UCSD CSE140L Spring 2014

**LAB#4 Report**

Demonstration Date : 6 / 2 /14 Student CID\_\_\_\_\_\_\_\_268\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Student Name: \_\_\_\_\_\_\_\_Kieth\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Vo\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

First M.I. Last

**TED Submission Date & Time :**

(FILLED BY Student BEFORE DEMO) (\*\*\* FILLED BY TUTOR/INSTRUCTOR \*\*\*)

**Self-test Report** Demo Reviewer

Name : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Working Not working **Demo** score **Report** score

**Part1**: \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_/3 Procedural Description( )/1

**Part2**: \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_/3 Verilog HDL codes ( )/1

**Part3**: \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_/3 State Diagram ( )/2

**Part4**: \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_/3 Compilation Report ( )/1

(Screen copy)

**Part5**: \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_ /3

**Subtotal**  **Subtotal**

\_\_\_\_\_\_\_\_/15 \_\_\_\_\_\_\_\_\_\_\_\_\_/5

**TOTAL Score:** **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/20**

* 1. For the first part I made a 4 bit variable and initialized it to 5. Then I had another variable which would be used to hold the value of the XNOR operation. In execute\_random I set register\_A to be equal to the 4 bit variable and then shifted the variable left by one and set the least significant bit to the value of the XNOR operation. Finally I fetched the next state.
  2. For this part I added the operation for memory\_address\_register to update to the memory\_data\_register for execute\_addind. Then I just called the fetch state and then the add state in order to get the data and add it to register\_A.
  3. For this part I did the same thing as part 2 except this time the memory address register was changed to program\_counter + instruction\_register[7:0] for execute\_addpcr.
  4. I looked through the code and found that SW[7] was a conditional statement for the clock. Then I tested it after loading the board and found that it did indeed freeze the clock.
  5. For this part I looked into the clock module and saw when the clock was reset. I added an input to the module which was SW[8] and then I added an if statement for SW[8]. If it was off then everything stayed the same. If it was on then the clock would reset at the DIV\_CONST variable divided by 5 which meant that it was 5 times faster.

// 7-seg display mux

always @ (\*)

begin

case (SW[2:0])

3'b000: hexdata <= 16'h0268;

3'b001: hexdata <= register\_A ;

3'b010: hexdata <= program\_counter ;

3'b011: hexdata <= instruction\_register ;

3'b100: hexdata <= memory\_data\_register\_out ;

3'b111: hexdata <= out;

default: hexdata <= 16'h0268 ;

endcase

end

clock\_divider clk1Hzfrom50MHz (

CLOCK\_50,

KEY[3],

clk\_1Hz,

SW[8]

);

// State Encodings

parameter reset\_pc = 5'h0,

fetch = 5'h1,

decode = 5'h2,

execute\_add = 5'h3,

execute\_store = 5'h4,

execute\_store2 = 5'h5,

execute\_store3 = 5'h6,

execute\_load = 5'h7,

execute\_jump = 5'h8,

execute\_jump\_n = 5'h9,

execute\_out = 5'ha,

execute\_xor = 5'hb,

execute\_or = 5'hc,

execute\_and = 5'hd,

execute\_jpos = 5'he,

execute\_jzero = 5'hf,

execute\_addi = 5'h10,

execute\_shl = 5'h11,

execute\_shr = 5'h12,

execute\_sub = 5'h13,

execute\_random = 5'h14,

execute\_addind = 5'h15,

execute\_addpcr = 5'h16;

reg [3:0] randomNumber = 4'b0101;

reg [1:0] placeHolder;

// Execute random

execute\_random:

begin

register\_A <= randomNumber;

placeHolder = randomNumber[3] ~^ randomNumber[2];

randomNumber = randomNumber << 1;

randomNumber[0] = placeHolder;

state <= fetch;

end

// Execute addind

execute\_addind:

begin

state <= fetch;

state <= execute\_add;

end

// Execute addpcr

execute\_addpcr:

begin

state <= fetch;

state <= execute\_add;

end

execute\_random: memory\_address\_register <= program\_counter;

execute\_addind: memory\_address\_register <= memory\_data\_register;

execute\_addpcr: memory\_address\_register <= program\_counter + instruction\_register[7:0];

8'b01000101:

state <= execute\_random;

8'b01000110:

state <= execute\_addind;

8'b01000111:

state <= execute\_addpcr;

module clock\_divider (clk, rst\_n, clk\_o, fast);

parameter DIV\_CONST = 10000000;

input clk;

input rst\_n;

input fast;

output reg clk\_o;

reg [31:0] div;

reg en;

always @ (posedge clk or negedge rst\_n)

begin

if (!rst\_n)

begin

div <= 0;

en <= 0;

end

else

if (!fast)

begin

begin

if (div == DIV\_CONST)

begin

div <= 0;

en <= 1;

end

else

begin

div <= div + 1;

en <= 0;

end

end

end

else if (fast)

begin

begin

if (div == (DIV\_CONST / 5))

begin

div <= 0;

en <= 1;

end

else

begin

div <= div + 1;

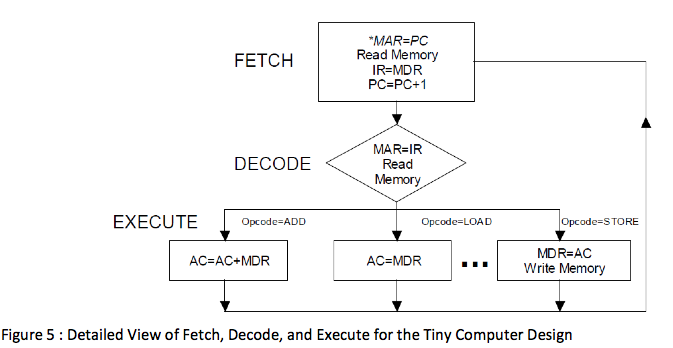
en <= 0;

end

end

end

end



Opcode = RANDOM OPCODE = ADDIND OPCODE = ADDPCR

AC <= 4-bit random number AC <= AC + IMA AC <= AC + PCR

