VE370 Project3 Memory Hierarchy

Write back+direct mapped

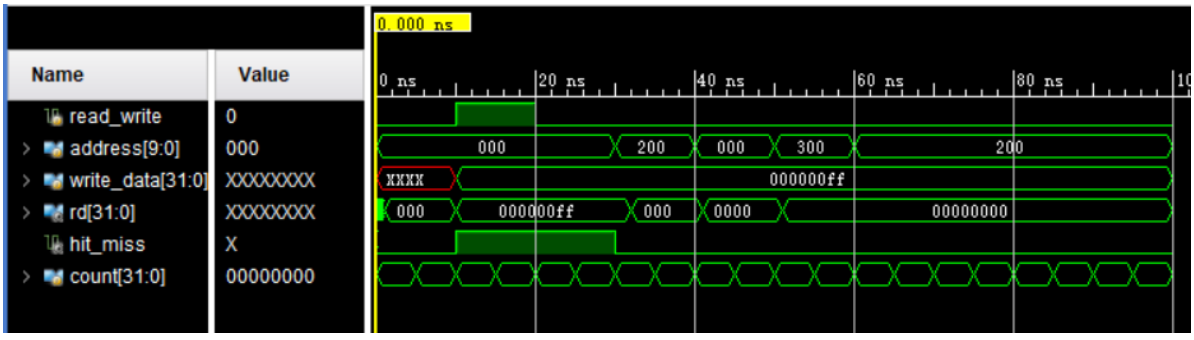
The testbench is posted below.

`

```
initial
begin
    #0 read_write = 0; address = 10'b0000000000; //should miss
    #10 read_write = 1; address = 10'b0000000000; write_data = 8'b11111111;
//should hit
    #10 read_write = 0; address = 10'b0000000000; //should hit and read out 0xff

//here check main memory content,
//the first byte should remain 0x00 if it is write-back,
//should change to 0xff if it is write-through.

#10 read_write = 0; address = 10'b1000000000; //should miss
#10 read_write = 0; address = 10'b0000000000; //should hit for 2-way
associative, should miss for directly mapped

#10 read_write = 0; address = 10'b1100000000; //should miss
#10 read_write = 0; address = 10'b1000000000; //should miss both for directly
mapped and for 2-way associative (Least-Recently-Used policy)

//here check main memory content,
//the first byte should be 0xff
end
```

`

The result are shown below, and we'll make brief explanation.



```
# run 100ns   //count*1=5ns
count=       0 MainMemory[0] = 0x00000000
--------------------------------------------------
count=       1 MainMemory[0] = 0x00000000
--------------------------------------------------
count=       2 MainMemory[0] = 0x00000000
--------------------------------------------------
count=       3 MainMemory[0] = 0x00000000
--------------------------------------------------
count=       4 MainMemory[0] = 0x00000000
--------------------------------------------------
count=       5 MainMemory[0] = 0x00000000
--------------------------------------------------
count=       6 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=       7 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=       8 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=       9 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=      10 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=      11 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=      12 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=      13 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=      14 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=      15 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=      16 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=      17 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=      18 MainMemory[0] = 0x000000ff
--------------------------------------------------
count=      19 MainMemory[0] = 0x000000ff
```

First what need to be clarify is that our simulation lasts for 100ns, and every 10ns there's a new input. Our console output is updated every 5ns (although seemingly redundant eventually), so count 0-1 corresponds to instruction1, count2-3 corresponds to instruction2 so on so forth.

The data size is 32 bits. Cache size 16 words. Main memory size 256 words.

For instruction 1,(#0) it tries to read information at 0x000. At first there's nothing in cache, so definitely it is a miss, then the information, which is 0x00000000 together with the block that contains it, is transferred to cache[0], and then read to output. We can see rd (read_data) equals 0x00000000 in the graph.

For instruction 2, (#10) it writes 0x000000ff into the same address. Because we've access this address in instruction 1, so it's a hit (see the graph). But the memory is still 0x00000000 because we just modify the cache and **write dirty to 1** in this case. The rd signal is 0x000000ff, but because read_write control signal says 'write', **so rd does no effect**. It has value because of design issue.

For instruction 3(#20), hit, now rd has effect. Its value now (0x000000ff) is the information stored in required address. This block is still dirty.

For instruction 4(#30), miss. Before we read the **dirty block**, the **information in the cache is first written back** into main memory. So now we can see main memory has changes (count 6). Then the **information in 0x10000000 is copied to cache and read** (0x00000000). Now **dirty=0.**There's **a delay in the graph to indicate that main memory is accessed.**

For instruction 5(#40), miss. Because not dirty, so directly copy from main memory.**0x000000ff is again written into cache from main memory,** and then read out. dirty=0. The value shown in the graph is 0x0000.... actually it's 0x000000ff. Also we can see there's a delay, meaning the main memory is indeed accessed.

The rest 3 instruction are all miss. The process is the same as instruction 5.