Lab 3 Report

|  |  |
| --- | --- |
| Name 1: Brandon Hoang | Name 2: ---------- |
| Course: EGCP-381 | Date: 3/05/2019 |

Grading Criteria:

|  |  |  |
| --- | --- | --- |
| **Section** | **Earned Points** | **Possible Points** |
| Problem/Objective: |  | 5 |
| Method: |  | 20 |
| Question(s): |  | 30 |
| Program Code: |  | 25 |
| Test Bench: |  | 20 |
| Total: | 0 | 100 |

Report Submission Instructions:

* Only **two** people per group and **both members must upload the same report.**
* Please, upload your report to TITANIum
* No paper submissions
* When showing your work, you can use MS Word’s equation tool. Or you can hand write your work, take a picture, and paste the image here. One app that I would suggest to easily do this is “CamScanner”.
* When giving the screenshots, please take a screenshot of the whole screen (i.e., include the OS taskbar, date, clock, etc.). No cropping.

# Problem/Objective

State the problem statement and/or objective of the lab. This must be a complete paragraph (i.e., at least 5 sentences).

The main objective is to learn direct cache mapping algorithms through VHDL simulation. Additionally, doing so will also demonstrate how to create FSMs in VHDL. We will be specifically mapping the direct cache, then simulating the output waveforms. In this lab the cache and main memory code are given. The cache controller and the component implementation must be completed in this lab.

# Methodology

You are to search on the web (i.e., [CSUF library](http://library.fullerton.edu/), [Google Scholar](http://scholar.google.com/), etc.) to find a scholarly paper (i.e., 1 paper) on Direct Cache Mapping. Briefly describe (about 2 complete paragraphs) what the paper is about.

<https://dl.acm.org/citation.cfm?id=325162>

This paper is about *Miss caching*, which “places a small fully-associative cache between a cache and its refill path”. According to the paper, misses in cache that are a hit in the miss cache only have a penalty of one cycle miss, which is a step up from many cycle misses without this miss cache. These miss caches can be small (2 to 5 entries) but still be effective in alleviating mapping conflict misses.

*Victim caching* builds upon this technique by loading the fully-associative cache with the ‘victim’ of a miss instead of a requested line. The victim cache is effective at 1 to 5 entries and is more effective than miss caching. *Stream buffers* pre-fetch cache lines that start at the miss address, and remove capacity, some instruction cache misses, and compulsory cache misses.

# Question(s)

Please, show your work. Don’t just give a single value. Explain how you obtained that number. HINT: see the Lecture 6.

1. What is the address length?

Address length = s +w

S = tag + line = 8 bits

W = 4 bits

Address length = 8 + 4 = 12 bits

1. What is the number of addressable units?
2. What is the block size?
3. How many blocks are in main memory?
4. How many lines of cache are there?
5. What is the size of the cache?

# Program Code

Copy your code here. Please provide comments in your code. This will help me analyze your code and remove any ambiguity. **Provide your code as text, not as a screenshot/image**.

***Direct Cache Controller:***

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

use work.definitions.all;

entity direct\_cache\_ctrl is

port (

clk, rst : in std\_logic;

addr : in std\_logic\_vector((cache\_tag\_width + cache\_line\_width + cache\_word\_width)-1 downto 0);

hit : out std\_logic;

cache\_data\_from : in std\_logic\_vector(cache\_data\_width-1 downto 0);

cache\_we : out std\_logic;

cache\_line : out std\_logic\_vector(cache\_line\_width-1 downto 0);

cache\_tag : out std\_logic\_vector(cache\_tag\_width-1 downto 0);

cache\_data\_to : out std\_logic\_vector(cache\_data\_width-1 downto 0);

main\_mem\_data\_from : in std\_logic\_vector(main\_mem\_data\_width-1 downto 0);

main\_mem\_addr : out std\_logic\_vector(main\_mem\_adr\_width-1 downto 0);

state : out integer

);

end direct\_cache\_ctrl;

architecture behavioral of direct\_cache\_ctrl is

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--\* TODO: Add FSM states and any signals

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

type eg\_state\_type is (CACHE\_R, GET\_TAG, GET\_TAG2, COMP\_TAGS, MAIN\_MEM\_R, CACHE\_UPDATE, CACHE\_UPDATE2);

signal state\_reg, state\_next: eg\_state\_type;

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--\* Don't delete/change the following signals

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

signal tag : std\_logic\_vector(cache\_tag\_width-1 downto 0) := (others => '0');

signal line : std\_logic\_vector(cache\_line\_width-1 downto 0) := (others => '0');

signal word : std\_logic\_vector(cache\_word\_width-1 downto 0) := (others => '0');

signal cache\_tag\_sig : std\_logic\_vector(cache\_tag\_width-1 downto 0) := (others => '0');

signal addr\_len : integer := cache\_tag\_width + cache\_line\_width + cache\_word\_width;

begin

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--\* Don't delete/change the following

--\* Seperate tag, line, and word concurrently

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

tag <= addr(addr\_len-1 downto (addr\_len-cache\_tag\_width));

line <= addr((addr\_len-cache\_tag\_width)-1 downto (addr\_len-cache\_tag\_width-cache\_line\_width));

word <= addr((addr\_len-cache\_tag\_width-cache\_line\_width)-1 downto 0);

cache\_line <= line;

cache\_tag <= tag;

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--\* TODO: State register

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

process(clk, rst)

begin

if (rst = '1') then

state\_reg <= CACHE\_R;

elsif (rising\_edge(clk)) then

state\_reg <= state\_next;

end if;

end process;

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--\* TODO: Next-state

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

process(state\_reg)

begin

case state\_reg is

when CACHE\_R =>

state\_next <= GET\_TAG;

when GET\_TAG =>

state\_next <= GET\_TAG2;

when GET\_TAG2 =>

state\_next <= COMP\_TAGS;

when COMP\_TAGS =>

if ( cache\_tag\_sig = tag ) then

state\_next <= CACHE\_R;

else

state\_next <= MAIN\_MEM\_R;

end if;

when MAIN\_MEM\_R =>

state\_next <= CACHE\_UPDATE;

when CACHE\_UPDATE =>

state\_next <= CACHE\_UPDATE2;

when CACHE\_UPDATE2 =>

state\_next <= CACHE\_R;

end case;

end process;

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--\* TODO: Moore logic

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

process(state\_reg)

begin--/Users/ybai/Dropbox (CSU Fullerton)/CSUF-Teaching/Spring 2019/EGCP381/Lab/Lab 3/Template Code/direct\_cache.vhd

case state\_reg is

when CACHE\_R =>

state <= 1;

cache\_we <= '0';

when GET\_TAG =>

state <= 2;

when GET\_TAG2 =>

state <= 3;

cache\_tag\_sig <= cache\_data\_from (35 downto 32);

when COMP\_TAGS =>

state <= 4;

when MAIN\_MEM\_R =>

state <= 5;

main\_mem\_addr(7 downto 4) <= tag;

main\_mem\_addr(3 downto 0) <= line;

when CACHE\_UPDATE =>

state <= 6;

cache\_we <= '1';

when CACHE\_UPDATE2 =>

state <= 7;

cache\_data\_to(31 downto 0) <= main\_mem\_data\_from;

cache\_data\_to(35 downto 32) <= tag;

end case;

end process;

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--\* TODO: Mealy logic

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

process(state\_reg)

begin

if ( state\_reg = COMP\_TAGS ) then

if (cache\_tag\_sig = tag) then

hit <= '1';

else

hit <= '0';

end if;

end if;

if ( state\_reg = CACHE\_UPDATE2 ) then

hit <= '1';

end if;

end process;

end behavioral;

***Direct Cache:***

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

use work.definitions.all;

entity direct\_cache is

port (

clk : in std\_logic;

rst : in std\_logic;

addr : in std\_logic\_vector((cache\_tag\_width + cache\_line\_width + cache\_word\_width)-1 downto 0);

hit : out std\_logic;

cache\_we : out std\_logic;

cache\_data\_from : out std\_logic\_vector(cache\_data\_width-1 downto 0);

cache\_line : out std\_logic\_vector(cache\_line\_width-1 downto 0);

cache\_tag : out std\_logic\_vector(cache\_tag\_width-1 downto 0);

cache\_data\_to : out std\_logic\_vector(cache\_data\_width-1 downto 0);

main\_mem\_addr : out std\_logic\_vector(main\_mem\_adr\_width-1 downto 0);

main\_mem\_data\_from : out std\_logic\_vector(main\_mem\_data\_width-1 downto 0);

state : out integer

);

end direct\_cache;

architecture structural of direct\_cache is

component direct\_cache\_ctrl is

port (

clk, rst : in std\_logic;

addr : in std\_logic\_vector((cache\_tag\_width + cache\_line\_width + cache\_word\_width)-1 downto 0);

hit : out std\_logic;

cache\_data\_from : in std\_logic\_vector(cache\_data\_width-1 downto 0);

cache\_we : out std\_logic;

cache\_line : out std\_logic\_vector(cache\_line\_width-1 downto 0);

cache\_tag : out std\_logic\_vector(cache\_tag\_width-1 downto 0);

cache\_data\_to : out std\_logic\_vector(cache\_data\_width-1 downto 0);

main\_mem\_data\_from : in std\_logic\_vector(main\_mem\_data\_width-1 downto 0);

main\_mem\_addr : out std\_logic\_vector(main\_mem\_adr\_width-1 downto 0);

state : out integer

);

end component;

component main\_mem is

port (

clk, rst : in std\_logic;

addr : in std\_logic\_vector(main\_mem\_adr\_width-1 downto 0);

data\_out : out std\_logic\_vector(main\_mem\_data\_width-1 downto 0)

);

end component;

component cache is

port (

clk, rst : in std\_logic;

we : in std\_logic;

line : in std\_logic\_vector(cache\_line\_width-1 downto 0);

data\_in : in std\_logic\_vector(cache\_data\_width-1 downto 0);

data\_out : out std\_logic\_vector(cache\_data\_width-1 downto 0)

);

end component;

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--\* TODO: Add any signal here

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

signal cache\_we0: std\_logic;

signal cache\_line0: std\_logic\_vector(3 downto 0);

signal cache\_data\_to0: std\_logic\_vector(35 downto 0);

signal main\_mem\_addr0: std\_logic\_vector(7 downto 0);

signal main\_mem\_data\_from0: std\_logic\_vector(31 downto 0);

signal cache\_data\_from0: std\_logic\_vector (35 downto 0);

begin

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--\* TODO: Complete the structural model

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CACHE0: cache port map (clk, rst, cache\_we0, cache\_line0, cache\_data\_to0, cache\_data\_from0 );

MAIN\_MEM0: main\_mem port map (clk, rst, main\_mem\_addr0, main\_mem\_data\_from0 );

CTRL: direct\_cache\_ctrl port map (clk, rst, addr, hit, cache\_data\_from0, cache\_we0,

cache\_line0, cache\_tag, cache\_data\_to0, main\_mem\_data\_from0, main\_mem\_addr0, state );

-- CACHE

cache\_data\_from <= cache\_data\_from0;

-- MEM

main\_mem\_data\_from <= main\_mem\_data\_from0;

-- CONTROL

main\_mem\_addr <= main\_mem\_addr0;

cache\_line <= cache\_line0;

cache\_we <= cache\_we0;

cache\_data\_to <= cache\_data\_to0;

end structural;

# Test Bench

Copy your test bench code here. Again, please provide comments in your code. This will help me analyze your code and remove any ambiguity. **Provide your test bench code as text, not as a screenshot/image**.

Also in this section, provide the waveform output of your test bench. Please, properly label the waveform. The waveform must be an image embedded in the document.

***Direct Cache Test Bench***

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

use ieee.std\_logic\_unsigned.all;

use work.definitions.all;

entity direct\_cache\_tb is

end direct\_cache\_tb;

architecture behavior of direct\_cache\_tb is

-- component declaration for the unit under test (uut)

component direct\_cache

port (

clk : in std\_logic;

rst : in std\_logic;

addr : in std\_logic\_vector((cache\_tag\_width + cache\_line\_width + cache\_word\_width)-1 downto 0);

hit : out std\_logic;

cache\_we : out std\_logic;

cache\_data\_from : out std\_logic\_vector(cache\_data\_width-1 downto 0);

cache\_line : out std\_logic\_vector(cache\_line\_width-1 downto 0);

cache\_tag : out std\_logic\_vector(cache\_tag\_width-1 downto 0);

cache\_data\_to : out std\_logic\_vector(cache\_data\_width-1 downto 0);

main\_mem\_addr : out std\_logic\_vector(main\_mem\_adr\_width-1 downto 0);

main\_mem\_data\_from : out std\_logic\_vector(main\_mem\_data\_width-1 downto 0);

state : out integer

);

end component;

--inputs

signal clk : std\_logic := '0';

signal rst : std\_logic := '0';

signal addr : std\_logic\_vector(9 downto 0) := (others => '0');

--outputs

signal hit : std\_logic;

signal cache\_we : std\_logic;

signal cache\_data\_to : std\_logic\_vector(35 downto 0);

signal cache\_line : std\_logic\_vector(3 downto 0);

signal cache\_tag : std\_logic\_vector(3 downto 0);

signal cache\_data\_from : std\_logic\_vector(35 downto 0);

signal main\_mem\_addr : std\_logic\_vector(7 downto 0);

signal main\_mem\_data\_from : std\_logic\_vector(31 downto 0);

signal state : integer;

-- clock period definitions

constant clk\_period : time := 20 ns;

begin

-- instantiate the unit under test (uut)

uut: direct\_cache port map (

clk => clk,

rst => rst,

addr => addr,

hit => hit,

cache\_we => cache\_we,

cache\_data\_to => cache\_data\_to,

cache\_line => cache\_line,

cache\_tag => cache\_tag,

cache\_data\_from => cache\_data\_from,

main\_mem\_addr => main\_mem\_addr,

main\_mem\_data\_from => main\_mem\_data\_from,

state => state

);

-- clock process definitions

clk\_process :process

begin

clk <= '0';

wait for clk\_period/2;

clk <= '1';

wait for clk\_period/2;

end process;

-- stimulus process

stim\_proc: process

variable addr\_len : integer := cache\_tag\_width + cache\_line\_width + cache\_word\_width;

begin

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--\* TODO: Complete the test bench

--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

rst <= '1';

wait for clk\_period;

rst <= '0';

addr <= "0000000000";

wait for clk\_period\*4;

addr <= "1111011100";

wait for clk\_period\*7;

addr <= "0000000000";

wait for clk\_period\*4;

addr <= "1111011100";

wait for clk\_period\*4;

addr <= "0110011000";

wait for clk\_period\*7;

addr <= "0110011000";

wait;

-- End

assert false report "End of Simulation" severity failure;

end process;

end;