OpenPiton Microarchitecture Specification

Wentzlaff Parallel Research Group

Princeton University

openpiton@princeton.edu

Revision History

Revision	Date	Author(s)	Description
1.0	04/02/16	Wentzlaff Parallel Research Group	Initial version

Contents

List of Figures

List of Tables

1 Introduction

This document introduces the OpenPiton microarchitecture specification. Openpiton adopts OpenSPARC T1 core, so this document focus on uncore components. For more information about microarchitecture of the core, please refer to the OpenSPARC T1 Micro architecture Specification [1].

The OpenPiton processor is a scalable, open-source implementation of the Piton processor, designed and taped-out at Princeton University by the Wentzlaff Parallel Research Group in March 2015. The RTL is scalable up to half a billion cores arranged in a 2D mesh, it is written in Verilog HDL, and a large test suite (∼9000 tests) is provided for simulation and verification. By making Piton open-source, we hope it would also be a useful tool to both researchers and industry engineers in exploring and designing future manycore processors.

This document covers the following topics:

- High-level architecture of OpenPiton
- Directory-based cache coherence protocol
- The L1.5 data cache
- The L2 cache
- On-chip network

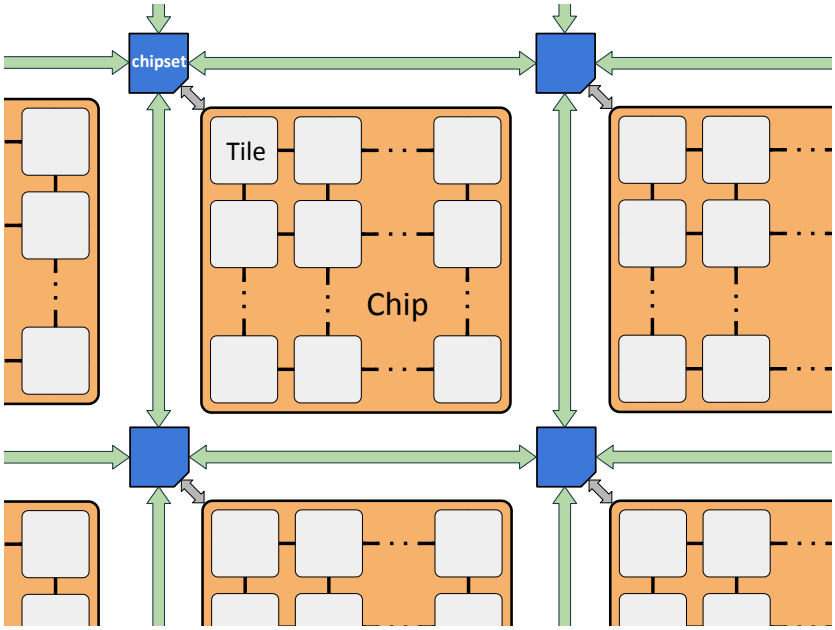

Figure 1: OpenPiton Architecture. Multiple manycore chips are connected together with chipset logic and networks to build large scalable manycore systems. OpenPiton's cache coherence protocol extends off chip.

2 Architecture

OpenPiton is a tiled-manycore architecture, as shown in Figure 1. It is designed to be scalable, both intra-chip and inter-chip.

Intra-chip, tiles are connected via three networks on-chip (NoCs) in a 2D mesh topology. The scalable tiled architecture and mesh topology allow for the number of tiles within an OpenPiton chip to be configurable. In the default configuration, the NoC router address space supports scaling up to 256 tiles in each dimension within a single OpenPiton chip (64K cores/chip).

Inter-chip, an off-chip interface, known as the chip bridge, connects the tile array (through the tile in the upper-left) to off-chip logic (chipset), which may be implemented on an FPGA or ASIC. The chip bridge extends the three NoCs off-chip, multiplexing them over a single link.

The extension of NoCs off-chip allows the seamless connection of multiple OpenPiton chips to create a larger system, as illustrated in Figure 1. The cache-coherence protocol extends off-chip as well, enabling shared-memory across multiple chips. A core in one chip may be cache coherent with cores in another chip.

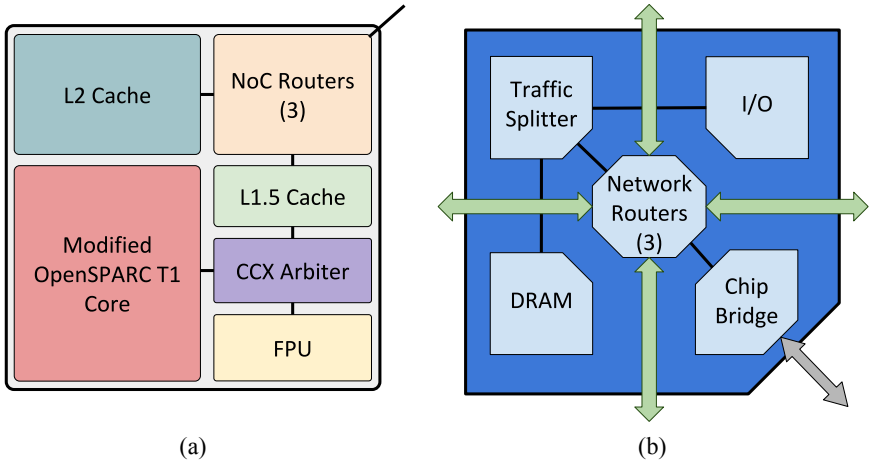

Figure 2: Architecture of (a) a tile and (b) chipset.

This enables the study of even larger shared-memory manycore systems.

2.1 Tile

The architecture of a tile is shown in Figure 2a. A tile consists of a modified OpenSPARC T1 core, an L1.5 cache, an L2 cache, a floating-point unit (FPU), a CCX arbiter, and three NoC routers.

The L2 and L1.5 caches connect directly to all three NoC routers and all messages entering and leaving the tile traverse these interfaces. The CPU Cache-Crossbar (CCX) is the crossbar interface used in the OpenSPARC T1 to connect the cores, L2 cache, FPU, I/O, etc. The L1.5 is responsible for transducing between CCX and the NoC router protocol. The CCX arbiter de-multiplexes memory and floating-point requests from the core to the L1.5 cache and FPU, respectively, and arbitrates responses back to the core.

2.2 Core

OpenPiton uses the OpenSPARC T1 core with minimal modifications. This core was chosen because of its industry-hardened design, multi-threaded capability, simplicity, and modest silicon area requirements. Equally important, the OpenSPARC framework has a stable code base, implements a stable ISA with compiler and OS support, and comes with a large test suite.

In the taped out Piton core, the default configuration for Open-

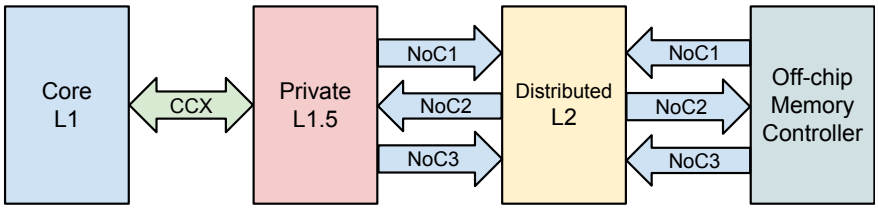

Figure 3: OpenPiton's memory hierarchy datapath.

Piton, the number of threads is reduced from four to two. This was primarily done to reduce the area requirement of the core and to reduce the pressure on the memory system. By default, the stream processing unit (SPU), essentially a cryptographic functional unit, is also removed from the core to save area. Instructions intending to utilize the SPU will trigger a trap and are emulated in software. The default TLB size is 16 entries to reduce area (which reflects the reduction in the number of threads), but it can be increased to 64 or decreased down to 8 entries.

An additional set of configuration registers were added which allow for extensibility within the core. The configuration registers are implemented as memory-mapped registers in an alternate address space (one way to implement CPU control registers in SPARC). These configuration registers can be useful for adding additional functionality to the core which can be configured from software, e.g. enabling/disabling functionality, configuring different modes of operation, etc.

Many of the modifications to the OpenSPARC T1 core, like the ones above, have been made in a configurable way. Thus, it is possible to configure the core to the original OpenSPARC T1 specifications or a set of parameters different from the original OpenSPARC T1 as is the default OpenPiton core.

2.3 Cache Hierarchy

OpenPiton's cache hierarchy is composed of three cache levels, with private L1 and L1.5 caches and a distributed, shared L2 cache. Each tile in OpenPiton contains an instance of the L1 cache, L1.5 cache, and L2 cache. The data path through the cache hierarchy is shown in Figure 3.

2.3.1 L1 Cache

The L1 cache is reused, with minimal modifications, from the OpenSPARC T1 design. It is tightly integrated to the OpenSPARC T1 pipeline, and composed of two separate caches: the L1 data cache and L1 instruction cache. The L1 data cache is an 8KB write-through cache; it is 4-way set-associative and the line size is 16-bytes. The 16KB L1 instruction cache is similarly 4-way set associative but with a 32-byte line size. Both L1 caches' sizes can be configured.

There are two fundamental issues with the original OpenSPARC T1 L1 cache design which made it suboptimal for use in a scalable multicore and hence required changes for use in OpenPiton. First, write-through caches require extremely high write-bandwidth to the next cache level, which is likely to overwhelm and congest NoCs in manycore processors with distributed caches. This necessitates a local write-back cache. Second, the cache communication interface needs to be compliant with OpenPiton's cache coherence protocol, i.e., tracking MESI cache line states, honoring remote invalidations, and communicating through OpenPiton's NoCs instead of the OpenSPARC T1's crossbar. Rather than modifying the existing RTL for the L1s, we introduced an extra cache level (L1.5) to satisfy the above requirements.

2.3.2 L1.5 Data Cache

The L1.5 serves as both the glue logic, transducing the OpenSPARC T1's crossbar protocol to OpenPiton's NoC coherence packet formats, and a write-back layer, caching stores from the write-through L1 data cache. It is an 8KB 4-way set associative write-back cache (the same size as the L1 data cache by default) with configurable associativity. The line size is the same as the L1 data cache at 16-bytes.

The L1.5 communicates requests and responses to and from the core through CCX. The CCX bus is preserved as the primary interface to the OpenSPARC T1. The L1.5 CCX interface could relatively easily be replaced with other interfaces like AMBA® or AXI to accommodate different cores. When a memory request results in a miss, the L1.5 translates and forwards request to the L2 through the network-on-chip (NoC) channels. Generally, the L1.5 issues requests on NoC1, receives data on NoC2, and writes back modified cache lines on NoC3, as shown in Figure 3.

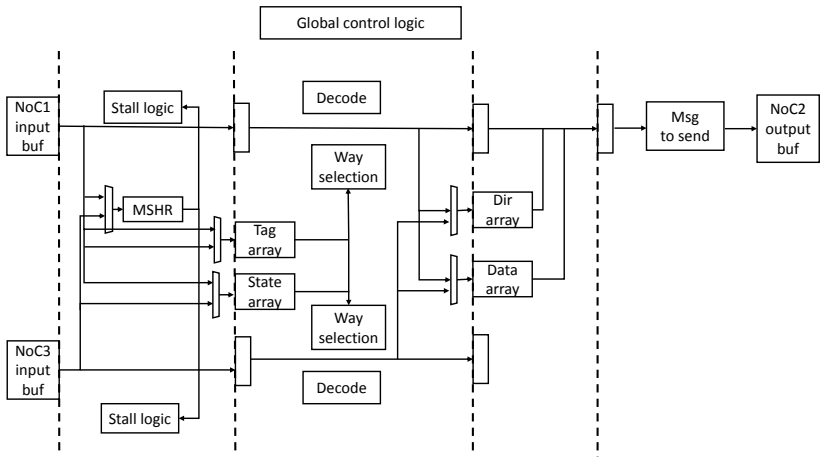

Figure 4: The architecture of the L2 cache.

While the L1.5 was named as such during the development of the OpenPiton ASIC prototype, in traditional computer architecture contexts it would be appropriate to call it the "private L2" and to call the next level cache the "shared/distributed L3". The L1.5 is inclusive of the L1 data cache; each can be independently sized with independent eviction policies. As a space- and performance-conscious optimization, the L1.5 does not cache instructions– these cache lines are bypassed directly between the L1 instruction cache and the L2. It would be possible to modify the L1.5 to also cache instructions.

More detailed interface descriptions and design notes of the L1.5 can be found in Chapter 4.

2.3.3 L2 Cache

The L2 cache is a distributed write-back cache shared by all tiles. The default cache configuration is 64KB per tile and 4-way set associativity, but both the cache size and associativity are configurable. The cache line size is 64 bytes, larger than caches lower in the hierarchy. The integrated directory cache has 64 bits per entry, so it can precisely keep track of up to 64 sharers by default.

The L2 cache is inclusive of the private caches (L1 and L1.5). Cache line way mapping between the L1.5 and the L2 is independent and is entirely subject to the replacement policy of each cache. In fact, since the L2 is distributed, cache lines consecu-

tively mapped in the L1.5 are likely to be strewn across multiple L2 tiles (L2 tile referring to a portion of the distributed L2 cache in a single tile). By default, OpenPiton maps cache lines using constant strides with the lower address bits across all L2 tiles, but Coherence Domain Restriction (CDR) [2], an experimental research feature integrated into OpenPiton, can be used to interleave cache lines belonging to a single application or page across a software-specified set of L2 tiles.

As shown in Figure 12, the L2 cache is designed with dual parallel pipelines. The first pipeline (top) receives cache miss request packets from lower in the cache hierarchy on NoC1 and sends memory request packets to off-chip DRAM and cache fill response packets to lower in the cache hierarchy on NoC2. The second pipeline (bottom) receives memory response packets from off-chip DRAM and modified cache line writeback packets from lower in the cache hierarchy on NoC3. The first L2 pipeline (top) contains 4 stages and the second pipeline (bottom) contains only 3 stages since it does not transmit output packets. The interaction between the L2 and the three NoCs is also depicted in Figure 3.

More information regarding L2 can be found in Chapter 5.

2.4 Cache Coherence and Memory Consistency Model

The memory subsystem maintains cache coherence with a directory-based MESI coherence protocol. It adheres to the TSO memory consistency model used by the OpenSPARC T1. Coherent messages between L1.5 caches and L2 caches communicate through three NoCs, carefully designed to ensure deadlock-free operation.

The L2 is the point of coherence for all memory requests, except for non-cacheable loads and stores which directly bypass the L2 cache. All other memory operations (including atomic operations such as compare-and-swap) are ordered and the L2 strictly follows this order when servicing requests.

The L2 also keeps the instruction and data caches coherent. Per the OpenSPARC T1's original design, coherence between the two L1 caches is maintained at the L2. When a line is present in a core's L1I and is loaded as data, the L2 will send invalidations to the relevant instruction caches before servicing the load.

Some of the high-level features of the coherence protocol include:

- 4-step message communication
- Silent eviction in Exclusive and Shared states
- No acknowledgments for dirty write-backs
- Three 64-bit physical NoCs with point-to-point ordering
- Co-location of L2 cache and coherence directory

We document the coherence protocol in more detail in Chapter 3.

2.5 Interconnect

There are two major interconnection types used in OpenPiton, the NoCs and the chip bridge.

2.5.1 Network On-chip (NoC)

There are three NoCs in an OpenPiton chip. The NoCs connect tiles in a 2D mesh topology. The main purpose of the NoCs is to provide communication between the tiles for cache coherence, I/O and memory traffic, and inter-core interrupts. They also route traffic destined for off-chip to the chip bridge. The NoCs maintain point-to-point ordering between a single source and destination, a feature often leveraged to maintain TSO consistency. In a multi-chip configuration, OpenPiton uses similar configurable NoC routers to route traffic between chips.

The three NoCs are physical networks (no virtual channels) and each consists of two 64-bit uni-directional links, one in each direction. The links use credit-based flow control. Packets are routed using dimension-ordered wormhole routing to ensure a deadlock-free network. The packet format preserves 29 bits of core addressability, making it scalable up to 500 million cores.

To ensure deadlock-free operation, the L1.5 cache, L2 cache, and memory controller give different priorities to different NoC channels; NoC3 has the highest priority, next is NoC2, and NoC1 has the lowest priority. Thus, NoC3 will never be blocked. In addition, all hardware components are designed such that consuming a high priority packet is never dependent on lower priority traffic. While the cache coherence protocol is designed to be logically deadlock free, it also depends on the physical layer and routing to also be deadlock free.

Classes of coherence operations are mapped to NoCs based on the following rules, as depicted in Figure 3:

- NoC1 messages are initiated by requests from the private cache (L1.5) to the shared cache (L2).
- NoC2 messages are initiated by the shared cache (L2) to the private cache (L1.5) or memory controller.
- NoC3 messages are responses from the private cache (L1.5) or memory controller to the shared cache (L2).

2.5.2 Chip Bridge

The chip bridge connects the tile array to the chipset, through the upper-left tile, as shown in Figure 1. All memory and I/O requests are directed through this interface to be served by the chipset. Its main purpose is to transparently multiplex the three physical NoCs over a single, narrower link in pin-limited chip implementations.

The chip bridge contains asynchronous buffers to bridge between I/O and core clock domains. It implements three virtual off-chip channels over a single off-chip physical channel, providing the necessary buffering and arbitration logic. The off-chip channel contains two 32-bit unidirectional links, one in each direction, and uses credit-based flow control. At 350MHz, our chip implementation's target I/O frequency, the chip bridge provides a total bandwidth of 2.8GB/s.

2.6 Chipset

The chipset, shown in Figure 2b, houses the I/O, DRAM controllers, chip bridge, traffic splitter, and inter-chip network routers. The chip bridge brings traffic from the attached chip into the chipset and de-multiplexes it back into the three physical NoCs. The traffic is passed to the inter-chip network routers, which routes it to the traffic splitter if it is destined for this chipset. The traffic splitter multiplexes requests to the DRAM controller or I/O devices, based on the address of the request, to be serviced. If the traffic is not destined for this chipset, it is routed to another chipset according to the inter-chip routing protocol. Traffic destined for the attached chip is directed back through similar paths to the chip bridge.

2.6.1 Inter-chip Routing

The inter-chip network router is configurable in terms of router degree, routing algorithm, buffer size, etc. This enables flexible exploration of different router configurations and network topologies. Currently, we have implemented and verified crossbar, 2D mesh, 3D mesh, and butterfly networks. Customized topologies can be explored by re-configuring the network routers.

We have proven that our network routing protocols can safely extend and be further routed (under some constraints) off chip while maintaining their deadlock-free nature.

2.7 Floating-point Unit

We utilize the FPU from OpenSPARC T1 [3]. In OpenPiton, there is a one-to-one relationship between cores and FPUs, in contrast to the OpenSPARC T1, which shares one FPU among eight cores [4]. This was primarily done to boost floating-point performance and to avoid the complexities of having shared FPUs among a variable number of tiles and providing sufficient floating-point performance. The CCX arbiter always prioritizes the L1.5 over the FPU in arbitration over the shared CCX interface into the core.

3 Cache Coherence Protocol

The OpenPiton processor adopts a directory-based MESI coherence protocol to maintain cache coherence among all processor cores. The coherence protocol operates between the private write-back L1.5 cache, distributed shared L2 cache, and memory controller, consisting of a set of packet formats and their expected behaviors. It is carefully crafted to ensure deadlock-free operation using network-on-chip (NoC) channels–specifically three NoCs with 64-bit channels. The L2 is inclusive of private caches, so each private cache line also has a copy in the L2. The directory information is also embedded in the L2 cache on a per-line basis, so coherence is tracked at L2 cache line level (64B).

Some of the high-level features of our coherence protocol include:

- 4-hop message communication (no direct communication between L1.5s)
- Silent eviction in E and S states
- No need for acknowledgement upon write-back of dirty lines from L1.5 to L2
- Use 3 physical NoCs with point-to-point ordering to avoid deadlock
- The directory and L2 are co-located but state information are maintained separately

Implementations of L1.5, L2, and memory controller need to give priority to packets of higher priority; namely, a request of lower priority cannot block a request of higher priority. In fact, NoC3 is will never be blocked in our design. While the cache coherence protocol is designed to be logically deadlock free, it also depends on the physical layer to be deadlock free. See the Section on NoC design for more detail.

3.1 Coherence operations

Non-cacheable loads and stores directly bypass the L2 cache so they are not considered as coherence operations (Non-cacheable loads still use L2 cache lines as temporary storage but invalidate the remaining data after completion). Classes of coherence operations are mapped to NoCs as followed: NoC1 messages are initiated by the private cache (L1.5) to shared cache (L2).

- ReqRd: includes load, ifill, prefetch (unused).
- ReqExRd: includes store, block stores (unused).
- ReqAtomic: includes compare-and-swap (CAS), swap.
- ReqWBGuard: is needed to prevent read requests from overtaking writeback requests and violate the memory ordering protocol.

Note: The prefetch instruction is unused in our memory system (treated as nop). Block stores are decomposed into regular stores in the L1.5 so they are not seen by the L2.

NoC2 messages are initiated by shared cache (L2) to private cache (L1.5), or memory controller.

- FwdRd: downgrades a private cache line due to a read request.
- FwdExRd: downgrades a private cache line due to an exclusive read request.
- Inv: invalidates a line from S $\rightarrow$ I.
- LdMem: loads a cache line from memory.
- StMem: stores a cache line to memory.
- AckDt: acknowledges the original request with data.
- Ack: acknowledges the original request without data.

NoC3 messages are responses from private cache to shared cache.

- FwdRdAck: L1.5 acknowledges the FwdRd operation.
- FwdExRdAck: L1.5 acknowledges the FwdExRd operation.
- InvAck: L1.5 acknowledges the FwdInv operation.
- LdMemAck: memory controller acknowledges the LdMem operation.
- StMemAck: memory controller acknowledges the StMem operation.
- ReqWB: L1.5 evicts/writes back dirty data.

Note how requests in NoC1 depends on responses from NoC2 to finish, and that some operations in NoC2 (ie. Inv) initiates operations in NoC3 (ie. ReqWB).

3.2 Coherence transaction

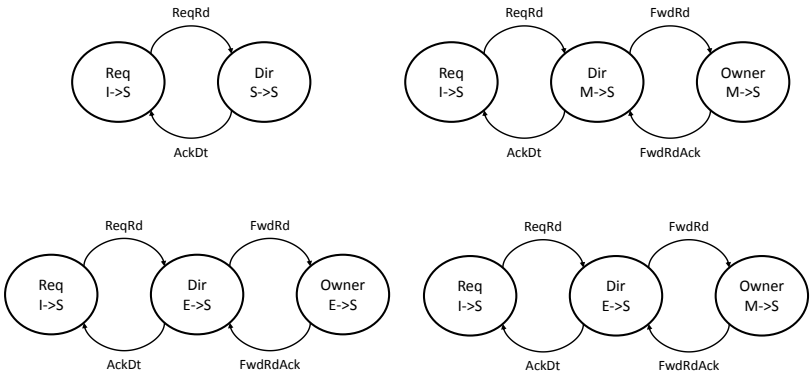

Figure 5: I→S state transition diagrams

Figure 5 shows all possible transition diagrams for the requester (L1.5 cache) from I state to S state when issuing a read request. If the directory is already in S state, it directly returns the requested data. This results in a two-hop transaction. Otherwise, the directory needs to send a downgrade request to the owner (a remote L1.5 cache) first which results in a 4-hop transaction. Notice that if the directory is in E state, the owner could be either in E or M state because of the possible silent transition from E to M state.

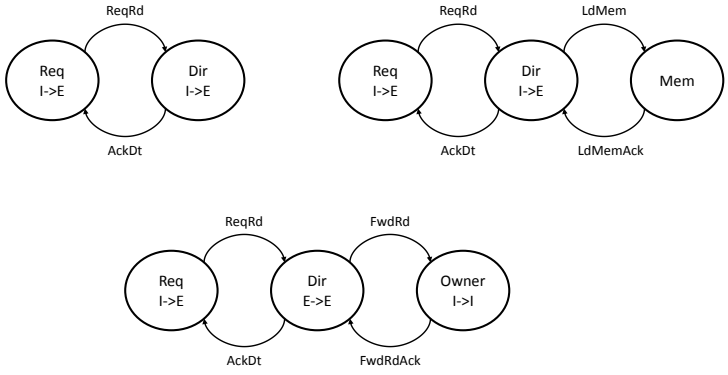

Figure 6: I→E state transition diagrams

Figure 6 shows all possible transition diagrams for the requester (L1.5 cache) from I state to E state when issuing a read request. If the directory is in I state but the cache line exists in L2, it directly returns the requested data. Otherwise if the cache line is not in L2, the L2 has to load the data for the memory first. Another case is the directory is in E state, and after a downgrade it detects the owner has already invalidated itself in silence, so the requester still ends up in E state.

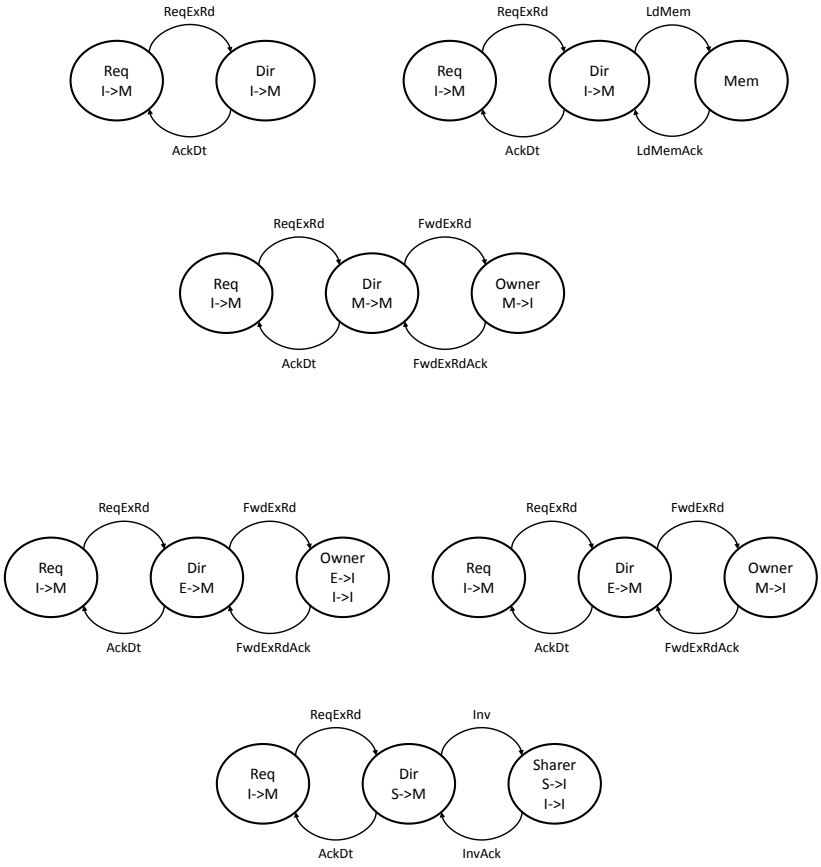

Figure 7: I→M state transition diagrams.

Figure 7 shows all possible transition diagrams for the requester (L1.5 cache) from I state to M state when issuing an exclusive read request. If the directory is in I state, it directly returns the requested data (may need to fetch it from the memory first if the data does not exist in L2). Otherwise the directory downgrades the owner (in E or M state) or invalidates the sharers (in S state) first before response.

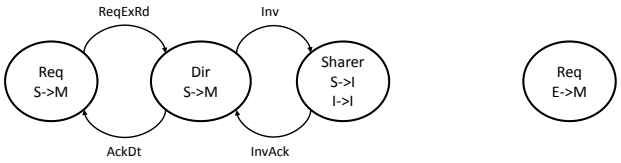

Figure 8: S→M and E→M state transition diagrams

The requester will transition from S to M state upon receiving an exclusive read request. The transition from E to M state is silent. The detail transition diagrams are shown in Figure 8.

Figure 9: L1.5 eviction state transition diagrams

Figure 9 shows all possible transition diagrams for a cache line eviction in the L1.5 cache. If the line is in E or S state, the eviction is silent without notify the directory. Otherwise if the line is in M state, the dirty data needs to be written back. In our design, we try to avoid acknowledgement for write-back requests to improve performance. In order to avoid race conditions, the write-back request is sent through NoC3 instead of NoC1. Since all NoCs maintain point-to-point ordering, this guarantees that

if another L1.5 requests the same cache line at the same time and the request arrives the directory before the write-back, the downgrade will not bypass the write-back request. Another race condition comes from the same L1.5 receives a load/store and sending out a read or exclusive read request to NoC1 after sending back the write-back request to NoC3. In order to avoid the later read/exclusive read request arrives in the directory first, a write-back guard message is also inserted into NoC1 upon sending out a write-back request to NoC3. The purpose of the write-back guard message is to prevent later request to the same cache line to be processed in the directory before the write-back request.

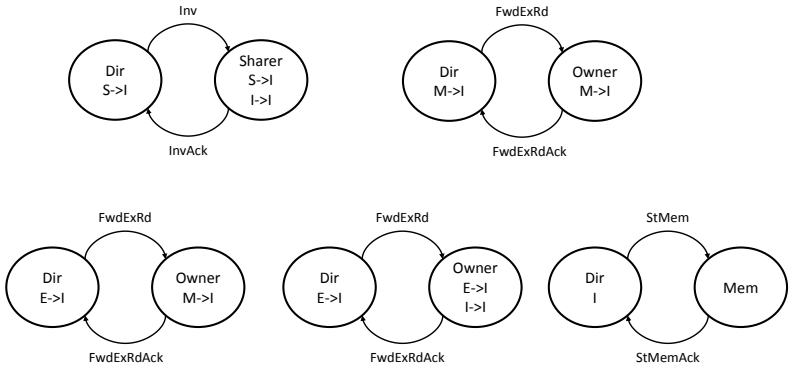

Figure 10: L2 eviction state transition diagrams

Figure 10 shows all possible transition diagrams for a cache line eviction in the L2 cache. Since the L2 is inclusive and the directory information is embedded with each L2 cache line, so a L2 eviction also invalidates all copies in private caches and the directory information. The directory sends out invalidations (in S state) or downgrade (in E or M state) to invalidate sharers or the owner and transition to I state in the end. Then the evicted cache line is written back to memory if it is dirty.

3.3 Packet formats

Request and response messages–from L1.5 to L2, L2 to L1.5, L2 to memory controller, etc...–are segmented to physical packets, or *flits*, 64-bit in size. Some messages, like a write back from L2 to memory, require as much as eleven flits, while others, like

	FIELDS	WIDTH	HI	LO
FLIT1	CHIPID	14	63	50
	XPOS	8	49	42
	YPOS	8	41	34
	FBITS	4	33	30
	PAYLOAD LENGTH	8	29	22
	MESSAGE TYPE	8	21	14
	MSHR/TAG	8	13	6
	OPTIONS1	6	5	0
FLIT2	ADDRESS	48	63	16
	OPTIONS2	16	15	0
FLIT3	SRC CHIPID	14	63	50
	SRC X	8	49	42
	SRC Y	8	41	34
	SRC FBITS	4	33	30
	OPTIONS3	30	29	0

Table 1: Packet header format for coherence requests

	FIELDS	WIDTH	HI	LO
FLIT1	CHIPID	14	63	50
	XPOS	8	49	42
	YPOS	8	41	34
	FBITS	4	33	30
	PAYLOAD LENGTH	8	29	22
	MESSAGE TYPE	8	21	14
	MSHR/TAG	8	13	6
	OPTIONS4	6	5	0

Table 2: Packet header format for coherence responses

	FIELDS	WIDTH	HI	LO
FLIT1	CHIPID	14	63	50
	XPOS	8	49	42
	YPOS	8	41	34
	FBITS	4	33	30
	PAYLOAD LENGTH	8	29	22
	MESSAGE TYPE	8	21	14
	MSHR/TAG	8	13	6
	RESERVED	6	5	0
FLIT2	ADDRESS	48	63	16
	OPTIONS2	16	15	0
FLIT3	SRC CHIPID	14	63	50
	SRC X	8	49	42
	SRC Y	8	41	34
	SRC FBITS	4	33	30
	RESERVED	30	29	0

Table 3: Packet header format for memory requests from L2 (NoC2)

	FIELDS	WIDTH	HI	LO
FLIT1	CHIPID	14	63	50
	XPOS	8	49	42
	YPOS	8	41	34
	FBITS	4	33	30
	PAYLOAD LENGTH	8	29	22
	MESSAGE TYPE	8	21	14
	MSHR/TAG	8	13	6
	RESERVED	6	5	0

Table 4: Packet header format for memory responses from memory controller to L2

a simple invalidation ack, as little as one flit. Table 1 presents the packet header format for coherence requests, while Table 2 shows the packet header format for coherence responses. Packet header formats for memory requests and responses are shown in Table 3 and Table 4.

The following tables detail the general packet format used in Piton.

3.3.1 Packet fields

CHIPID, XPOS, YPOS CHIPID, XPOS, and YPOS (and the SRC CHIPID, SRC X, SRC Y fields) collectively identify the precise location of the destination (source) tile. CHIPID identifies which chip, while XPOS and YPOS indicate which tile on said chip.

The packet format allocated 30-bits for tile address space, theoretically allows the design to scale up to 1-Mega tiles. OpenPiton's cache system (L1.5 & L2) is indeed designed to handle up to 30-bits of tile addressing, and should be forward-compatible with future designs.

	PORT	VALUE
FBITS	WEST	4'b0010
	SOUTH	4'b0011
	EAST	4'b0100
	NORTH	4'b0101
	PROCESSOR	8'b0000

Table 5: FBITS configurations.

FBITS FBITS (final destination bits) field informs the destination tile router where to push the packets to. Table 5 shows all possible configurations of FBITS. Currently possible targets include L1.5, L2, and the off-chip interface to memory controller. The L1.5 and L2 share the same FBITS (the PROCESSOR port).

In the current implementation, most destinations can be inferred semantically based on the message type, but FBITS field is still required for the router to forward packets to the off-chip interface in the correct direction (N/E/S/W).

PAYLOAD LENGTH PAYLOAD LENGTH field, located in the first flit of a message, indicates to both the router and re-

MESSAGE TYPE	VALUE	DESCRIPTION
RESERVED	8'd0	reserved
LOAD_REQ	8'd31	load request
PREFETCH_REQ	8'd1	prefetch request, unused
STORE_REQ	8'd2	store request
BLK_STORE_REQ	8'd3	block-store request, unused
BLKINIT_STORE_REQ	8'd4	block-store init request, unused
CAS_REQ	8'd5	compare and swap request
CAS_P1_REQ	8'd6	first phase of a CAS request, only used within L2
CAS_P2Y_REQ	8'd7	second phase of a CAS request if the comparison returns true, only used within L2
CAS_P2N_REQ	8'd8	second phase of a CAS request if the comparison returns false, only used within L2
SWAP_REQ	8'd9	atomic swap request
SWAP_P1_REQ	8'd10	first phase of a swap request
SWAP_P2_REQ	8'd11	second phase of a swap request
WB_REQ	8'd12	write-back request
WBGUARD_REQ	8'd13	write-back guard request
NC_LOAD_REQ	8'd14	non-cacheable load request
NC_STORE_REQ	8'd15	non-cacheable store request
INTERRUPT_FWD	8'd32	interrupt forward
LOAD_FWD	8'd16	downgrade request due to a load request
STORE_FWD	8'd17	downgrade request due to a store request
INV_FWD	8'd18	invalidate request
LOAD_MEM	8'd19	load request to memory
STORE_MEM	8'd20	store request to memory
LOAD_FWDACK	8'd21	acknowledgement of a LOAD_FWD request
STORE_FWDACK	8'd22	acknowledgement of a STORE_FWD request
INV_FWDACK	8'd23	acknowledgement of a INV_FWD request
LOAD_MEM_ACK	8'd24	acknowledgement of a LOAD_MEM request
STORE_MEM_ACK	8'd25	acknowledgement of a STORE_MEM request
NC_LOAD_MEM_ACK	8'd26	acknowledgement of a NC_LOAD_REQ request, unused
NC_STORE_MEM_ACK	8'd27	acknowledgement of a NC_STORE_REQ request
NODATA_ACK	8'd28	acknowledgement to L1.5 without data
DATA_ACK	8'd29	acknowledgement to L1.5 with data
ERROR	8'd30	error message, unused
INTERRUPT	8'd33	interrupt
L2_LINE_FLUSH_REQ	8'd34	flush a specific L2 cache line, used only within L2
L2_DIS_FLUSH_REQ	8'd35	displacement of a L2 cache line, used only within L2

Table 6: Message types used in the memory system.

ceiving target how long the message is. The additional flits are either inherent (eg. an L1.5 request always have two additional flits), or optional flits containing cache line or other data.

MESSAGE TYPE MESSAGE TYPE field contains the message type ID shown below in Table 6.

MSHR/TAG MSHR/TAG field contains the MSHR (miss-status-handling-register) ID, or tag, of the message. The sender includes this ID in the request in order to correctly identify the responses.

FIELDS	WIDTH	HI	LO
Options 1			
RESERVED	6	5	0
Options 2			
SUBCACHELINE BIT VECTOR	4	15	12
ICACHE BIT (CACHE TYPE)	1	11	11
DATA SIZE	3	10	8
Options 3			
RESERVED	30	29	0
Options 4			
FILL CACHELINE STATE	2	5	4
L2 MISS	1	3	3
ADDRESS_5_4	2	2	1
LAST SUB-CACHELINE	1	0	0

Table 7: Option fields for requests and responses

OPTIONS OPTIONS1/2/3/4 fields add additional information to a request or response. Table 7 aggregates these optional fields. Note that OPTIONS1/3 are empty and reserved for future use.

OPTIONS2 field contains three sub-fields: sub-cache-line bit vector, icache bit, and data-size.

Sub-cache-line bit vector is valid for the *Inv/FwdERd/FwdEWr/FwdMRd/FwdMWr* message types, where the L2 requests L1.5 to invalidate/downgrade 16-byte portions of a 64-byte cache line. The bit vector is also used by the L1 in the *FwdAckDt* message to indicate which sub cache line portion is dirty.

The icache bit indicates that a *ReqRd* message is meant for the instruction cache.

VALUE	Transaction Size (Bytes)
3'b000	0
3'b001	1
3'b010	2
3'b011	4
3'b100	8
3'b101	16
3'b110	32
3'b111	64

Table 8: Data Size field

Data-size subfield indicates the transaction size. It is necessary for write-through, non-cacheable, and atomic operations. Since the packets are always 64-bit long, data addressing is aligned to 8 bytes (for example, if the core sends an 1-byte non-cacheable store request to address 0x05 to the memory, the request will then have 1 data packet, containg 8 bytes from 0x00 to 0x07). In addition, the L2 always expects 8 packets in the response. If data size is less than 64 bytes, packets will be repeated to fulfill the 8 packets requirement (for example, if the core sends an 1-byte non-cacheable load request to address 0x05 to the memory, the response will have 8 data packets, each containing 8 bytes from 0x00 to 0x07).

OPTIONS4 field contains four sub-fields: fill-cacheline-state, L2 miss, address_5_4, and last-sub-cacheline.

The fill-cache-line-state field provides the state (M/E/S) of the cache line that L1.5 should have when filling the data cache.

The L2 miss field is an optional field for L2 to indicate the L1.5 whether the cache line request was a miss in the L2 cache.

The last-sub-cache-line bit, when used with invalidation/downgrade acks, indicate that this message is the last of possibly several write backs of sub-cache-lines.

4 L1.5 Data Cache

In the following sections, we will delve in-depth the more peculiar functionalities in the L1.5 such as write-backs, way-map table, and interfacing with the core through CCX.

4.1 Interfacing with L1I

Instruction cache lines are not cached in the L1.5; regardless they still need to be cache coherent. The L1.5 does not keep track of the L1I, rather, it delegates the instruction cache line tracking to the L2, and only forwards data and other coherence operations (like invalidation) to the core through the CPX bus.

OpenPiton modifies the CCX bus protocol slightly to accommodate cache reconfigurability, namely to be able to address bigger and wider L1s than the default.

4.1.1 Thought experiment: caching L1I in L1.5

Fundamentally, caching L1I cache lines within the L1.5 (inclusive) should be straightforward. However, the change is likely to be complex due to assumptions made during the development of the L1.5.

Such implementations may need to be careful of the following points (non-exhaustive list):

- Mismatched line sizes between icache/dcache (32B/16B). Perhaps it would be best to make L1.5 line size be 64B. Unfortunately, changing line size here will also necessitate changes to the L2.
- Removal/modification to L2's icache handling logics.
- Contention between icache and dcache necessitate additional logics for invalidations and evictions.

4.2 Interfacing with L1D

The L1.5 is inclusive of L1D. Communication between these two caches is done with the CCX bus interface, and is almost unchanged from the T1 design.

4.2.1 Thought experiment: replacing CCX

Replacing the CCX while still using the T1 core is likely to require colossal efforts as CCX is highly integrated with the core pipelines. If power is a concern (as the CCX bus is quite wide, at 124b for PCX and 145b for CPX), it is better to buffer and transduce locally at the core before sending to L1.5 where the interface is easier to modify.

Modifying L1.5 to use with T2 core should be easier as cache interface is mostly the same between T1 and T2. Bit positions and defines are slightly different though so `l15_cpxencoder.v` and `l15_pcxdecoder.v` (where CCX signals are translated to L1.5 internal signals) should be modified accordingly.

4.3 Design note: way-map table

The way-map table (WMT) supports two important cache coherence operations: store acknowledgments and cache line invalidations. Store-acks and invals by index and way is mandatory per T1 design. The core does not support invals by cache tag. This is why T1 also has the L2s store copies of the L1I/D tag table. Instead of duplicating the tag table, the L1.5 minimizes overheads by implementing a novel way-map table to precisely indicate index and way of the target cache line in L1D.

Some sort of table is needed for mismatched L1D/L1.5 sizes/associativities. However, such table is also needed for matching L1D/L1.5 because way allocation is decoupled between the two caches. Specifically, the L1D dictates where it will be putting the new cache line, and L1.5 cannot always allocate similarly.

4.4 Design note: write-buffer

A write-buffer–16B (cache block size), one per thread[1]–was necessitate by the use of *allocation-on-fill* instead of *allocation-on-miss* allocation policy.[2] The write-buffer captures the store, which can be any size from 1B to 8B, then merges this data when the filling cache line comes back from the L2.

In practice, the implementation is more complex and prone to bug. There are two primary reasons for this. First, stores from

[1] T1 can issue one outstanding store per thread.

[2] Changing the allocation policy will likely affect the deadlock-free coherence protocol.

thread A can alias to a cache line of another in-flight store from thread B. For correctness, the L1.5 needs to do a full-tag comparison between the two stores (in stage S1), and update (or coalesce write) the store-buffer entry accordingly. Second, in the corner-case where the in-flight store is asking for write-permission from L2 (upgrading MESI state from S to M), and somehow the cache line (in S state) is evicted due to another fill from the L2, some metadata are needed to guarantee correctness. The correctness question here is whether the store acknowledgment to the L1D should set the modify-bit or not, where this bit indicates whether the L1D contain the cache line and should be updated. If the cache line was in S state, was not evicted, and is present in L1D, then the bit must be set; otherwise the bit is not set.

4.5 Design note: handling requests from core

These notes give extra detail on how the L1.5 handle load/store requests from the T1 core.

4.5.1 Non-cachable loads/stores

These are requests either to IO space (b39 set to 1) or to memory pages marked as non-cachable or when the core's L1I/D are not turned on. In all cases, the PCX requests will have the non-cachable bit set and the L1.5 will interpret them as such.

These requests are handled as two transactions. First, L1.5 will flush/invalidate its own and L1D cache for the request's address (if found valid). Then, L1.5 forwards load/write request to L2.

4.5.2 Prefetch loads

Prefetch reqs are ignored at L1.5. Prior implementations have prefetch reqs allocated in the L1.5 but these interfere with the reconfigurable distributed cache feature so we turned it off.

4.5.3 Cachable load/store requests

These requests are stalled from entering the pipeline if any other operations in the pipeline alias to the (1) same set, (2) same MSHR, or (3) when MSHR entries cannot be allocated. Condition (2) is mostly a failsafe check in the T1.

Condition (3) occurs when one thread issue multiple loads (or stores) concurrently, which only happens when it does a block

load (or store). In contrast, normal cachable loads (or stores) can only be issued serially, adhering to TSO coherence protocol. The current Piton design does not optimize block load/store. The L1.5 treats these requests as if they're regular requests. As the L1.5 only has one load (or store) MSHR per thread, the L1.5 will stall the PCX buffer until the current request is finished.

A cachable store could potentially hit L1.5's store buffer (core is writing to the same cache line). In this case, L1.5 coalesces the writes and acknowledges the store immediately.

4.5.4 Load/store to special addresses

39:32	**31:26**	**25:24**	**23:4**	**3:0**
0xb0	0	way	index	0

Table 9: L1.5 Diag Load/Store Field Usage

39:32	**31:26**	**25:24**	**23:4**	**3:0**
0xb3	0	way	index	0

Table 10: L1.5 Data Flush Field Usage

While OpenSPARC T1 can address up to 40b of memory space, there are special ranges of memory space dedicated to specific purposes. The followings document the ranges that L1.5 recognizes using the top 8 bits of the address (bit 39 to 32):

- 0xb0: Non-coherent diagnosis load/store
- 0xb0: Non-coherent diag load/store. OS can load/store directly to L1.5 SRAMs with the format in Table 9.
- 0xb3: Data cache line flush. OS can store to the address format, as shown in Table 10 to flush a specific cache index and cache way to L2. Since L1.5 is inclusive of L1D, flushing L1.5 results in both being flushed to L2.
- 0xb2: HMC access (related to CSM)
- 0xb5: HMC flush (related to CSM)
- 0xba: Configure registers access

4.5.5 CAS/SWP/LOADSTUB

CAS/SWP/LOADSTUB are almost the same to L1.5, and are handled in 2 stages. Like for non-cachable load/stores, L1.5 first checks tag and writes back cache line if found, and also invalidates L1D if applicable. In the following stage, L1.5 sends appropriate data packets, 16B for CAS (8B compare, 8B swap), or 8B (SWP/LOADSTUB), to the L2 to resolve the atomic operation. L1.5 then waits for response from L2. The response is passed to the core and is not cached; in all cases, the cache line containing the address of the atomic operation is cleared from L1D and L1.5. Whether the line is still cached in L2 is up to L2's policy (and likely to be cached since L2 represents the global coherence point for memory).

4.5.6 L1D/L1I self-invalidations

L1I/D self invals (due to data corruption) is acknowledged, but not passed to L2.

4.6 Design note: handling requests from L2

Piton's cache coherence scheme is based on MESI, and L1.5, conforms to it through a deadlock-free NoC-based coherence scheme. The packet format and coherence request diagram are further discussed in Chapter 3.

4.6.1 Invalidations

Invalidation requests from L2 operate on 64-byte cache blocks, whereas L1.5 (as does L1D) employs 16-byte cache block size. This means L1.5 needs to invalidate multiple (4) cache indices that covers the said 64-byte block. For each 16-byte segment, L1.5 checks tag and evict found cache lines (and writes back if dirty). On the last cache line in the sequence, L1.5 sends back an inval ack to L2.

4.6.2 Downgrades

Like invals, downgrades from L2 also operate on 64-byte cache blocks. These downgrades are handled similarly to invals.

4.7 Inter-processor Interrupts

L1.5 primarily forwards inter-processor interrupts (IPI) for both directions: from own core to other core, and reverse. Originating from a core, the IPI passes through the PCX to L1.5, then NoC1 to the local L2, then bounces back to the target L1.5 through NoC2, and finally to the target core through CPX. Otherwise if originating from off-chip (initial wake-up message), the packet is sent to L2 through NoC1, and continues as above.

There are two addressing scheme with inter-proc interrupts: a compatibility scheme that requires no OS modification but only addresses up to 64 cores (and 2 threads each), or a new scheme for up to 64k cores. The two schemes can be used at the same time during runtime; they are detailed in Table 11

Scheme	63	62:34	33:26	25:18	17:15	14:9	8	7:0
Compat	0	0	0	0	type	coreid	threadid	intvector
New	1	0	ypos	xpos	type	0	threadid	intvector

Table 11: IPI vector format. "type" and "intvector" are described in more detail in the OpenSPARC T1 Microarchitectural Manual.

4.8 Interfaces

L1.5 connects to other components through these main channels: PCX/CPX, NoC1, NoC2, and NoC3. Another bus, UBC, is used for debugging with JTAG.

4.8.1 CCX Transceiver

The core sends and receives information encoded in PCX and CPX format as described in Tables *3-1*, *3-2*, *3-3*, and *3-4* in the OpenSPARC T1 Micro Specification documentation. PCX/CPX are the bus/protocol used in the T1 to connect the core to L2, and are mostly unchanged in Piton.

The core issues read and write requests through the PCX bus; there can be up to 1 read, 1 write, and 1 icache request outstanding per thread. So, in Piton configured with 2 threads, the L1.5 can receive up to 6 concurrent requests. The number of PCX buses per core is reduced from 5 to 1. In T1, the core can issue requests to 4 banks of L2, plus an IOB memory controller,

and each requires a separate PCX channel. In Piton though, the only destination is the L1.5. The PCX bus is buffered by a 2-entry queue at L1.5–the minimum requirement by the PCX bus protocol to support `CAS` requests.

The L1.5 returns data to the core through CPX interface. Like PCX, it is mostly unmodified. The same CPX bus is multiplexed between the L1.5 and the FPU module, with priority given to the L1.5. While unnecessary, the L1.5 buffers the CPX response to optimize for critical path and thus synthesis.

Three network-on-chip (NoC) channels are used by the L1.5 and L2 to communicate. General detail of the coherence protocol is portrayed in another section. Specifically to L1.5, NoC1/3 are outputs, and NoC2 is input. To break cyclic dependencies between the channels, the L1.5 equipped the NoC1 output with a 8-entry request buffer and 2-entry 8-byte data buffer–just enough to satisfy 6 inflight requests including 2 stores from the core. NoC2 is currently configured with a 512-bit buffer, enough for the largest response from L2 (icache response); and NoC3 is with a 1-entry queue, not for correctness but for performance and relax synthesis timing requirements.

4.9 Testing/debugging support

There are two ways to access the L1.5 cache by index and way instead of by address. First, the software can do so through reading and writing the ASI addresses. Second, hardware debugger can access the SRAMs directly by means of JTAG to CTAP to RTAP to BIST engine.

L1.5 also filters accesses to the config register address space and forwards them accordingly. The L1.5 itself does not have any configurable register.

4.10 Implementation

4.10.1 Pipelined implementation

L1.5's is pipelined to three stages: S1, S2, and S3. Fundamentally, S1 decodes requests from either PCX or NoC2; S2 does tag check, and issues read/write command to data SRAM; and S3 steers data and makes request to the right output buffer. Figure 11 shows a simplified diagram of the L1.5 pipeline.

S1

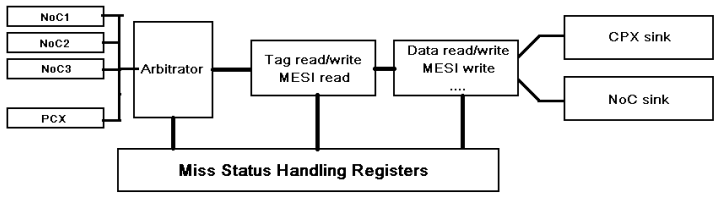

Figure 11: Pipeline diagram of the L1.5

- Pick request from either PCX (core) or NoC2 (L2) and decode to internal opcodes. Give preference to NoC2 over PCX to prevent deadlock.
- MSHR related operations: access ,allocate new entries, tag check, restore saved operations, and store/update data to the write-buffers.
- Stall conflicted requests due to index-locking, address conflict, and full MSHR.
- Initiate tag/MESI read.

S2

- Mainly tag check and initiate data read.
- Read write-buffer (for some requests).
- Misc ops include: write MESI, write tag, read WMT, read-/write config registers.

S3

- Return data to core/L2.
- Use WMT to ack stores from core.
- Update LRU/WMT/MSHR

4.11 Cache configurability

With regards to cache size and configurability, the L1.5 is completely decoupled from both L1D and L2. By default, the L1.5 is 8192KB in size and 4-way associative, but Piton users can specify different parameters in the config files. Please refer to the simulation manual for more detail.

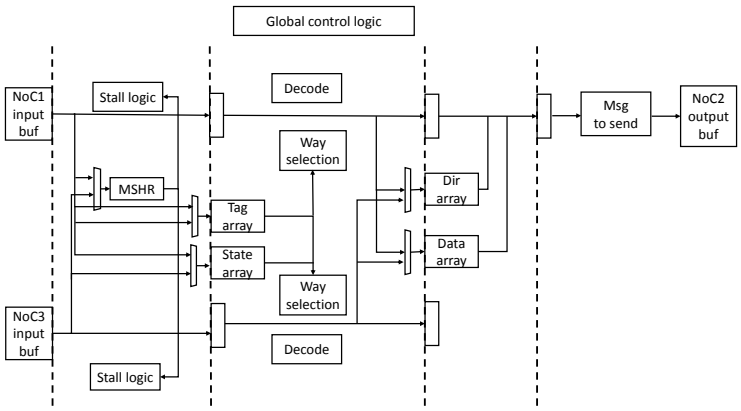

Figure 12: The architecture of the L2 cache.

5 L2 Cache

5.1 Overview

The L2 cache is a distributed write-back cache shared by all cores. The default cache size is 64KB per core. It is 4-way set associative and the block size is 64 bytes. A directory array is also integrated with the L2 cache with 64 bits per entry. Therefore, the directory is able to keep track of up to 64 sharers. The L2 cache is inclusive of private L1.5 and L1 caches so every private cache line has a copy in the L2 cache.

Each distributed L2 cache receives input requests from NoC1 and NoC3 and sends output responses to NoC2. The NoC interface is converted from credit-based to val/rdy before connecting to the L2 cache, so the val/rdy interface is used in those I/O ports of L2.

5.2 Architecture Description

As shown in Figure 12, the L2 cache is pipelined with dual pipelines in parallel. The first pipeline receives input packets from NoC1 and sends output packets to NoC2; the second pipeline receives input packets from NoC2 and do not directly send out packets. An input buffer is inserted at each input port and an output buffer is inserted at each output port. The first pipeline contains 4 stages and the second pipeline contains only 3 stages (Because it does not need to send output messages). The L2 cache contains 4 sub-arrays: the state array, the tag array, the data array and the directory array. Besides, a miss status han-

dling register (MSHR) module is maintained to store meta-data during L2 transactions. They are all shared and can be accessed by both pipelines. However, since most sub-modules only have a read/write port, they cannot be accessed by both pipelines at the same time. Therefore, if one stage in one pipeline is active, the corresponding stage in the other pipeline must be stalled.

5.2.1 Input Buffer

The purpose of the input buffer is to receive and buffer the entire message before sending to the pipeline. In our design we have two separate buffers for message header and message data. This design simplifies the logic design by separating the control path and data path. So the message header can be fetched in an early stage while the message data can be fetched in a late stage without affecting the pipelining of the next message header. For NoC1 input buffer, we choose 8 flits as the header buffer size and 4 flits as the data buffer size because most of the input messages from NoC1 do not contain data. While for NoC3 input buffer, we choose 4 flits as the header buffer size and 16 flits as the data buffer size because NoC3 input messages include the memory response which contains 8 flits of data.

5.2.2 State Array

POSITION	FIELD	DESCRIPTION
5:0	OWNER	the location of the owner
9:6	SUBLINE	a 4-bit vector to indicate whether each 16B subline has a private copy
10	DI	0 means data cache line, 1 means instruction cache line
11	D	L2 dirty bit
12	V	L2 valid bit
14:13	MESI	MESI state bits
29:15	see above	state bits for way 2
44:30	see above	state bits for way 3
59:45	see above	state bits for way 4
63:60	LRU	LRU bits for four ways
65:64	RB	round-robin bits to circle around four ways

Table 12: Field decomposition of the state array.

The L2 state array contains 256 entries and each entry contains 66 bits. Each entry stores the state information for all 4 ways in a set. There are 4 LRU bits and 2 round-robin bits to implement a pseudo-LRU replacement policy. Besides, each way is assigned

15 bits: 2 bits for MESI state, 2 bits for valid/dirty state, 1 bit to indicate a instruction/data cache line, 4 bits to indicate which sublines exist in the private caches (Because each L2 cache line is 64 bytes but each L1.5 cache line is only 16 bytes) and 6 bits for owner ID. The detail field decomposition of the state array is shown in Figure 12.

The state array has one read port and a write port. Each read-/write operation takes one cycle. It is read in pipeline stage 1 (the data will be available in the next stage) and written in stage 4 for the first pipeline and stage 3 for the second pipeline.

5.2.3 Tag Array

The L2 tag array contains 256 entries and each entry has 104 bits for 4 ways (26 bits per way). The tag array has only one read/write port and is accessed at stage 1.

5.2.4 Data Array

The L2 data array contains 4096 entries and each entry has 144 bits (128 data bits and 16 ECC bits). The data array has only one read/write port and is accessed at stage 2. Since the data width is only 16 bytes, a memory response for a cache line requires four cycles for the write to complete.

5.2.5 Directory Array

The directory array contains 1024 entries and each entry has 64 bits. Therefore, it can keep track of up to 64 sharers. The directory array has only one read/write port and is accessed at stage 2.

5.2.6 MSHR

The MSHR has 8 entries and each entry has 2 bits of state and 122 bits of meta-data shown in Table 13. The purpose of the MSHR is to keep meta-data for in-flight requests that need extra communication such as coherence invalidation or memory fetch before completion.

5.2.7 Output Buffer

The output buffer receives the entire output message at once and sends out one flit per cycle to NoC2 through a val/rdy in-

POSITION	FIELD	DESCRIPTION
39:0	ADDR	the physical address of the request
41:40	WAY	the way in the L2 set
49:42	MSHRID	the MSHR ID of the request in the original requester (L1.5)
50	CACHE_TYPE	0 means data cache line, 1 means instruction cache line
53:51	DATA_SIZE	the requested/stored data size
61:54	MSG_TYPE	message type
62	L2_MISS	whether it causes a L2 miss
76:63	SRC_CHIPID	source CHIPID
84:77	SRC_X	source X
92:85	SRC_Y	source Y
96:93	SRC_FBITS	source FBITS
119:97		reserved
120	RECYCLED	re-excute the request in the L2 pipeline
121	INV_FWD_PENDING	indicate that the L2 is currently sending out invalidations

Table 13: Field decomposition of the MSHR meta-data array.

terface. The size of the output buffer is 5 flits which matches the maximum output message size to NoC2.

5.3 Pipeline Flow

The first pipeline handles all requests from L1.5 caches through NOC1. It contains four pipeline stages as follows: Stage 1:

- All MSHR entries are cammed to check if there is a in-flight request having the same index bits as the current request. If so, the current request is stalled. Other stall conditions are also checked.
- The state array and tag array are accessed if necessary.

Stage 2:

- Use the state and tag information to determine which way to select.
- Decode the current request and generate corresponding control signals.
- Access the directory array and data array if necessary.

Stage 3:

- This stage is mainly introduced for timing purpose due to the high latency of the data array accesses.

Stage 4:

- Generate output messages.
- write meta-data into the state array and MSHR if necessary

The second pipeline handles responses from DRAM and remote L1.5 caches through NOC3. It contains three pipeline stages as follows:

Stage 1:

- The MSHR is cammed for write-back requests. For other types of requests, the specific MSHR entry is read out based on the MSHR ID field in the current request.
- The state array and tag array are accessed if necessary.

Stage 2:

- Use the state and tag information to determine which way to select if necessary.
- Read the data from input buffer if needed (May take multiple cycles for DRAM response).
- Access the directory array and data array if necessary.

Stage 3:

- write meta-data into the state array and MSHR if necessary

5.4 Special Accesses to L2

I/O addresses starting from 0xA0 to 0xAF are assigned for special accesses to L2 cache. Non-cacheable loads and stores to those addresses are translated into special accesses to L2 based on other bits of the address. All types of special accesses are listed as follows:

: 0xA0: Diagnostic access to the data array

: 0xA1: Diagnostic access to the directory array

: 0xA3: Coherence flush on a specific cache line

: 0xA4: Diagnostic access to the tag array

: 0xA6: Diagnostic access to the state array

: 0xA7: Access to the coreid register

: 0xA8: Access to the error status register

: 0xA9: Access to the L2 control register

: 0xAA: Access to the L2 access counter

: 0xAB: Access to the L2 miss counter

: 0xAC, 0xAD, 0xAE, 0xAF: Displacement line flush on a specific address

The detailed formats of different types of special accesses are explained below.

5.4.1 Diagnostic access to the data array

The L2 data array is a 4096x144 SRAM array. Each line contains 128 bits of data and 16 bits of ECC protection bits. Each half of the 128 bits is protected by 8 ECC bits. Each diagnostic load or store can either access 64 bits of data or 8 bits of ECC bits, depending on the 31:30 bit of the address. The access format of the address is described below.

: 39:32 ⇒ access type: 0xA0

: 31:30 ⇒ access op: 2'b00 means data bits, 2'b01 means ECC bits, other values are undefined

: 29:24 ⇒ home node

: 23:16 ⇒ undefined

: 15:14 ⇒ way selection

: 13:6 ⇒ index selection

: 5:3 ⇒ offset selection

: 2:0 ⇒ 3'b000

A stx or ldx instruction can be used to read or write the data array. For accesses to the data bits the entire 64 bits are stored or loaded, while for ECC bits only the lowest 8 bits (7:0) are stored or loaded.

5.4.2 Diagnostic access to the directory array

The L2 directory array is a 1024x64 SRAM array. Each line contains 64 bits of sharers to keep track of up to 64 sharers.

Each diagnostic load or store is access on the entire 64 bits. The access format of the address is described below.

: 39:32 ⇒ access type: 0xA1

: 31:30 ⇒ undefined

: 29:24 ⇒ home node

: 23:16 ⇒ undefined

: 15:14 ⇒ way selection

: 13:6 ⇒ index selection

: 5:3 ⇒ offset selection

: 2:0 ⇒ 3'b000

A stx or ldx instruction can be used to read or write the directory array.

5.4.3 Coherence flush on a specific cache line

This request flushes a L2 cache line in a selected set and way. It also sends out invalidations to flush related L15 cache lines if needed. The access format of the address is described below.

: 39:32 ⇒ access type: 0xA3

: 31:30 ⇒ undefined

: 29:24 ⇒ home node

: 23:16 ⇒ undefined

: 15:14 ⇒ way selection

: 13:6 ⇒ index selection

: 5:0 ⇒ 6'b000000

A load instructions can be used to flush a L2 line regardless of the data width (ldx, ldub ...). The loaded data has no meaning and will not be checked.

5.4.4 Diagnostic access to the tag array

The L2 tag array is a 256x104 SRAM array. Each line contains 4 tags, each of which is 26 bits. Each diagnostic load or store is

access on one tag. The access format of the address is described below.

: 39:32 ⇒ access type: 0xA4

: 31:30 ⇒ undefined

: 29:24 ⇒ home node

: 23:16 ⇒ undefined

: 15:14 ⇒ way selection

: 13:6 ⇒ index selection

: 5:0 ⇒ 6'b000000

A stx or ldx instruction can be used to read or write the directory array. Only the lowest 26 bits (25:0) are stored or loaded.

5.4.5 Diagnostic access to the state array

The L2 state array is a 256x66 SRAM array. Each entry contains state bits for 4 ways, each of which is 15 bits, as well as 6 LRU bits shared by or 4 ways. Each diagnostic load or store can either access 60 bits of state data or 6 bits of LRU bits, depending on the 31:30 bit of the address. The access format of the address is described below.

: 39:32 ⇒ access type: 0xA6

: 31:30 ⇒ access op: 2'b00 means state bits, 2'b01 means LRU bits, other values are undefined

: 29:24 ⇒ home node

: 23:14 ⇒ undefined

: 13:6 ⇒ index selection

: 5:0 ⇒ 6'b000000

A stx or ldx instruction can be used to read or write the data array. For accesses to the state bits the lowest 60 bits (59:0) are stored or loaded, while for ecc bits only the lowest 6 bits (5:0) are stored or loaded.

5.4.6 Access to the coreid register

The coreid register is a 64-bit register in L2 cache. The lowest 34 bits store the node id of the L2 (chipid, x, y and fbits). The higher 30 bits store the maximum number of cores in the system aggregated across multiple chips (in the format of max chipid, max y, max x). The access format of the address is described below.

: 39:32 ⇒ access type: 0xA7

: 31:30 ⇒ undefined

: 29:24 ⇒ home node

: 23:0 ⇒ undefined

A stx or ldx instruction can be used to read or write the data array. Each diagnostic load or store operates on the entire 64 bits.

5.4.7 Access to the error status register

The error status register is a 64-bit register in L2 cache. The lowest bit indicates the last error is correctable or not. The second bit indicates whether the last error is uncorrectable. The 3rd bit indicates whether there are multiple errors. Bit 14 to 4 stores the line address of the error data array entry. Bit 54 to 15 stores the physical address of the request that causes the error. The access format of the address is described below.

: 39:32 ⇒ access type: 0xA8

: 31:30 ⇒ undefined

: 29:24 ⇒ home node

: 23:0 ⇒ undefined

A stx or ldx instruction can be used to read or write the data array. Each diagnostic load or store operates on the entire 64 bits.

5.4.8 Access to the L2 control register

The L2 control register is a 64-bit register in L2 cache. The lowest bit is used as the enable bit for clumpy shared memory. The 2nd bit is the enable bit for the error status register. The

3rd bit is the enable bit for l2 access counter and the 4th Bit is the enable bit for l2 miss counter. 32 to 53 are used as the base address for the sharer map table (SMT). Other bits are undefined. The access format of the address is described below.

: 39:32 ⇒ access type: 0xA9

: 31:30 ⇒ undefined

: 29:24 ⇒ home node

: 23:0 ⇒ undefined

A stx or ldx instruction can be used to read or write the data array. Each diagnostic load or store operates on the entire 64 bits.

5.4.9 Access to the L2 access counter

The L2 access counter stores the total number of L2 accesses. It can be reset by writing all zeros into it. The access format of the address is described below.

: 39:32 ⇒ access type: 0xAA

: 31:30 ⇒ undefined

: 29:24 ⇒ home node

: 23:0 ⇒ undefined

A stx or ldx instruction can be used to read or write the data array. Each diagnostic load or store operates on the entire 64 bits.

5.4.10 Access to the L2 miss counter

The L2 access counter stores the total number of L2 misses. It can be reset by writing all zeros into it. The access format of the address is described below.

: 39:32 ⇒ access type: 0xAB

: 31:30 ⇒ undefined

: 29:24 ⇒ home node

: 23:0 ⇒ undefined

A stx or ldx instruction can be used to read or write the data array. Each diagnostic load or store operates on the entire 64 bits.

5.4.11 Displacement line flush on a specific address

This request flushes a specific address in the L2 cache. It also sends out invalidations to flush related L15 cache lines if needed. In order to convey both the tag and the index information in the address, actually only the highest 6 bits of the address are used as the access type (this is why the highest 8 bits can be either 0xAC, 0xAD, 0xAE or 0xAF), and the highest 6 bits of the tag are stored in bit 5:0 instead. L2 will be rearrange the address to be the correct format. The access format of the address is described below.

: 39:34 ⇒ access type: 6'b101011

: 33:14 ⇒ lower part of the tag

: 13:6 ⇒ index selection

: 5:0 ⇒ higher part of the tag

A ldub instructions can be used to flush a L2 line. It cannot be replaced by ldx because the offset bits are actually part of the tag so they can be arbitrary value and violate the address alignment requirement for data width larger then one byte. The loaded data has no meaning and will not be checked.

6 On-chip Network

OpenPiton is a Tile Processor consisting of a 2D grid of homogeneous and general-purpose compute elements. Three dynamic switched networks provide massive on-chip communication bandwidth. These three networks communicate the tiles for cache coherence, I/O and memory traffic, and inter-core interrupts.

These three NoCs are physical networks without the support of virtualization. Each NoC consists of two 64-bit uni-directional links, one for each direction. The NoC relies on a credit-based flow control. Packets are routed by XY dimention-ordered wormhole routing to avoid deadlocks. Within each NoC, there is a fully connected crossbar, which allows all-to-all five-way communication.

These three NoCs are dynamic, providing a packetizd, fire-and-forgot interface. Each packet contains a header word denoting the x and y destination location for the packet along with the packet's length. The dynamic networks are dimention-ordered wormhole-routed. Each hop takes one cycle when packets are going straight and one extra cycle for route calculation when a packet must make a turn at a switch. There are four-entry FIFO buffers that serve only to cover the link-level flow-control cost.

6.1 Dynamic Node Top

The top-level module of dynamic node is *dynamic_node_top.v*. It implements a full crossbar. The two main components are *dynamic_input_top_X* and a *dynamic_output_top*. We have two implementations of different X values for *dynamic_input_top*, where X indicates the size of NIBs inside.

6.2 Dynamic Input Control

Table 14 shows the packet fields. An input control unit (*dynamic_input_control*. maintains the control of where different signicals want to go and counters of how many words are left in messages. An route request calculate unit (*dynamic_input_route_request_calc.v*) generates all of the *route_request* lines and the $default\_ready$ lines from the absolute location of the tile and the absolute address of the destination tile.

POSITION	FIELD	DESCRIPTION
$DATA\_WIDTH - CHIP\_ID\_WIDTH - 1 : DATA\_WIDTH - CHIP\_ID\_WIDTH - XY\_WIDTH$	abs_x	absolute x position
$DATA\_WIDTH - CHIP\_ID\_WIDTH - XY\_WIDTH - 1 : DATA\_WIDTH - CHIP\_ID\_WIDTH - 2 * XY\_WIDTH$	abs_y	absolute y position
$DATA\_WIDTH - 1 : DATA\_WIDTH - CHIP\_ID\_WIDTH$	abs_chip_id	absolute chip id
$DATA\_WIDTH - CHIP\_ID\_WIDTH - 2 * XY\_WIDTH - 2 : DATA\_WIDTH - CHIP\_ID\_WIDTH - 2 * XY\_WIDTH - 4$	$final\_bits$	the direction of the final route
$DATA\_WIDTH - CHIP\_ID\_WIDTH - 2 * XY\_WIDTH - 5 : DATA\_WIDTH - CHIP\_ID\_WIDTH - 2 * XY\_WIDTH - 4 - PAYLOAD\_LEN$	length	offload length

Table 14: Field decomposition of a packet.

6.3 Dynamic Output Control

A dynamic output control unit (*dynamic_output_control.v*) maintains the control of the output mux for a dynamic network port. It takes routing decisions from all input ports and output the control for the respective crossbar mux and the validOut signal for a respective direction. If multiple input ports request for the same output port, a round-robin selection is used to choose an input port that is least recently used.

6.4 Buffer Management

A buffer management unit (*space_avail_top.v*) keeps track of how many spots are free in the NIB that the packet is sent to. There are two important inputs in this module, valid and yummy. "Valid" indicates one packet is sent out to the output port while "yummy" indicates a packet is consumed by the input port of next hop. Combining a counter, the unit keeps the number of empty buffers in input port of next hop.

References

[1] Sun Microsystems, Santa Clara, CA, *OpenSPARC T1 Microarchitecture Specification*, 2008.

[2] Y. Fu, T. M. Nguyen, and D. Wentzlaff, "Coherence domain restriction on large scale systems," in *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, (New York, NY, USA), pp. 686–698, ACM, 2015.

[3] Oracle, "OpenSPARC T1." http://www.oracle.com/technetwork/systems/opensparc/opensparc-t1-page-1444609.html.

[4] *OpenSPARC T1 Microarchitecture Specification*. Santa Clara, CA, 2006.