LOGIC CIRCUIT

Verilog project: Direct-mapped-cache

Dr. FakhrAhmad

Spring 2025

Part I: The Basics

1.1 direct mapped cache

For this project you have been instructed to make a direct-mapped-cache using Verilog

This cache consists of 16 lines with 4 words per line, and the RAM we are working with is 1KB (1024 Bytes).

1.2 Inputs and outputs

These are the ports used in the module

- Clk: perform like a normal clock on a positive edge(input).
- Address: the 32-bit address of the data we are looking for (input).
- Reset: resets the cache (input).
- Write-enable: tells the cache whether it should read or write (1 is write 0 is read) (input).
- Write-data: the 32-bit data we want to write to memory (input).
- Data-out: the 32-bit data at the address value we provided(output).
- Hit-miss: determines whether the accesses was a hit or a miss(output).
- Total_accesses: determines the amount of times we have accessed the cache (output) .
- Total_misses: determines the amount of cache-misses we have seen after all the accesses (output).

Part II: registers

We have four main registers that work with the positive edge of the clock (when we have a pulse our registers update on the positive edge) you can add others if the need arises .

- Cache_Memory: our Direct-mapped-cache that is a two-dimensional array in Verilog were each element is 32-bits

(reg [31:0] Cache_Memory [0:NUM_LINES-1][0:WORDS_PER_LINE-1])

- Cache_tag: a one-dimensional array in Verilog that holds the tags of each line of the cache

(reg [17:0] Cache_Tags [0:NUM_LINES-1])

- Valid_bit: if a line in cache has an address stored in it the valid-bit will be 1 otherwise its 0

(reg Valid_Bit [0:NUM_LINES-1])

- Memory: it's a byte addressable memory

(reg [7:0] memory [0:MEM_SIZE-1])

Part III: The actual cache

3.1 Implementation

Make a direct mapped cache with 16 lines and 4 words per line

1. You should read the data in memory from a separate file using this command { $readmemb("memory.list", memory) }.
2. If the reset pin is 1 you should reset the cache.
3. Check if we want to write or read (using write-enable port in section 1.2).
4. If we want to read check if the given address is a hit or miss.
5. If it's a hit our data_out(section 1.2) should be the data stored in the memory location.
6. If it's a miss you should load the address into the cache.

3.2 Testbench

You should write a testbench for your module but a sample will be provided

In your testbench after giving the address you should use $display to display

1. The address
2. Data_out
3. Hit or miss
4. Total_accesses
5. Total misses so far

For example for address 0 we have :

$display("Read Addr=0: Data_Out=%h, Hit_Miss=%b, Acc=%0d, Miss=%0d", Data_Out, Hit_Miss, total_accesses, total_misses);

Bonuses

1. Provide a full documentation of all the steps you took to complete the project

2. If your code is easy to read(has comments ….) you will be given some extra points

Final Remarks

Submit the (project file including the .xpr file if you are using vivado) or (the .v file if you are using VScode or other apps) and the documentation as a single zip file.

Upload with the given format:

FirstName_LastName_StudentNumber_CacheVerilog.zip

Associated TAs

Masiha Mostofizadeh , Mojtaba Zandavi , Erfan Attar, Moein Yeganeh , Seyed Ehsan Mousavi, Ahmad Shams, Ali Nickhoo , Mohsen Taheri