Constructive Computer Architecture

Tutorial 9: Final Project: Part 1

Overview and Advice

Andy Wright
6.175 TA

Final Project: Part 1

- Make four modules:
  - mkMessageFIFO
  - mkMessageRouter
  - mkParentProtocolProcessor
  - mkNBCache
- To pass five sets of tests:
  - message-fifo-test
  - message-router-test
  - ppp-test
  - nb-cache-mini-test
  - nb-cache-test

MSI Overview

- Three states for each cache line:
  - **I**nvalid
  - **S**hared
  - **M**odified
- CacheTypes.bsv has an MSI enumeration
  - Also has instance of Ord typeclass so `y > I` is a valid expression

Coherency Messages

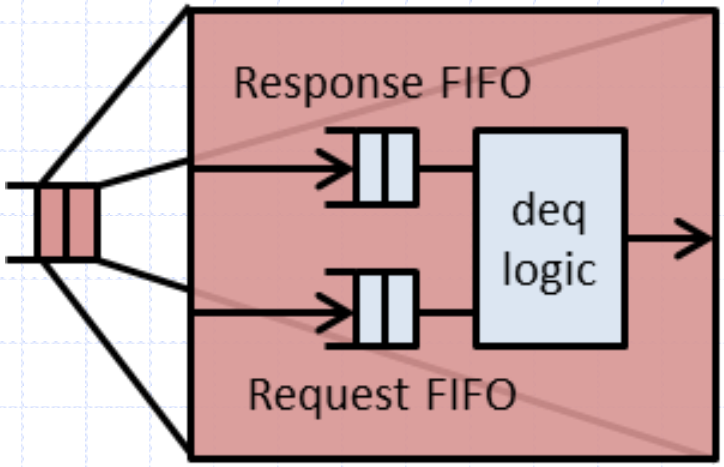
- Each message is either a request or a response
  - Responses can have data, requests cannot
  - Cache to Parent messages:
    - upgrades are requests
    - downgrades are responses
  - Parent to Cache messages:
    - downgrades are requests
    - upgrades are responses

Coherency Message Types

- CacheMemResp: (struct)
  - CacheID child — child sending or receiving request
  - Addr addr
  - MSI state — new (or next) state
  - CacheLine data — Data for I -> S,M or M -> S,I transitions
- CacheMemReq: (struct)
  - CacheID child — child sending or receiving request
  - Addr addr
  - MSI state — new (or next) state
- CacheMemMessage: (tagged union)
  - CacheMemResp Resp
  - CacheMemReq Req

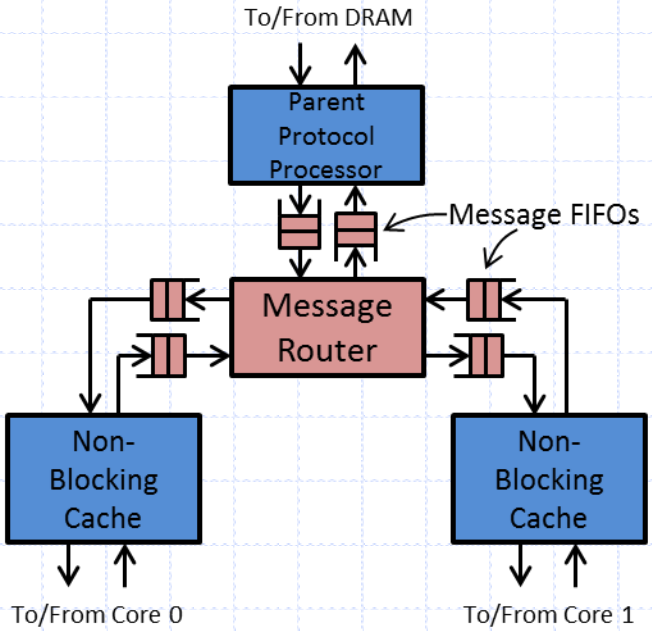
Message FIFO

- Combination of two FIFOs: request FIFO and response FIFO
- The Dequeue logic should prefer dequeuing from the response FIFO over the request FIFO



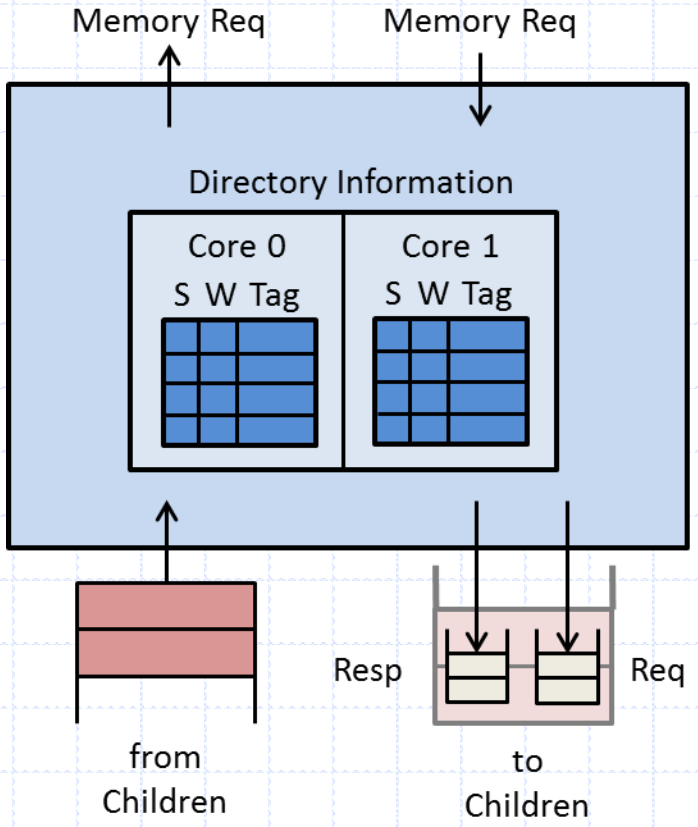
Message Router

- Sends messages from caches to the parent protocol processor, and from the parent to the correct cache
  - It must not allow requests to block responses from passing through the router



Parent Protocol Processor

- Handles cache coherency for children caches and communicates with main memory
  - Contains an MIS state, a waitc state, and the current tag for each cache line in each child cache



Parent Protocol Processor

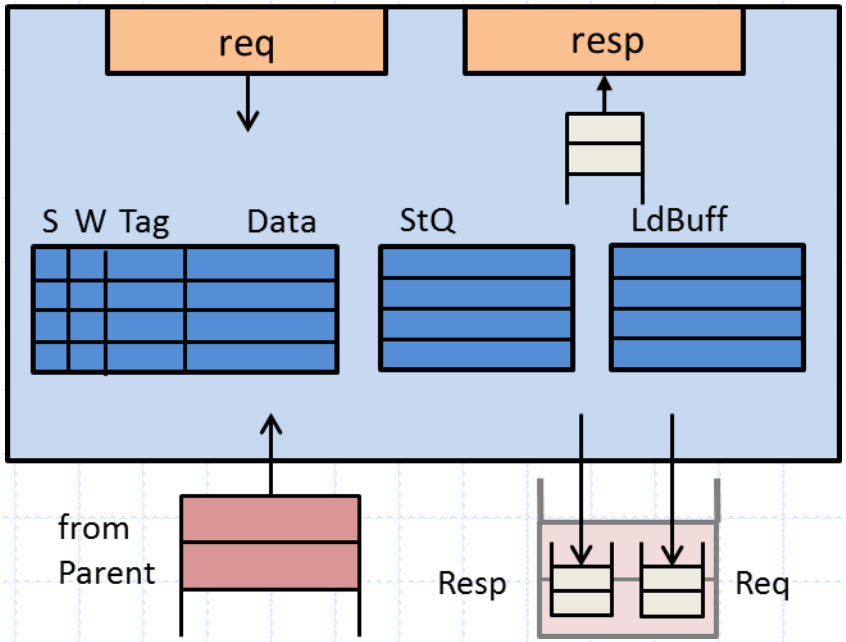
- Performs 3 of the 8 rules for our MSI coherency protocol
  - Rule 2 – Cache n is requesting an upgrade that is compatible with other caches, so send an upgrade response
  - Rule 4 – Send a downgrade request to a cache
  - Rule 6 – Receive a downgrade response from a cache

Parent Protocol Processor

- Rules 2 and 6 deals with responses that may have data
  - When sending an upgrade response from I to S or M, the parent protocol processor first needs to read the main memory for the data to send
  - When receiving a downgrade response from M to S or I, the parent protocol processor needs to write the cache line back to main memory

Non-Blocking Cache

- Handles requests and responses coming from two directions
  - Load and store requests come from the processor
  - Upgrade responses from the parent protocol processor bring in new cache lines
  - Downgrade requests from the parent protocol processor downgrade or evict cache lines
- Uses store queue and load buffer to keep track of requests in flight



Non-Blocking Cache
Handling load and store requests

- if load request:
  - if hit in store queue -> return hit
  - if hit in data cache -> return hit
  - otherwise:
    - insert into load buffer
    - send upgrade request if possible
- if store request:
  - if hit in data cache and store queue empty -> update data cache
  - otherwise:
    - insert into store queue
    - send upgrade request if possible

*Send upgrade request if possible implies rule 8 if necessary

Non-Blocking Cache
FSM for handling responses from memory

- Ready state (if memory to cache message FIFO has response)
  - update cache according to response
  - save response to register
  - go to load hit stage
- load hit state
  - if hit in load buffer -> remove entry and return hit
  - if no hit in load buffer -> go to store hit state
- store hit state
  - if hit at head of store queue -> dequeue hit and update data cache
  - if no hit at head of store queue -> go to store req state
- store req state
  - send upgrade request for head of store queue if possible
  - go to load req state
- load req state
  - if conflicting address in load buffer -> send upgrade request if possible
  - go to ready state

*Send upgrade request if possible implies rule 8 if necessary

Non-Blocking Cache
Handling requests from memory

- receiving downgrade to y request from parent
- if cache line's tag matches incoming address and the cache line's state > y:
  - Do rule 5:
    - Update cache
    - send downgrade response
- if cache line's tag doesn't match incoming address or the cache line's state <= y:
  - Do rule 7:
    - Ignore downgrade request

Module Tests

- Modules are tested with testbenches that emulate the use of each module in a larger system
- Testbenches are written using the StmtFSM library

StmtFSM example

```
1:  Stmt test = (seq
2:                  fifo.enq(1);
3:                  fifo.enq(2);
4:                  action
5:                    fifo.enq(3);
6:                    fifo.deq;
7:                  endaction
8:                  action
9:                    $display("Done");
10:                   $finish;
11:                 endaction
12:               endseq);
13: mkAutoFSM(test);
```

Each action in this seq is performed one at a time

Performed together

Creates the requested FSM. Starts automatically and quits simulation when it finishes.

StmtFSM – Generated Rules

```
Reg#(State) i <- mkReg(0);
rule L2C16(i == 0);
    fifo.enq(1);
    i <= 1;
endrule
rule L3C16(i == 1);
    fifo.enq(2);
    i <= 2;
endrule
rule L4C16(i == 2);
    fifo.enq(3);
    fifo.deq(4);
    i <= 3;
endrule
rule L8C16(i == 3);
    $display("Done");
    $finish;
endrule
```

Generated rules have "l<line_num>c<column_num>" in their name

If an action in an StmtFSM has a guard that is false, the FSM will stall until the guard is true. There is an additional FSM in the provided tests to cause a failure if the FSM has been stalled for too long.

Provided Testbenches

- *Lets look at some of the actual testbenches*

Advice

- Never use "?"
  - You can pass a poorly written test benches by just returning "?". These modules won't pass any tests in hardware
  - Instead you can use "unpack(0)" to initialize something with all 0's if its exact value doesn't matter
- Work together
  - It is often more efficient to write the same module together than to work separately

Advice – Non-Blocking Cache

- Combine rules 1 and 8 into one cycle
  - Rule 8 produces a response and rule 1 produces a request, so they can happen in the same cycle.
- Make two functions for "send upgrade request if possible"
  - One to check if it is possible
  - One to send the request (and response if necessary)
- Use vectors of registers, not RegFile
- Don't copy the code from lecture
  - Instead follow the pseudo code here and in the handout

Advice

- Start early
- Understand what you are doing
- Take a structured approach to debugging
  - Record what steps you have taken when debugging
- Ask questions

Questions?