rst

Idle

Stall = 0
enable = 0

Rd

Wr

Compare Read

comp = 1
write_cache1 = 0
write_cache2 = 0
tag = Addr[15:11]
index = Addr[10:3]
offset = Addr[2:0]

Hit

Done = 1
CacheHit = 1
invert_way = 1
enable = 0

Done

Done = 1
invert_way = 1
enable = 0

Compare Write

comp = 1
write_cache1 = 1
write_cache2 = 1
tag = Addr[15:11]
index = Addr[10:3]
offset = Addr[2:0]

(hit1 & valid1) |
(hit2 & valid2)

(hit1 & valid1) |
(hit2 & valid2)
do not need to woorry about dirty output as this is write

(~hit1 & ~dirty1) |
(~hit2 & ~dirty2) |
(hit1 & ~valid1) |
(hit2 & ~valid2)
not dirty, so just bring in from memory and write into cache, second two cases to handle a hit with not valid (treat as miss)

(~hit1 & valid1 & dirty1) |
(~hit2 & valid2 & dirty2)
if dirty we need to read first

(~hit1 & ~dirty1) |
(~hit2 & ~dirty2) |
(hit1 & ~valid1) |
(hit2 & ~valid2)

(~hit1 & valid1 & dirty1) |
(~hit2 & valid2 & dirty2)

We need to know which data we will evict before Access Read 0 so that we know which cache to cache line we need to store into memory.

neeed to delay 2 cycles for mem reads, so we starting reading word 0 in AW0, and then can get word 0 in AW2

Access Write 0

(reading word 0)

comp = 0
write_cache1 = victimway
write_cache2 = ~victimway
addr_mem = {Addr[15:3], 3'b000}
rd_mem = 1
just reading from mem, so only set mem signals

Access Write 1

(reading word 1)

comp = 0
write_cache1 = victimway
write_cache2 = ~victimway
addr_mem = {Addr[15:3], 3'b010}
rd_mem = 1
just reading from mem, so only set mem signals

Access Write 2

(reading word 2, getting word 0)

comp = 0
write_cache1 = victimway
write_cache2 = ~victimway
valid_out = 1
tag = Addr[15:11]
index = Addr[10:3]
offset = 3'b000
wr = 1
data_out_cache = data_in_mem
addr_mem = {Addr[15:3], 3'b100}
rd_mem = 1
now need to read, and get word

Access Write 3

(reading word 3, getting word 1)

comp = 0
write_cache1 = victimway
write_cache2 = ~victimway
valid_out = 1
tag = Addr[15:11]
index = Addr[10:3]
offset = 3'b010
wr = 1
data_out_cache = data_in_mem
addr_mem = {Addr[15:3], 3'b110}
rd_mem = 1
now need to read, and get word

Access Write 4

(getting word 2)

comp = 0
write_cache1 = victimway
write_cache2 = ~victimway
valid_out = 1
tag = Addr[15:11]
index = Addr[10:3]
offset = 3'b100
wr = 1
data_out_cache = data_in_mem
now just getting word

Access Write 5

(getting word 3)

comp = 0
write_cache1 = victimway
write_cache2 = ~victimway
valid_out = 1
tag = Addr[15:11]
index = Addr[10:3]
offset = 3'b110
wr = 1
data_out_cache = data_in_mem
now just getting word

Access Read 0

comp = 0
index = Addr[10:3]
offset = 3'b000
addr_mem = {Addr[15:3], offset}
data_out_mem = data_in_cache *(select correct)*
mem_wr = 1

Access Read 1

comp = 0
index = Addr[10:3]
offset = 3'b010
addr_mem = {tag_in, Addr[10:3], offset}
data_out_mem = data_in_cache *(select correct)*
mem_wr = 1

Access Read 2

comp = 0
index = Addr[10:3]
offset = 3'b100
addr_mem = {tag_in, Addr[10:3], offset}
data_out_mem = data_in_cache *(select correct)*
mem_wr = 1

Access Read 3

comp = 0
index = Addr[10:3]
offset = 3'b110
addr_mem = {tag_in, Addr[10:3], offset}
data_out_mem = data_in_cache *(select correct)*
mem_wr = 1

finished reading dirty, now write

Key

red = mem_system.v output
purple = mem_system.v input
blue = cache.v inputs
orange = cache.v outputs
green = four_bank_mem.v inputs
teal = four_bank_mem.v outputs
black = internal to the controller

State Machine Default Outputs:

write_cache1 = 1'b0;
write_cahce2 = 1'b0;
tag = 5'bxxxxx;
index = 8'hxx
offset = 3'bxx;
data_out_mem = *16'hxxxx;*
data_out_cache = 16'hxxxx;
comp = 1'b0;
enable = 1'b1;
valid_out = 1'b0;
addr_mem = 16'hxxxx;
wr_mem = 1'b0;
rd_mem = 1'b0;
CacheHit = 1'b0;
Stall = 1'b1;
Done = 1'b0;
DataOut = 16'hxxxx;

If there is a miss, decide which cache module to victimize based on this logic: If one is valid, select the other one. If neither is valid, select way zero. If both are valid, use the pseudo-random replacement algorithm.

In Access Write: Make sure only the correct module has its write input asserted.