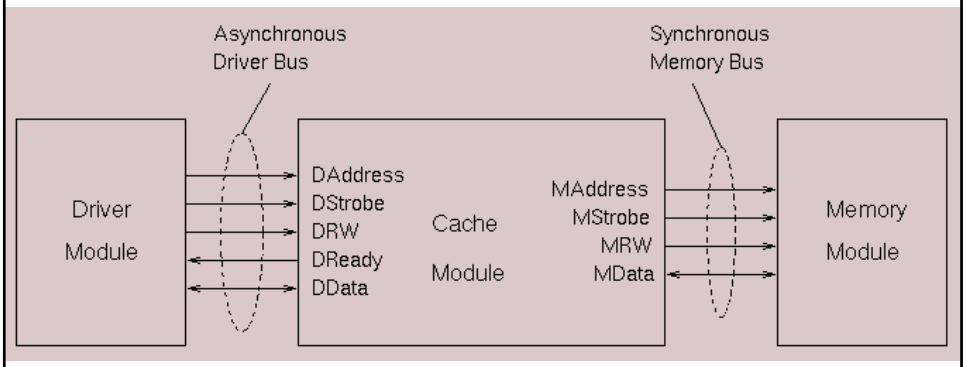
Second Verilog Project (Cache Memory)

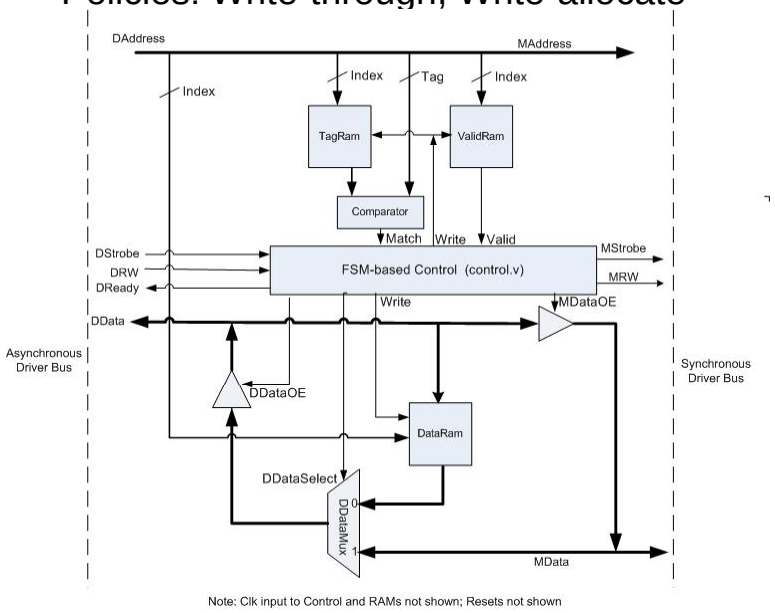
HPCA, Spring 2011

Overall Structure:

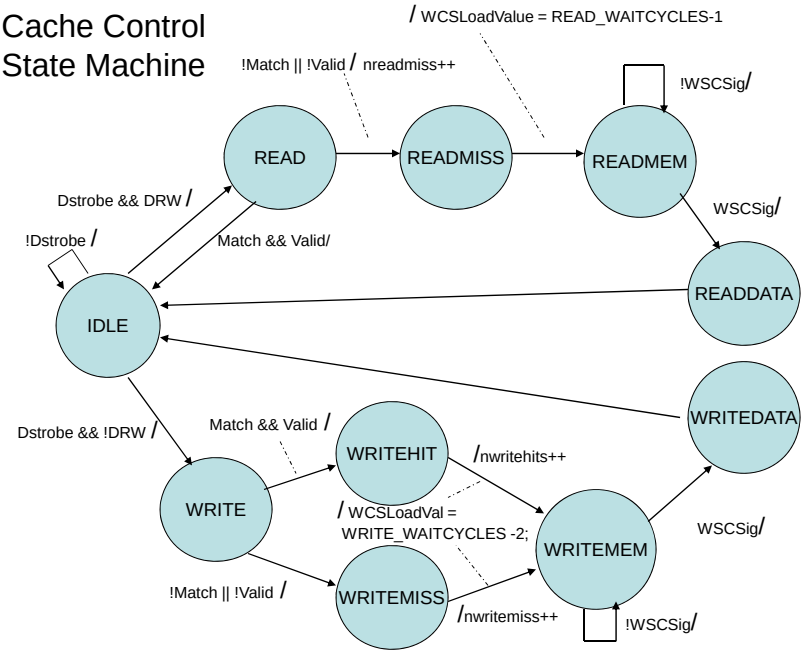


Direct-Mapped Cache Structure

Policies: Write-through, Write-allocate



Cache Control State Machine



Cache Controller--States

- IDLE: no memory access underway
- READ: Read access initiated by driver; Cache is checked during this state. If hit, access is satisfied from cache during this cycle and control returns to IDLE state at next transition. If miss, transition to READMISS state to initiate main memory access
- READMISS: Initiate memory access following a read miss. Wait state counter is loaded to time the wait for completion of the main memory access. Transition to READMEM State.
- READMEM: Main memory read in progress. Remain in this state until wait state counter expires then transition to READDATA state. (Main memory read requires READ_WAITCYCLES cycles to complete)
- READDATA: Data available from main memory read. Write this data into the cache line and use it to satisfy the original processor (driver) read request

Cache Controller States--Continued

- WRITE: Write access initiated by Driver. If cache is hit, transition to WRITEHIT state. If miss, transition to WRITEMISS state.
- WRITEHIT: Cache has been hit on a write operation. Complete write to cache and initiate write-through to main memory. Load wait state counter to time main memory access waiting period. Transition to WRITEMEM state.
- WRITEMISS: Cache has been missed on a write operation. Write to cache (cache load) and initiate write-through to main memory Load wait state timer to time main memory waiting period
- WRITEMEM: Main memory write in progress. Wait for expiration of wait state counter, then transition to WRITEDATA state.
- WRITEDATA: Last Cycle of Main memory write. Assert Ready signal to Driver to indicate completion of write.

Cache Controller Inputs

- Match: Daddress[Tag] == TagRam[Daddress[Index]]
- Valid: ValidRam[Daddress[Index]] == 1'b1
- Dstrobe: Data strobe for asynchrous driver interface (indicates initiation of memory access by Driver)
- DRW: Indicates type of driver memory access (1'b0 indicates write, 1'b1 indicates read)

Cache Contoller Outputs

- WSCLoad: Load Wait State Controller for memory access delay timing
- DReddyEnable: Asserted in Read State. Used in computing the DReady signal for Driver interface (see later slide)
- Ready: Indicates completion of memory access. Used in computing Dready signal for Driver interface (see later slide).
- Write: Update contents of TagRam[Daddress[Index]], ValidRam[Daddress[Index]], and DataRam[Daddress[Index]]
- Mstrobe: Initiate main memory access on synchronous Memory interface
- MRW: Type of main memory access (1'b0 indicates write, 1'b1 indicates read)
- DDataSelect: Selects data source to satisfy Driver read request
- DDataOE: Configures bidirectional Driver bus for Read operation
- Configures bidirectional Memory bus for Write Operation

Control module: State to output mappings

```
task OutputVec;
input   [8:0]   vector;
begin
   WSCLoad          = vector[8];
   DReadyEnable  = vector[7];
   Ready              = vector[6];
   Write              = vector[5];
   MStrobe           = vector[4];
   MRW                = vector[3];
   DDataSelect       = vector[2];
   DDataOE           = vector[1];
   MDataOE           = vector[0];
end
endtask
```

```
task UpdateSignals;
input [3:0] state;
   case (state)
      `STATE_IDLE:          OutputVec(9'b000000000);
      `STATE_READ:          OutputVec(9'b010000010);
      `STATE_READMISS:  OutputVec(9'b100011010);
      `STATE_READMEM:   OutputVec(9'b000001010);
      `STATE_READDATA:  OutputVec(9'b001101110);
      `STATE_WRITE:         OutputVec(9'b000000000);
      `STATE_WRITEHIT:     OutputVec(9'b100110001);
      `STATE_WRITEMISS:  OutputVec(9'b100110001);
      `STATE_WRITEMEM:  OutputVec(9'b000000001);
      `STATE_WRITEDATA: OutputVec(9'b001000001);
   endcase
endtask
```

Control module: State to output mappings

```
task OutputVec;
input   [8:0]   vector;
begin
   WSCLoad          = vector[8];
   DReadyEnable  = vector[7];
   Ready              = vector[6];
   Write              = vector[5];
   MStrobe           = vector[4];
   MRW                = vector[3];
   DDataSelect       = vector[2];
   DDataOE           = vector[1];
   MDataOE           = vector[0];
end
endtask
```

```
task UpdateSignals;
input [3:0] state;
   case (state)
      `STATE_IDLE:          OutputVec(9'b000000000);
      `STATE_READ:          OutputVec(9'b010000010);
      `STATE_READMISS:  OutputVec(9'b100011010);
      `STATE_READMEM:   OutputVec(9'b000001010);
      `STATE_READDATA:  OutputVec(9'b001101110);
      `STATE_WRITE:         OutputVec(9'b000000000);
      `STATE_WRITEHIT:     OutputVec(9'b100110001);
      `STATE_WRITEMISS: OutputVec(9'b100110001);
      `STATE_WRITEMEM: OutputVec(9'b000000001);
      `STATE_WRITEDATA: OutputVec(9'b001000001);
   endcase
endtask
```

Control module: State to output mappings

```
task OutputVec;
input   [8:0]   vector;
begin
  WSCLoad         = vector[8];
  DReadyEnable    = vector[7];
  Ready           = vector[6];
  Write           = vector[5];
  MStrobe         = vector[4];
  MRW             = vector[3];
  DDataSelect     = vector[2];
  DDataOE         = vector[1];
  MDataOE         = vector[0];
end
endtask
```

```
task UpdateSignals;
input [3:0] state;
  case (state)
    `STATE_IDLE:          OutputVec(9'b000000000);
    `STATE_READ:          OutputVec(9'b010000010);
    `STATE_READMISS:  OutputVec(9'b100011010);
    `STATE_READMEM:   OutputVec(9'b000001010);
    `STATE_READDATA:  OutputVec(9'b001101110);
    `STATE_WRITE:         OutputVec(9'b000000000);
    `STATE_WRITEHIT:    OutputVec(9'b100110001);
    `STATE_WRITEMISS:  OutputVec(9'b100110001);
    `STATE_WRITEMEM:  OutputVec(9'b000000001);
    `STATE_WRITEDATA: OutputVec(9'b001000001);
  endcase
endtask
```

Control module: State to output mappings

```
task OutputVec;
input   [8:0]   vector;
begin
  WSCLoad         = vector[8];
  DReadyEnable    = vector[7];
  Ready           = vector[6];
  Write           = vector[5];
  MStrobe         = vector[4];
  MRW             = vector[3];
  DDataSelect     = vector[2];
  DDataOE         = vector[1];
  MDataOE         = vector[0];
end
endtask
```

```
task UpdateSignals;
input [3:0] state;
  case (state)
    `STATE_IDLE:          OutputVec(9'b000000000);
    `STATE_READ:          OutputVec(9'b010000010);
    `STATE_READMISS:  OutputVec(9'b100011010);
    `STATE_READMEM:   OutputVec(9'b000001010);
    `STATE_READDATA:  OutputVec(9'b001101110);
    `STATE_WRITE:         OutputVec(9'b000000000);
    `STATE_WRITEHIT:    OutputVec(9'b100110001);
    `STATE_WRITEMISS:  OutputVec(9'b100110001);
    `STATE_WRITEMEM:  OutputVec(9'b000000001);
    `STATE_WRITEDATA: OutputVec(9'b001000001);
  endcase
endtask
```

Cache Control—Signals Asserted

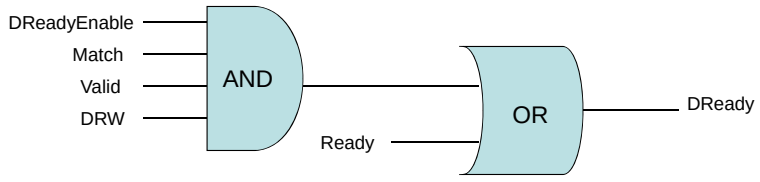
- IDLE: none
- READ: DReadyEnable, DDataOE
- READMISS: WSCLoad, MStrobe, MRW, DDataOE
- READMEM: MRW, DDataOE
- READDATA: Ready, Write, MRW DDataSelect, DDataOE
- WRITE:
- WRITEHIT: Hit, WSCLoad, Write, MStrobe, MDataOE
- WRITEMISS: Miss, WSCLoad, Write, MStrobe, MDataOE
- WRITEMEM: MDataOE
- WRITEDATA: Ready, MDataOE

Explanation of the DReady Signal

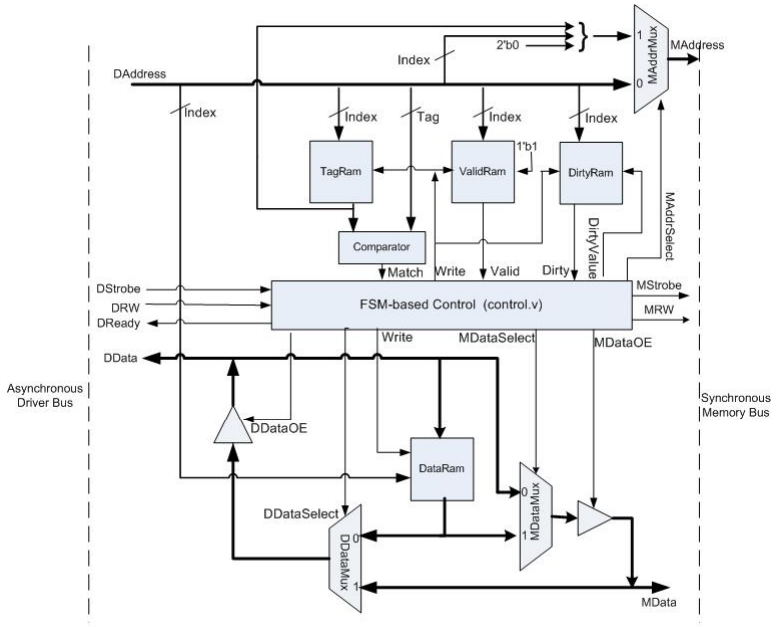
- In module CacheControl, the output DReady is controlled by a “continuous assignment” of the form:

```
wire DReady = (DReadyEnable && Match && Valid && DRW) || Ready;
```

- This is equivalent to:

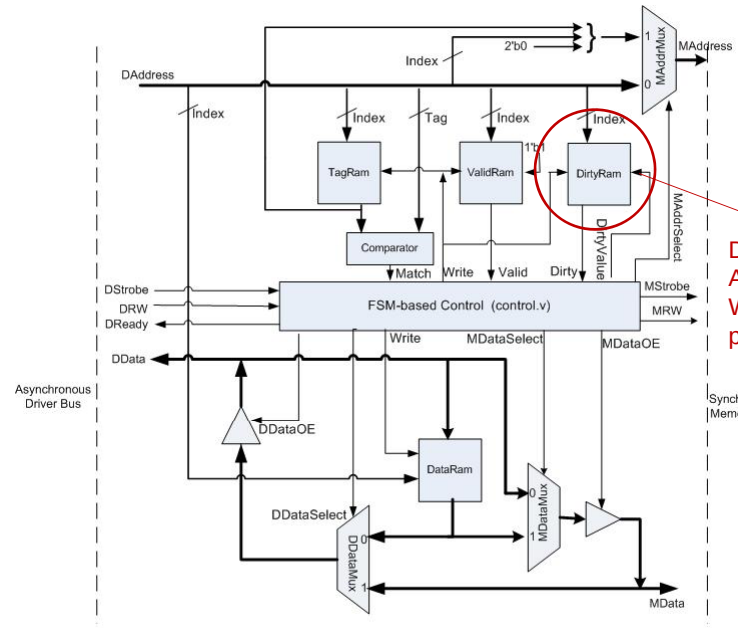


Write-back, Non-allocate Cache



Note: Clk input to Control and RAMs not shown; Resets not shown

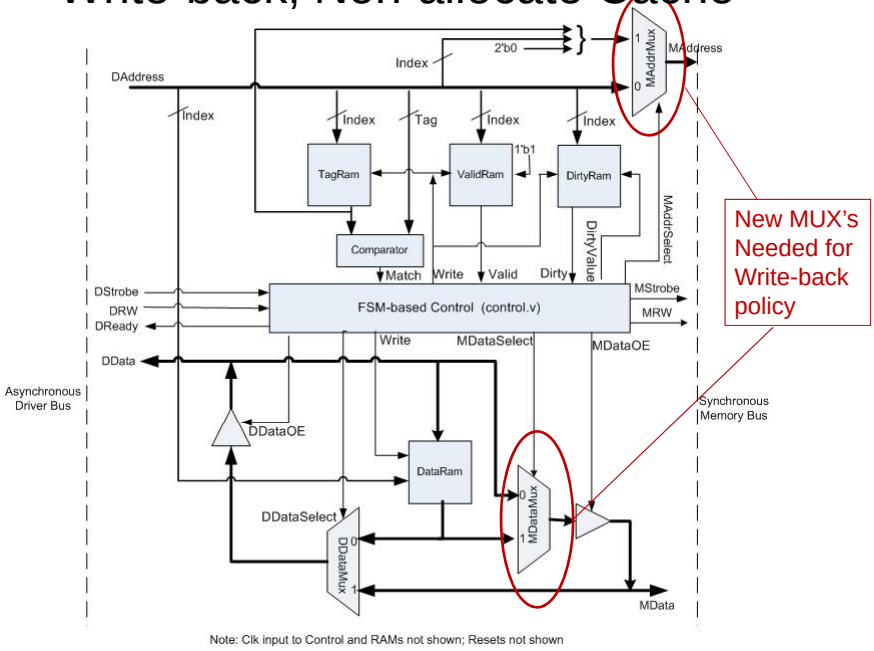
Write-back, Non-allocate Cache



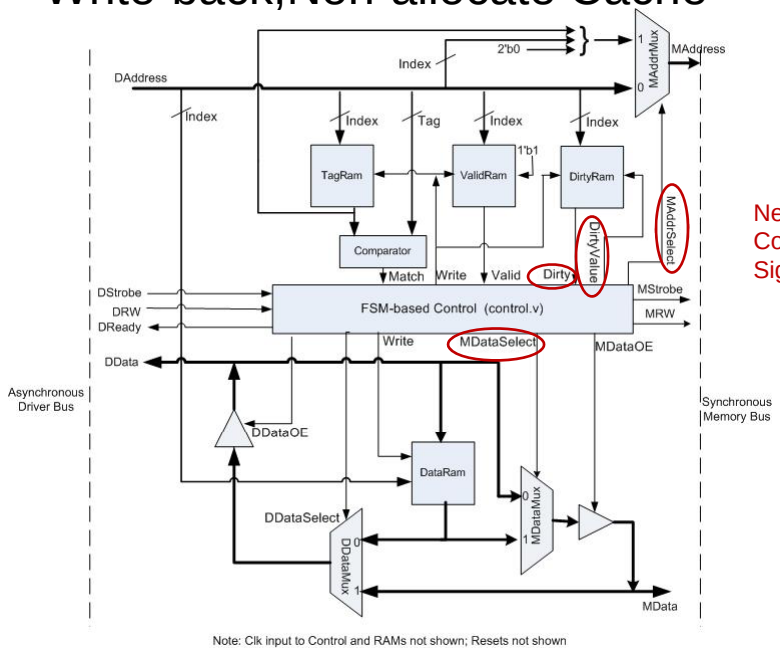
Dirty RAM Added for Write-back policy

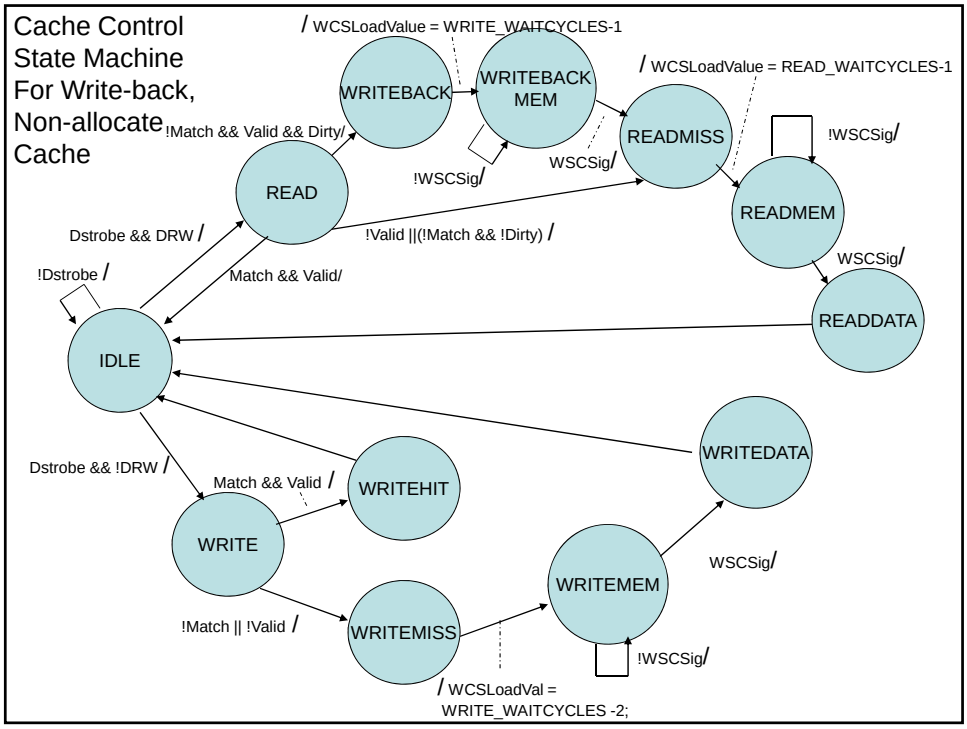
Note: Clk input to Control and RAMs not shown; Resets not shown

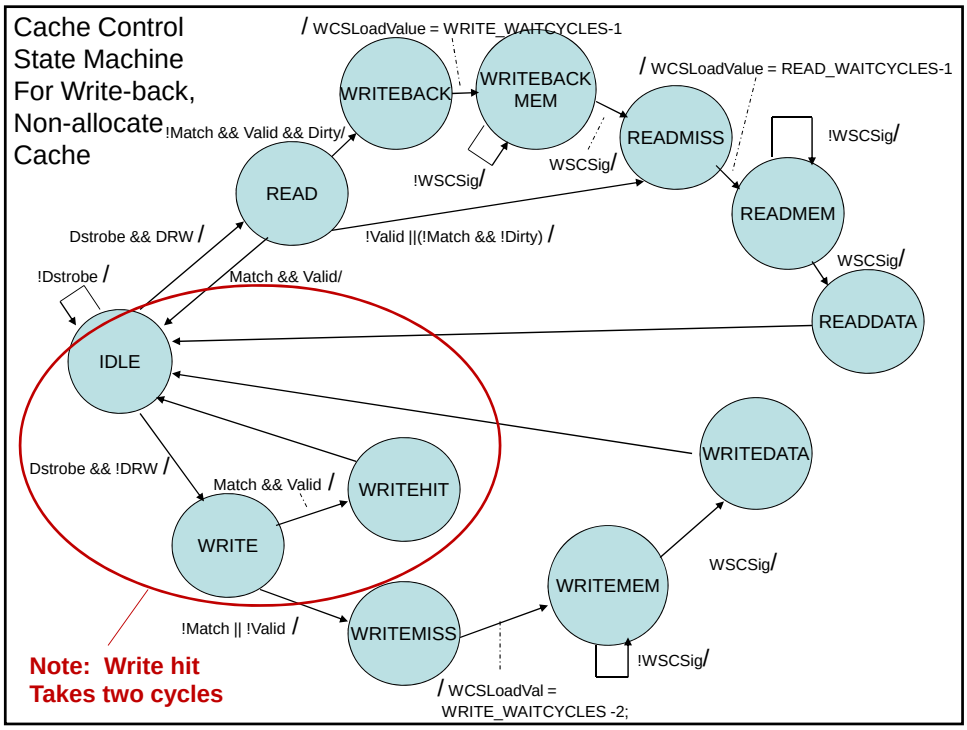
Write-back, Non-allocate Cache



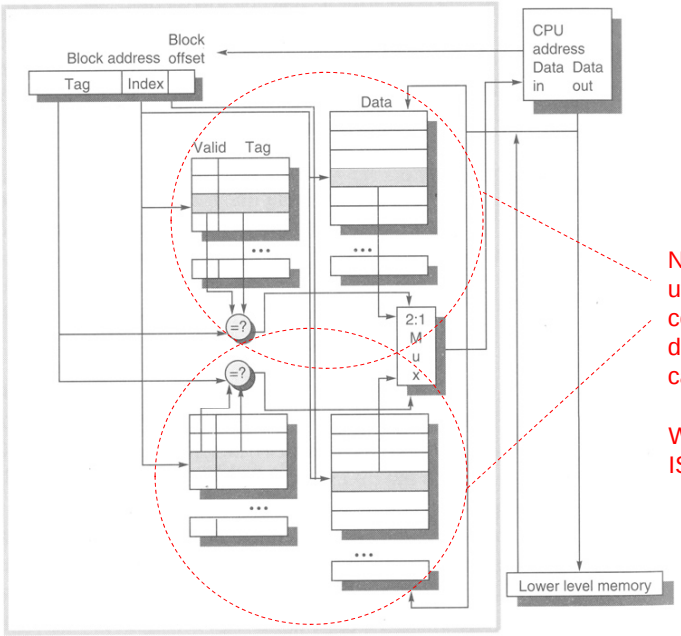
Write-back,Non-allocate Cache




Cache Control
State Machine
For Write-back,
Non-allocate
Cache
/ WCSLoadValue = WRITE_WAITCYCLES-1
WRITEBACK
WRITEBACK MEM
/ WCSLoadValue = READ_WAITCYCLES-1
!Match && Valid && Dirty/
READMISS
!WSCSig/
WSCSig/
!WSCSig/
READ
READMEM
Dstrobe && DRW /
!Valid ||(!Match && !Dirty) /
WSCSig/
!Dstrobe /
Match && Valid/
READDATA
IDLE
WRITEDATA
Dstrobe && !DRW /
Match && Valid /
WRITEHIT
WRITE
WSCSig/
WRITEMEM
!Match || !Valid /
WRITEMISS
!WSCSig/
/ WCSLoadVal =
WRITE_WAITCYCLES -2;


Cache Control
State Machine
For Write-back,
Non-allocate
Cache
/ WCSLoadValue = WRITE_WAITCYCLES-1
WRITEBACK
WRITEBACK MEM
/ WCSLoadValue = READ_WAITCYCLES-1
!Match && Valid && Dirty/
READMISS
!WSCSig/
WSCSig/
!WSCSig/
READ
READMEM
Dstrobe && DRW /
!Valid ||(!Match && !Dirty) /
WSCSig/
!Dstrobe /
Match && Valid/
READDATA
IDLE
WRITEDATA
Dstrobe && !DRW /
Match && Valid /
WRITEHIT
WRITE
WSCSig/
WRITEMEM
!Match || !Valid /
WRITEMISS
!WSCSig/
/ WCSLoadVal =
WRITE_WAITCYCLES -2;
Note: Write hit
Takes two cycles

Two-way set-associative cache



Note: Basically utilizes two copies of a direct mapped cache.

WHAT ELSE IS NEEDED?

Some Additional Pointers

- You should not need to mess with, or understand the internals of, the driver module (driver.v) or the main memory module (hashmem.v)
- Be sure that you **thoroughly** understand your design before making any modifications to the Verilog Model.
- Use the small (12 entry) trace file for debugging purposes, with the verbose and debug options enabled (debug.h)