force Cache Inject, Regular Cache Inject done by IO or accelerator unit)
5. Implementation-specific characteristics of the coherence mechanism cause the reservation to be lost.
6. dcbf (Kill) --> Issued by a MCD for cleaning up the remote Shared copy which includes the reservation.

A reservation is not lost when

1. Some other thread executes a dcbtst, dcbtstep, or dcbtstls that specifies a location in the same reservation granule: the reservation is not lost.
2. Some other thread executes a dcba that specifies a location in the same reservation granule: the reservation is not lost. the target block is not newly established in a data cache; the instruction is no-op'ed.
3. Some other thread executes a dcbi that specifies a location in the same reservation granule: the reservation is not lost, the instruction is as a load
4. An interrupt occurs on the thread holding the reservation: the reservation is not lost. (However, system software invoked by interrupts may clear the reservation.)