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)

Self-test Report

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

(*** 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.

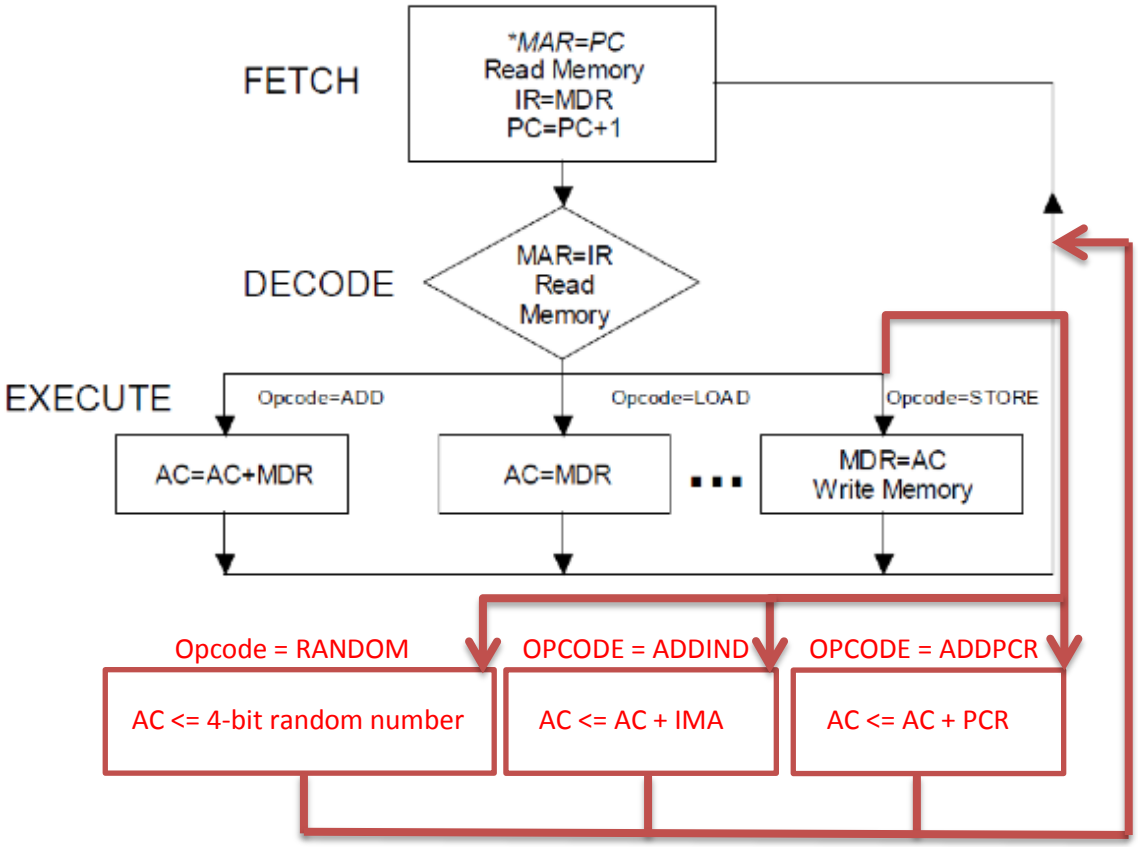
1) For the first part I made a 4 bit variable and initialized it to 5. Then I had another variable which would be used to hold the value of the XNOR operation. In execute_random I set register_A to be equal to the 4 bit variable and then shifted the variable left by one and set the least significant 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.

B.

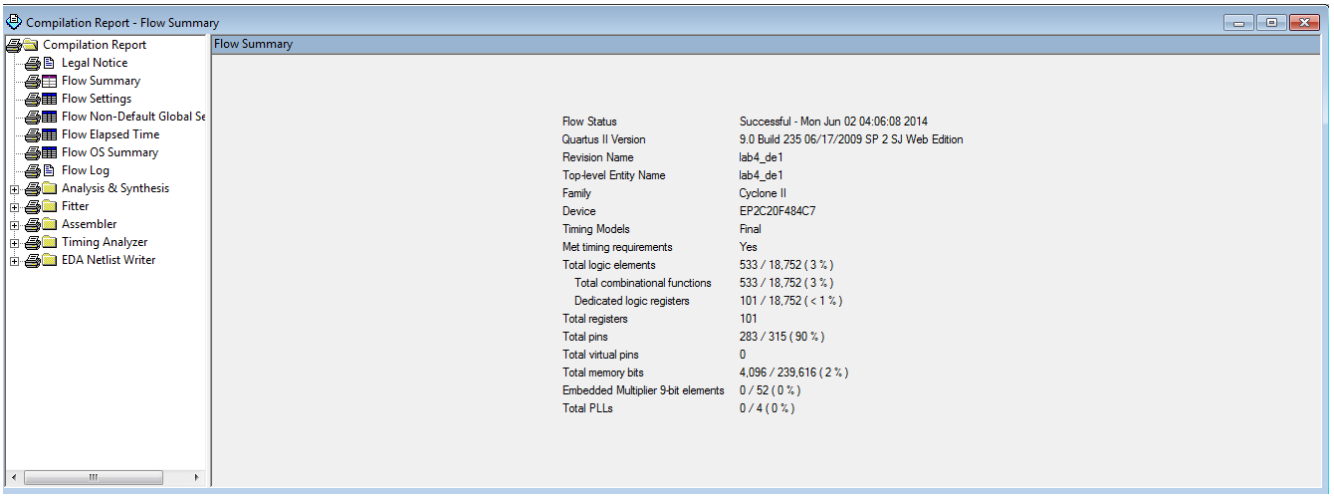
```
// 7-seg display mux
always @ (*)
begin
        case (SW[2:0])
                3'b000: hexdata <= 16'h0268;
                3'b001: hexdata <= register_A ;
                3'b010: hexdata <= program_counter ;
                3'b011: hexdata <= instruction_register ;
                3'b100: hexdata <= 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
```

C.


FETCH
*MAR=PC
Read Memory
IR=MDR
PC=PC+1
DECODE
MAR=IR
Read
Memory
EXECUTE
Opcode=ADD
Opcode=LOAD
Opcode=STORE
AC=AC+MDR
AC=MDR
MDR=AC
Write Memory
Opcode = RANDOM
OPCODE = ADDIND
OPCODE = ADDPCR
AC <= 4-bit random number
AC <= AC + IMA
AC <= AC + PCR

D.


Compilation Report - Flow Summary
Compilation Report
Legal Notice
Flow Summary
Flow Settings
Flow Non-Default Global Se
Flow Elapsed Time
Flow OS Summary
Flow Log
Analysis & Synthesis
Fitter
Assembler
Timing Analyzer
EDA Netlist Writer
Flow Summary
Flow Status Successful - Mon Jun 02 04:06:08 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 533 / 18,752 (3 %)
Total combinational functions 533 / 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 %)