OpenPiton in Action

Princeton University

http://openpiton.org
Extension Using NoCs
P-Mesh NoC Connected I/O and Accelerators
**P-Mesh NoC: packet format**

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIPID</td>
<td>0-62</td>
</tr>
<tr>
<td>XPOS</td>
<td>50-49</td>
</tr>
<tr>
<td>YPOS</td>
<td>42-41</td>
</tr>
<tr>
<td>FBITS</td>
<td>34-33</td>
</tr>
<tr>
<td>PAYLOAD LENGTH</td>
<td>30-29</td>
</tr>
<tr>
<td>RESERVED</td>
<td>22-21</td>
</tr>
</tbody>
</table>

PAYLOAD PACKET 1

PAYLOAD PACKET N

**CHIPID**: Highest bits indicate whether the destination is on-chip or off-chip, the rest of the 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.
P-Mesh 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

```cpp
181 // Memory requests from L2 to DRAM
182 `define MSG_TYPE_LOAD_MEM 8'd19
183 `define MSG_TYPE_STORE_MEM 8'd20
184
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
```

```cpp
144 // Requests from L15 to L2
145 // Should always make #0 an error
146 `define MSG_TYPE_RESERVED 8'd0
147 `define MSG_TYPE_LOAD_REQ 8'd31
148 `define MSG_TYPE_PREFETCH_REQ 8'd1
149 `define MSG_TYPE_STORE_REQ 8'd2
150 `define MSG_TYPE_BLK_STORE_REQ 8'd3
151 `define MSG_TYPE_BLKINIT_STORE_REQ 8'd4
152 `define MSG_TYPE_CAP_REQ 8'd5
153 `define MSG_TYPE_CAP_P1_REQ 8'd6
154 // condition satisfied
155 `define MSG_TYPE_CAP_P2Y_REQ 8'd7
156 // condition not satisfied
157 `define MSG_TYPE_CAP_P2N_REQ 8'd8
158 // Both SWAP and LDSTUB are the same for L2
159 `define MSG_TYPE_SWAP_REQ 8'd9
160 `define MSG_TYPE_SWAP_P1_REQ 8'd10
161 `define MSG_TYPE_SWAP_P2_REQ 8'd11
162 `define MSG_TYPE_WB_REQ 8'd12
163 `define MSG_TYPE_WBGUARD_REQ 8'd13
164 `define MSG_TYPE_NC_LOAD_REQ 8'd14
165 `define MSG_TYPE_NC_STORE_REQ 8'd15
166 `define MSG_TYPE_INTERRUPT_FWD 8'd32
```
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, NoC2, NoC3

NoC1, NoC2, NoC3

NoC1, NoC2, NoC3
NoC Messages

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

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

Core 1

Memory

Core 2

Ld
Coherence Transaction Example (2)

1. Core 1:
   - E → I
   - L1.5
   - Downgrade
   - DG Ack
   - E → M
   - L2

2. Core 2:
   - I → M
   - L1.5
   - Data Ack
   - Ack
   - Data Ack
   - Store

3. Memory:

4. Diagram:
   - Core 1: Ld
   - Core 2: St
Coherence Transaction Example (3)
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

```python
258 // generate the tiles and connect them through a template
259 <<

# generate wires
260 if (NETWORK_CONFIG == "xbar_config"):
261     for i in range(X_TILES + 1):
262         for k in [1,2,3]:
263             print "wire [DATA_WIDTH-1:0] xbar_%d_out_noc%d_data;" % (i, k)
264             print "wire xbar_%d_out_noc%d_valid;" % (i, k)
265             print "wire xbar_%d_out_noc%d_yummy;" % (i, k)
266     for i in range(X_TILES):
267         for j in range(Y_TILES):
268             for k in [1,2,3]:
269                 print "wire [DATA_WIDTH-1:0] tile_%d_%d_out_noc%d_data;" % (j,i,k)
270                 print "wire tile_%d_%d_out_noc%d_valid;" % (j,i,k)
271                 print "wire tile_%d_%d_out_noc%d_yummy;" % (j,i,k)
272 # make offchip signals
273     for k in [1,2,3]:
274         print "wire [DATA_WIDTH-1:0] offchip_out_noc%d_data;" % (k)
275         print "wire offchip_out_noc%d_valid;" % (k)
276         print "wire offchip_out_noc%d_yummy;" % (k)
```
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
Cache Coercence 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
Cache Coherence Protocol (2)

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 config 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