Getting to Work with OpenPiton

Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, Matthew Matl, David Wentzlaff

Princeton University

http://openpiton.org
Extension Using NoCs
NoC: packet format

CHIPID: The highest bit indicates whether the destination is on-chip or off-chip, the rest bits indicates the chip ID
XPOS: The position of the destination tile in the X dimension
YPOS: The position of the destination tile in the Y dimension
FBITS: The router output port to the destination
PAYLOAD LENGTH: The number of payload packets
RESERVED: Reserved Bits used by higher-level protocols.
NoC: .h files

piton/design/include/network_define.h
Defines the header flits b63-22
(all except messageid, tag, and options 1)

piton/design/include/define.vh
defines the rest

```
181 //Memory requests from L2 to DRAM
182 `define MSG_TYPE_LOAD_MEM 8'd19
183 `define MSG_TYPE_STORE_MEM 8'd20

196 //Memory acks from memory to L2
197 `define MSG_TYPE_LOAD_MEM_ACK 8'd24
198 `define MSG_TYPE_STORE_MEM_ACK 8'd25
199 `define MSG_TYPE_NC_LOAD_MEM_ACK 8'd26
200 `define MSG_TYPE_NC_STORE_MEM_ACK 8'd27
```
Cache Coherence Protocol

Directory-based MESI coherence Protocol
- Four-hop message communication (no direct communication between private L1.5 caches)
- Uses 3 physical NoCs with point-to-point ordering to avoid deadlock
- The directory and L2 are co-located but state information are maintained separately
- Silent eviction in E and S states
- No need for acknowledgement upon write-back of dirty lines from L1.5 to L2, but writeback guard needed in some cases.
Memory Hierarchy Datapath

Private L1.5 <-> Distributed shared L2 <-> Off-chip Chipset

NoC1 <-> NoC1
NoC2 <-> NoC2
NoC3 <-> NoC3
NoC Messages

In order to avoid deadlock, NoC3 messages will never be blocked.
Coherence Transaction Example

1. Load
2. Mem Req
3. Mem Reply
4. Data Ack

Memory

Core 1

Core 2

Ld
Coherence Transaction Example (2)
Coherence Transaction Example (3)

1. **Writeback** (WbGuard)
2. **Writeback** (Writeback)

Core 1: L1.5 → L2 → Memory

Core 2: M → L1.5 → L2

Core 1: Ld
Core 2: St

Wb
Backup Slides
Adding to OpenPiton

- AXI-Lite
- Wishbone
- Interfacing with the Network on Chip
Hooking up an AXI-Lite device
Interfacing with the Networks-on-Chip

1. Packet format
   - Highlighting key packet fields

2. Definition files
   - .h files

3. Instantiations in Verilog design
NoC: packet format

64-bit flits
1 packet header (64b) + X packet payload flits
(64b * X)
Ex: Cache request from L1.5 to L2
    Header flit + req. address flit + metadata flit
Ex: Cache response from L2 to L1.5
    Header flit + 2x data flits (16B cache line)
Ex: Instruction cache response
    Header flit + 4x data flits (32B cache line)
NoC: instantiations

piton/design/chip/rtl/chip.v.pyv

Chip-wide connections between tiles
Auto generated using PYHP
NoC: instantiations

piton/design/chip/tile/rtl/tile.v.pyv
Instantiation of NoC1/2/3

piton/design/chip/tile/rtl/tile.v.pyv
Selectable between router and crossbar design

```verilog
if (NETWORK_CONFIG == "xbar_config"):
    s = '"
    assign dyn0_do = buffer_router_data_noc1;
    assign dyn0_do_valid = buffer_router_valid_noc1;
    assign dyn0_yummyOut = buffer_router_yummy_noc1;
    assign router_buffer_data_noc1 = dyn0_dataIn;
    assign router_buffer_data_val_noc1 = dyn0_validIn;
    assign router_buffer_consumed_noc1 = dyn0_do_yummy;

    assign dyn1_do = buffer_router_data_noc2;
    assign dyn1_do_valid = buffer_router_valid_noc2;
    assign dyn1_yummyOut = buffer_router_yummy_noc2;
    assign router_buffer_data_noc2 = dyn1_dataIn;
    assign router_buffer_data_val_noc2 = dyn1_validIn;
    assign router_buffer_consumed_noc2 = dyn1_do_yummy;

    assign dyn2_do = buffer_router_data_noc3;
    assign dyn2_do_valid = buffer_router_valid_noc3;
    assign dyn2_yummyOut = buffer_router_yummy_noc3;
    assign router_buffer_data_noc3 = dyn2_dataIn;
    assign router_buffer_data_val_noc3 = dyn2_validIn;
    assign router_buffer_consumed_noc3 = dyn2_do_yummy;

    print s
    %>
```
Cache Coherence Protocol

Directory-based MESI coherence Protocol
- Four-hop message communication (no direct communication between private L1.5 caches)
- Uses 3 physical NoCs with point-to-point ordering to avoid deadlock
Directory-based MESI coherence Protocol
- The directory and L2 are co-located but state information are maintained separately

<table>
<thead>
<tr>
<th>L2 State</th>
<th>Dir State</th>
<th>Tag</th>
<th>Data</th>
<th>Sharer List</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Cache Coherence Protocol (3)

Directory-based MESI coherence Protocol
- Silent eviction in E and S states
- No need for acknowledgement upon write-back of dirty lines from L1.5 to L2
Example: Add an on-chip accelerator

1. Implement the NoC interface for the accelerator
2. Design and implement the control flow for the accelerator
   - Use interrupt packets to init and stop the accelerator
   - Use special load and stores to configure the accelerator
   - Follow the coherence protocol if a coherence cache is maintained
3. Connect the accelerator to NoCs and assign it a new tile ID
4. Modify the OS code to init the accelerator if needed
5. Write tests to test the accelerator