A5. Mini Cache

CSE261: Computer Architecture, Fall 2021

Hyungon Moon

Out: Nov 30, 2021

Due: Dec 9 2021 11:59pm

Goal

- Implement a simple FSM-based cache.
- Pass the five provided tests.

Vagrant and VirtualBox

- Vagrant allows you to easily create and access a virtual machine.
- Install Vagrant from

```
https://www.vagrantup.com/
```

- Vagrant (in this class) need VirtualBox. Install from

```
https://www.virtualbox.org/
```

- Should work on Ubuntu, Mac or Windows.
- May not work inside another virtual machine.

Initializing VM (Mac, Ubuntu (Linux))

- Use terminal to enter the `hw1` directory
- Make sure that you are seeing `Vagrantfile` in the directory
- Create and boot VM

```
vagrant up
```

- Connect to the VM

```
vagrant ssh
```

- From V2 of the assignment, the `hw1` directory is automatically synced with the `hw` directory

Initializing VM (Windows)

- Use terminal to enter the `hw1` directory
- Make sure that you are seeing `Vagrantfile` in the directory
- Create and boot VM

```
vagrant.exe up
```

- Connect to the VM

```
vagrant.exe ssh
```

- From V2 of the assignment, the `hw1` directory is automatically synced with the `hw` directory

SBT

- We will use a complex set of tools enabling us to write in Chisel.
- We don't need to care about them: SBT does the dirty job.
  - ➢Located in `hw/sbt` or `sbt`.
- You will use three commands, referring to the lecture slides.
  - ➢To run the Main object (and test the core in this assignment)

```
sbt/bin/sbt run
```

SBT trouble shooting

- When running sbt for the first time, you may see this message

```
java.io.IOException: org.scalasbt.ipcsocket.NativeErrorException:
[1] Operation not permitted
Create a new server? y/n (default y)
```

- Just press enter to go for the default value.

FSM to implement

- Design a cache that has five states

```
// State machine
 val s_idle :: s_compare_tag :: s_allocate :: s_wait :: s_write_back :: Nil = Enum(5)
 val state = RegInit(s_idle)
```

  - `s_idle`: wait for the request, move to `s_compare_tag` on a valid request.
  - `s_compare_tag`: use the request and loaded tag to check if hit.
    - on hit, move to `s_idle` and generate response
    - on, move to `s_write_back` if valid. Otherwise, move to `s_allocate`.
  - `s_write_back`: write the data back to memory and move to `s_allocate`.
  - `s_allocate`: request to read the data from memory and move to `s_wait`.
  - `s_wait`: if mem_resp becomes available, write to cache and respond.

Skeleton: UniCache

- You are to fill out the UniCache module.

```
class UniCache() extends Module  {
  val io = IO(new Bundle {
    val req = Input(new UniCacheReq())                 // request arrives
    val req_ready = Output(Bool())                     // should be true.B to receive request.
    val resp = Output(new UniCacheResp())              // interface for response
    val mem_req = Output(new MemReq())                 // memory read/write request
    val mem_resp = Input(new MemResp())                // memory response

    val debug_clear = Input(UInt(4.W))             // for testing.
    val debug_valid = Input(Bool())                // for testing.
  })
}
```

```
class UniCacheReq() extends Bundle {
  val addr = UInt(64.W)
  val wdata = UInt(64.W)
  val write = Bool()
  val valid = Bool()
}
```

```
class UniCacheResp() extends Bundle {
  val rdata = UInt(64.W)
  val valid = Bool()
}
```

Skeleton: UniCacheBase

- `UniCacheBase` wraps `UniCache` and tests it.
- Use `MemReq` to send the read/write requests.

```
class MemReq extends Bundle {
  val raddr = UInt(64.W)
  val waddr = UInt(64.W)
  val wdata = UInt(256.W)
  val write = Bool()
  val read = Bool()
}
```

```
class MemResp extends Bundle {
  val rdata = UInt(256.W)
  val valid = Bool()
}
```

- `valid` field in `MemResp` becomes `true.B` when the response becomes available.

Skeleton: UniCache (2)

- You will need to define and use registers including:

```
// Registers to keep data

 val req_reg = Reg(new UniCacheReq())
 val tag_reg = Reg(UInt(58.W))
 val data_reg = Reg(UInt(256.W))
 val valid_reg = Reg(Bool())
```

  - To use the request received on `s_init` on later states.

Skeleton: UniCache (3)

- How to use and access the cache (tag, valids, data)?
- The skeleton defines them as separate memories as follows.

```
val tagArray = SyncReadMem(4, UInt(58.W))
val validArray = SyncReadMem(4, Bool())
val dataArray = SyncReadMem(4, UInt(256.W))
```

- The read interface is defined as follows with predefined wires.

```
fromTagArray := tagArray.read(rindex)
fromValidArray := validArray.read(rindex)
fromDataArray := dataArray.read(rindex)
```

- The read data becomes available on the next cycle.

Skeleton: UniCache (4)

- This defines the write interface

```
when(io.debug_valid) {
    tagArray.write(io.debug_clear, 0.U)
    dataArray.write(io.debug_clear, 0.U)
    validArray.write(io.debug_clear, false.B)
  }.elsewhen(cache_write) {
    tagArray.write(windex, wtag)
    dataArray.write(windex, wdata)
    validArray.write(windex, wvalid)
  }
```

- The write happens when `cache_write` is `true.B`, using `windex` as the address and `wtag`, `wdata` and `wvalid` as data.

Skeleton: UniCache (5)

- The wires you will define and use for cache access are pre-declared.

```
val rindex = Wire(UInt(2.W))
 val fromTagArray = Wire(UInt(58.W))
 val fromValidArray = Wire(Bool())
 val fromDataArray = Wire(UInt(256.W))

 val cache_write = Wire(Bool())
 val windex = Wire(UInt(2.W))
 val wtag = Wire(UInt(58.W))
 val wvalid = Wire(Bool())
 val wdata = Wire(UInt(256.W))
```

Testing

- Run to test all cases

```
sbt/bin/sbt run
```

```
selected tests: compulsoryReadMissTest, compulsoryWriteMissTest, writebackTest,
readConfilctMissTest, logTest
Test compulsoryReadMissTest passed
Test compulsoryWriteMissTest passed
Test writebackTest passed
Test readConfilctMissTest passed
Test logTest passed
```

- You can choose to test only one with, e.g.,

```
sbt/bin/sbt "run writebackTest"
```

```
selected tests: writebackTest
Test writebackTest passed
```

Investigating logs

- The tests compare the results with the expected logs.
- Expected logs are available in `src/main/scala/Tests.scala`
- The generate logs are saved as <test name>.log file, e.g., `writebackTest.log`

Task 1: compulsoryReadMissTest

- The request sequence looks like this

```
Request("b1", "b0", "xC00", "x100") // valid = 1, write = 0, addr = 0xC00, wdata = 0x100
```

- The expected log

```
cache state changes:
initially:
 Tag  | Valid |     Data (Slot3) |     Data (Slot2) |     Data (Slot1) |     Data (Slot0)
    0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
    0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
    0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
    0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
after valid: 1, write: 0, addr: c00, wdata: 100: (resp: )
 Tag  | Valid |     Data (Slot3) |     Data (Slot2) |     Data (Slot1) |     Data (Slot0)
   30 | [ 1 ] | 000000000FFF00C0 | 000000000FFF00C0 | 000000000FFF00C0 | 000000000FFF00C0
    0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
    0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
    0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
--------------------------------------------------------------------------------
```

Task 2: compulsoryWriteMissTest

- The request sequence looks like this

```
Request("b1", "b1", "xC00", "x100")
```

- The expected log

```
cache state changes:
initially:
 Tag   | Valid |     Data (Slot3) |     Data (Slot2) |     Data (Slot1) |     Data (Slot0)
     0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
     0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
     0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
     0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
after valid: 1, write: 1, addr: c00, wdata: 100:
 Tag   | Valid |     Data (Slot3) |     Data (Slot2) |     Data (Slot1) |     Data (Slot0)
    30 | [ 1 ] | 000000000FFF00C0 | 000000000FFF00C0 | 000000000FFF00C0 | 0000000000000100
     0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
     0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
     0 | [ 0 ] | 0000000000000000 | 0000000000000000 | 0000000000000000 | 0000000000000000
--------------------------------------------------------------------------------
```

Task 3: writebackTest

- The request sequence looks like this

```
    Request("b1", "b1", "xC00", "x100"),
    Request("b1", "b1", "xC04", "x101"),
    Request("b1", "b1", "xC08", "x102"),
    Request("b1", "b1", "xC0c", "x103"),
    Request("b1", "b0", "xD00", "x103"),
    Request("b1", "b0", "xC00", "x103")
```

- Please refer to `src/main/scala/Tests.scala` for the expected results

Task 4: readConfilctMissTest

- The request sequence looks like this

```
    Request("b1", "b1", "xC00", "x100"),
    Request("b1", "b1", "xD00", "x200"),
    Request("b1", "b0", "xC00", "x100"),
    Request("b1", "b0", "xD00", "x100"),
```

- Please refer to `src/main/scala/Tests.scala` for the expected results

Task 5: logTest

- The request sequence looks like this

```
    Request("b1", "b1", "xC04", "x100"),
    Request("b1", "b1", "xC18", "x101"),
    Request("b1", "b1", "xC20", "x101"),
    Request("b1", "b1", "xC30", "x102"),
    Request("b1", "b1", "xD00", "x103"),
    Request("b1", "b1", "xC00", "x104")
```

- Please refer to `src/main/scala/Tests.scala` for the expected results