UCSD CSE140L Spring 2014

LAB#4 Report

Demonstration Date : 06 / 3 /14 Student CID_____580_____

Student Name: _____Adrian_____Jimenez_____
First M.I. Last

TED Submission Date & Time : June 3, 2014 9:30am

(FILLED BY Student BEFORE DEMO)

Self-test Report

	Working	Not working
Part1:	____X_____	_____
Part2:	____X_____	_____
Part3:	____X_____	_____
Part4:	____X_____	_____
Part5:	____X_____	_____

(*** FILLED BY TUTOR/INSTRUCTOR ***)

Demo Reviewer Name : _____

Demo score	**Report** score
_____/3	Procedural Description( )/1
_____/3	Verilog HDL codes ( )/1
_____/3	State Diagram ( )/2
_____/3	Compilation Report ( )/1 (Screen copy)
_____ /3	
Subtotal	**Subtotal**
_____/15	_____/5

TOTAL Score: _____/20

a) Description:
    a. In the precondition, Since the program was already functional with these requirements to begin with. The only task I had to do for this step was change the CID to my CID.
    b. In part one, I used key[3], sw[2:0] and HEX[3:0]. Key[3] was used as the reset button, HEX[3:0] to display values, and sw[2:0] was used as inputs to determine what to display.
    c. In part two, I used key[3], sw[2:0] and HEX[3:0]. Key[3] was used as the reset button, HEX[3:0] to display values, and sw[2:0] was used as inputs to determine what to display.
    d. In part three, I used key[3], sw[2:0] and HEX[3:0]. Key[3] was used as the reset button, HEX[3:0] to display values, and sw[2:0] was used as inputs to determine what to display.
    e. In part four, I found the sw[?] that freezes the cpu by testing all the switches until I found the right switch and finding it in lab4_de1.v.
    f. In part five, I used sw[8] in order to indicate if I want the clock speed to be 5 times faster. I put in a condition if the switch is up it will run 5 times faster.

b) Verilog Code:

Lab4_de1.v----------

```
always @ (*)
begin
        case (SW[2:0])
                3'b000: hexdata <= 16'h0580;
                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'h0580 ;
        endcase
end

clock_divider clk1Hzfrom50MHz (
                                        CLOCK_50,
                                        KEY[3],
                                        clk_1Hz,
                                        SW[8]
                                        );
```

Tc140l.v----------

```
reg         [3:0] random = 4'b0101;

// 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;

always @(posedge clock or posedge reset)  begin
        if (reset)  begin
    state       <= reset_pc;
          program_counter <= 8'b00000000;
        register_A    <= 16'b0000000000000000;
          out         <= 16'h0 ;
          random                    <= 4'b0101;
end


        // Execute the RANDOM instruction
                execute_random :
                begin
                                random <= (random << 1);
                                random[0] <= (random[3] ~^ random[2]);
                                register_A <= random;
                                state <= fetch;
                        end
                // Execute the ADDIND instruction
                        execute_addind :
                        begin
                                state <= execute_add;
                        end
                // Execute the ADDPCR instruction
                        execute_addpcr :
                        begin
                                state <= execute_add;
                        end


                execute_random:     memory_address_register <= program_counter;
                execute_addind: memory_address_register <= memory_data_register;
                execute_addpcr: memory_address_register <= instruction_register[7:0] + program_counter;
```

instruction_decoder.v----------

```
// State Encodings
parameter
                fetch               = 5'h1,
                execute_add         = 5'h3,
                execute_store       = 5'h4,
                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;


8'b01000101 :
                    state <= execute_random;
8'b01000110 :
                    state <= execute_addind;
8'b01000111         :
                    state <= execute_addpcr;
```

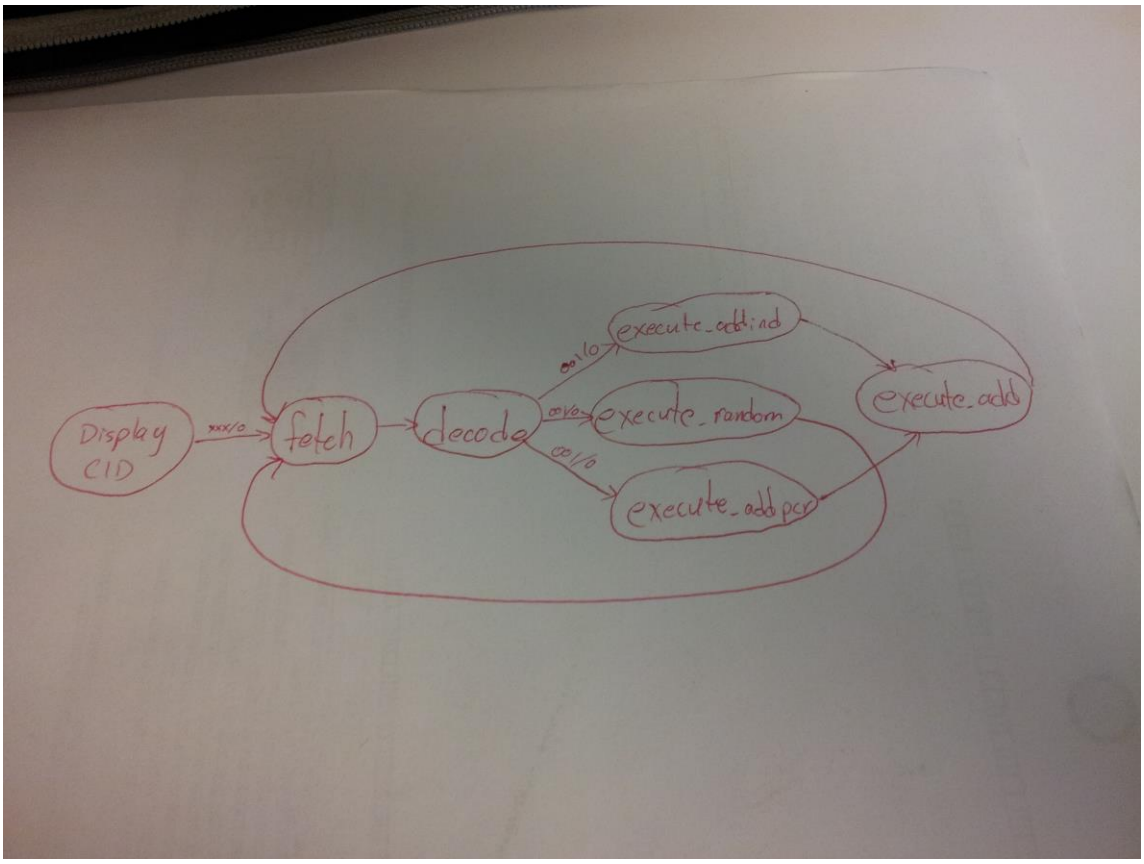
clock_divider.v----------

```
module clock_divider (clk, rst_n, clk_o, sw);

input sw;

                else
                begin
                        if (sw == 0) begin
                                div <= div + 1;
                        end
                        else if (sw == 1) begin
                                div <= div + 5;
                        end
                        en <= 0;
                end
```

c) State Diagram:



d) Compilation Report:

Flow Status	Successful - Mon Jun 02 23:39:18 2014
Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Web Edition
Revision Name	lab4_de1
Top-level Entity Name	lab4_de1
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Met timing requirements	Yes
Total logic elements	564 / 18,752 (3 %)
Total combinational functions	564 / 18,752 (3 %)
Dedicated logic registers	101 / 18,752 (< 1 %)
Total registers	101
Total pins	283 / 315 (90 %)
Total virtual pins	0
Total memory bits	4,096 / 239,616 (2 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)