ECE 368 Digital Design

Spring 2014

Lab 1 – Binary Counter & UMD ALU

Dates Performed: Wednesday, January 29th , 2014

Submission Date: Wednesday, February 12th , 2014

Team #2 :

Massarrah Tannous _____

Daniel Noyes _____

Table of Contents

Problem Statement

This lab is divided into two parts. First part is to follow the procedure given in the modelsim tutorial. This tutorial allowed creating a simple binary counter where the counter is of 4 bits and can count in both directions, up and down. The direction of the counting is based on a switch. The count is displayed through the LEDs on the Sparten 3E borad. This counter is running very fast based on the clock speed where the change in the count was not noticeable on the LEDs. As part of the extra credit, slowing down the counter is needed to solve the issue.

While for the second part of this lab, is to simulate an ALU called UMD ALU. This ALU has a top level component that connects the component. The components perform the following functions: arithmetic, logical, shift, load and store, and multiplexing. The ALU top level has four inputs: two 24 bits vectors representing the values that the operation will be performed, a 4-bits value representing the op-code of the operation, and a clock signal. The ALU unit has two outputs: 24 bits vector representing the result of the operation, and 4-bits vector representing the Condition Code Register (CCR). The bits in the CCR corresponds to negative, zero, overflow, and carry. Implementing a hardware test procedure is part of the extra credit. It uses the components available on the Sparten-3E board in order taken input and display the results. In our case, we used the push buttons and the switches for user inputs and the LCD to output the inputs and outputs.

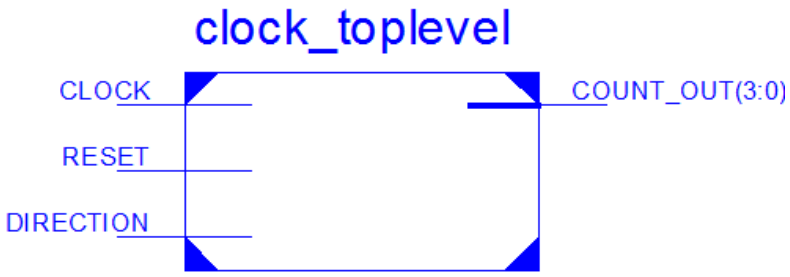
RTL Block Diagram (Hand Drawn Design)

The original designs for both the counter and the ALU are the attached engineer papers which are labeled as “Original Designs”.

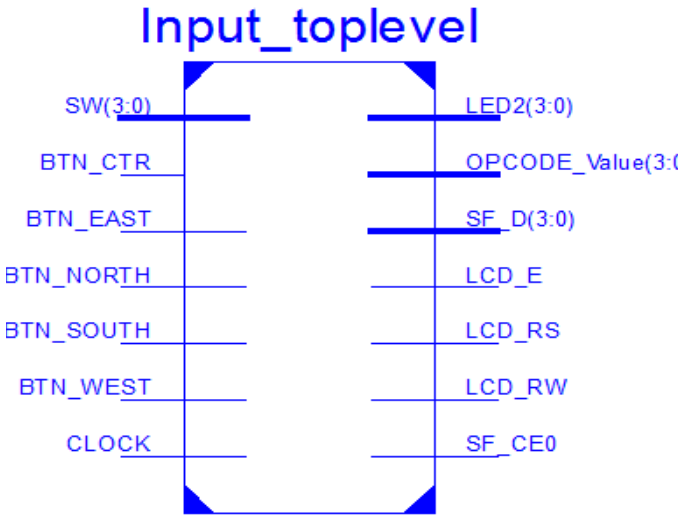
VHDL Entities Specification

The specifications of each entity used for the counter and the ALU are shown in Appendix A.

System design top level entity



Drawing 1: Top level entity for the counter.

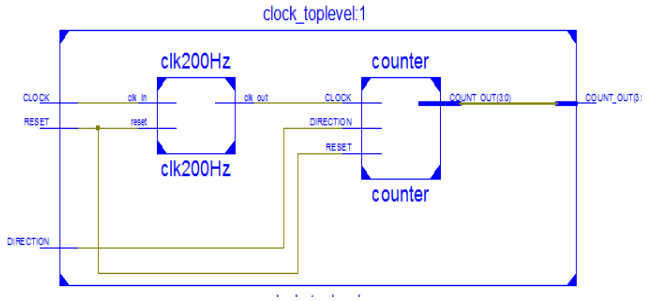


Drawing 2: Top level entity for the ALU.

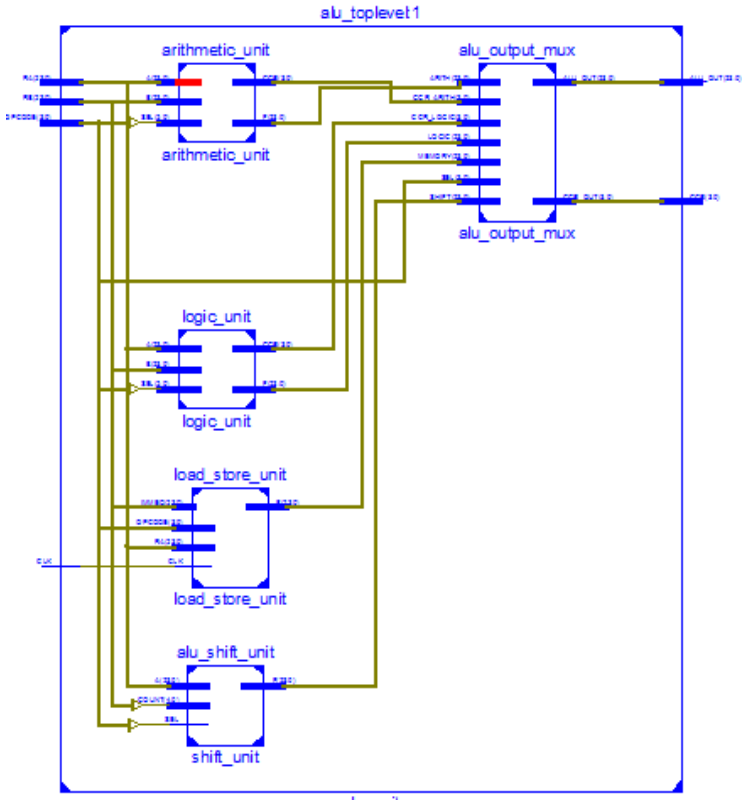
VHDL Code

The VHDL code for both the counter and ALU has been emailed as zip files to Dr. Fortier.

Generated RTL Designs with comparisons.



Drawing 3: RTL Design for the counter.



Drawing 4: RTL diagram for the ALU.

Between the various RTL diagrams there were various differences between each diagram on the second part of the lab which revolved around the ALU segment. On the first section of the code, the counter, there were no changes in the RTL block diagram from original to final. The reason was that the first section of the lab was only partially completed. In the second section, from the original we used a central input section with only outputs of A, B, and opcode. This was later changed to have the position output to control the writing of the bits and to out put the display to the LCD. Which leads to the addition of position to the LCD input. Overall the lab had example code to use in the lab which made it easier to create a RTL diagram.

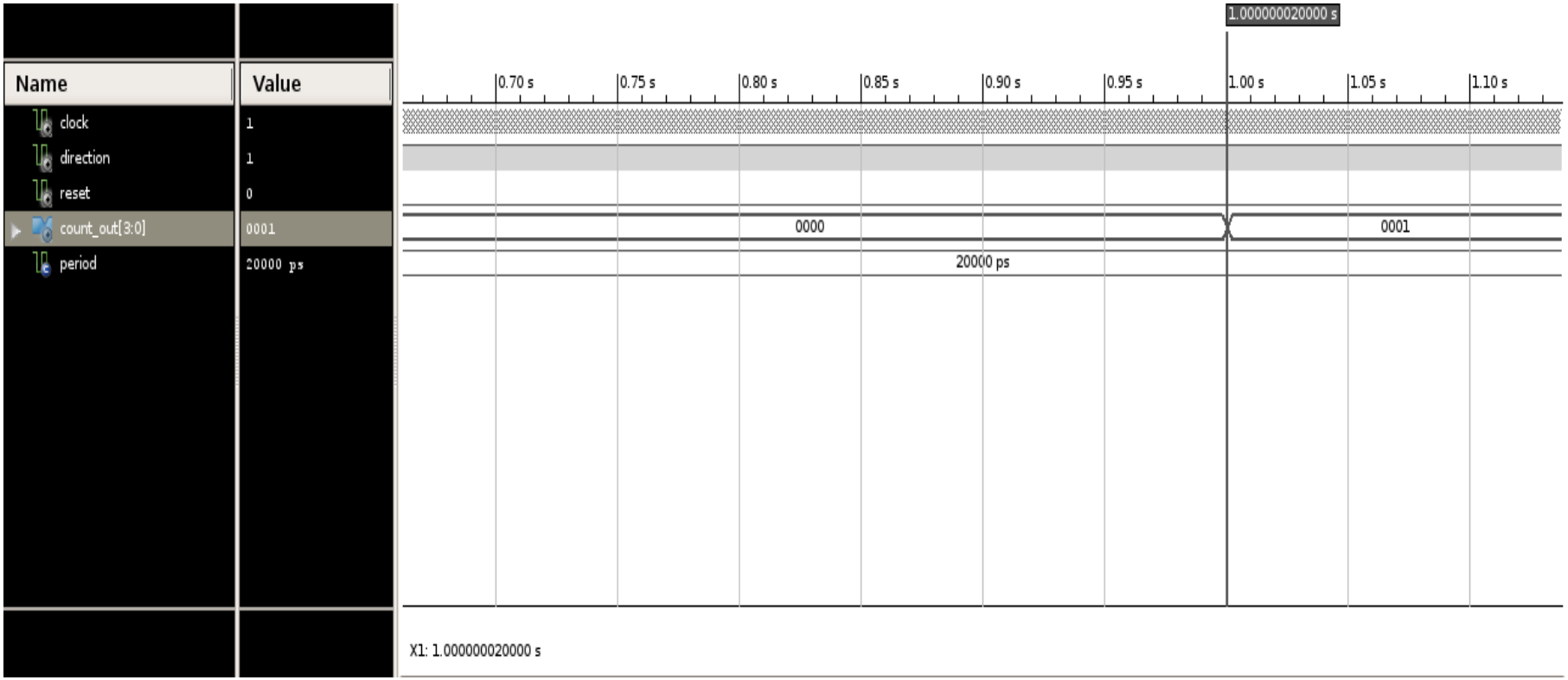
Test Plan

Overall plan (simulation and HW): For the overall plan in the simulation, the first part was just an automated counter that had to be observed to see how it changes. For the ALU, various inputs were used to test the outputs of the ALU.

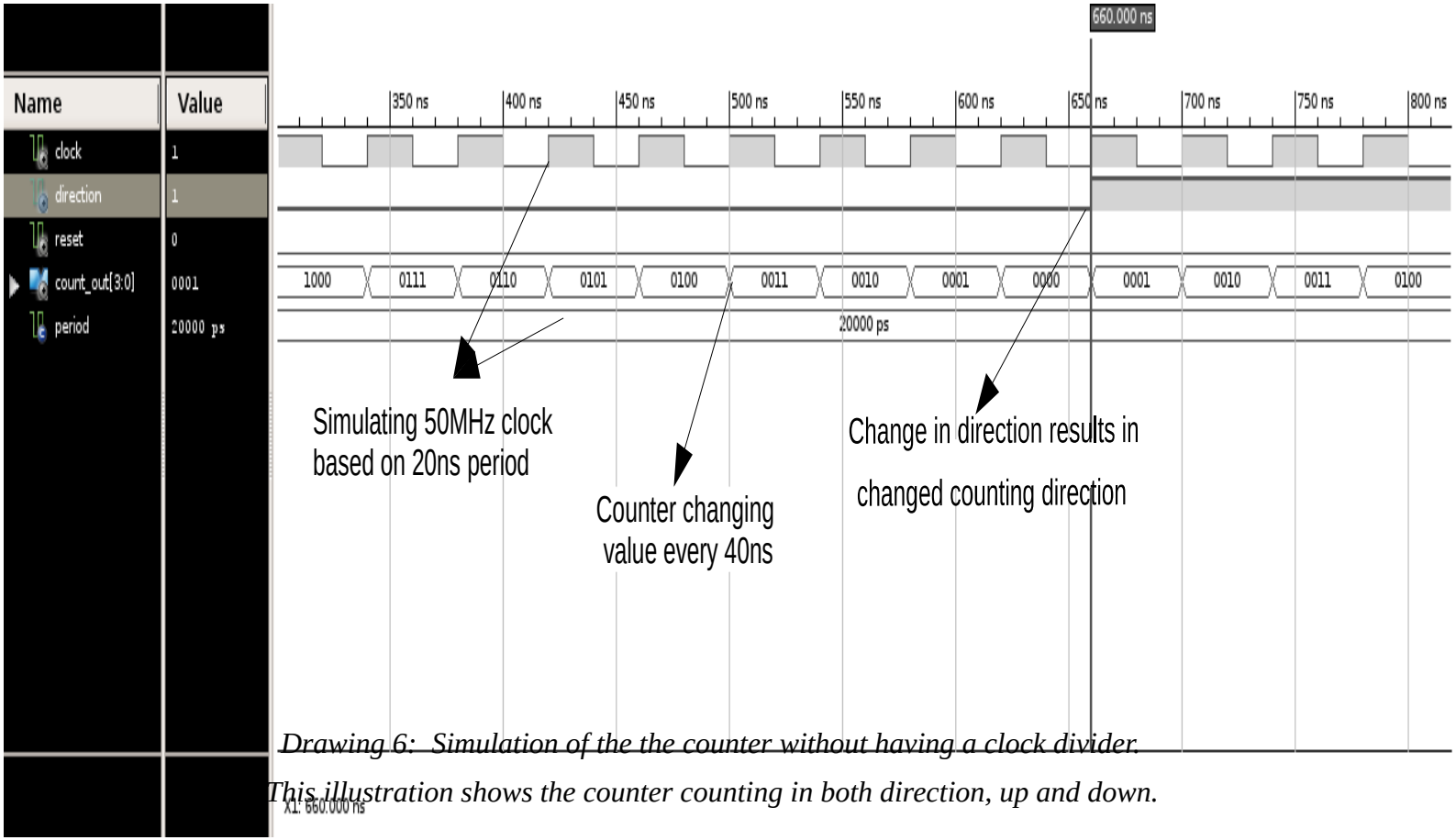
Simulation: The current simulation for the counter is the following figure 5. When you incorporate a clock divider, you will able to see the changes of the counter going at 1 second as shown in figure 6. For part two of the lab, it was to demonstrate the ALU operations/ which can be shown in figure Q). In the hardware aspect of the test plan. The only procedure for the counter was to observe the changes in the led from the counter. On the ALU the observation was done by testing various values on the ALU with the user interface of various buttons and toggle switches that will display the result on the LCD.

Testbench and simulation results: Overall the test bench and simulation results we on the mark for each function they had to achieve. For the counter it was only needed to be set at 1 second change so it can be observed on the hardware LED's. On the ALU, each value was tested on the hardware with user inputs and observed the outputs of how it changed.

Hardware test plan and procedure (what will be tested, how you will do it): For the hardware test plan and procedure's, the first part was to observer the LED changes with the counter either go up or gown based on the directional bit. And for the second part, the ALU was tested with the various inputs and the results were observed on the LCD. All the test cases and their results are shown in Table 1.

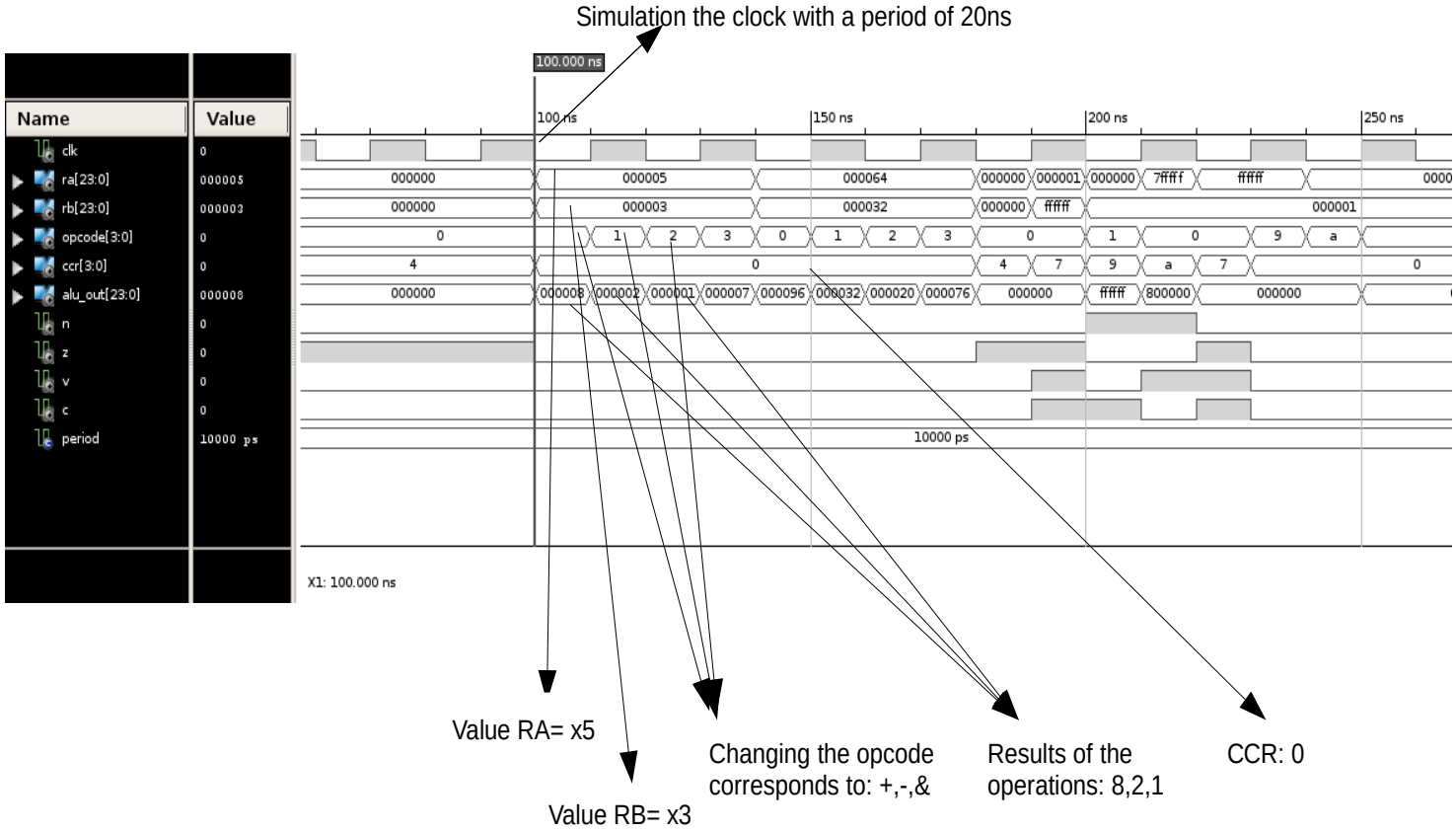


Drawing 5: Simulation of the the counter by having a clock divider.
A change of count occur every second.



Drawing 6: Simulation of the the counter without having a clock divider.

This illustration shows the counter counting in both direction, up and down.



Drawing 7: Simulation of the ALU with the change of inputs.

	Inputs			LCD Output					Test Results
	A	B	Opcode	A	B	Opcode	F	CCR	
Arithmetic Tests	x5	x3	0000	“000005”	“000003”	"++"	“000008”	---	Passed
	x5	x3	0001	“000005”	“000003”	"- -"	“000002”	---	Passed
	x5	x3	0010	“000005”	“000003”	"&&"	“000001”	---	Passed
	x5	x3	0011	“000005”	“000003”	"||"	“000007”	---	Passed
	x64	x32	0000	“000064”	“000032”	"++"	“000096”	---	Passed
	x64	x32	0001	“000064”	“000032”	"- -"	“000032”	---	Passed
	x64	x32	0010	“000064”	“000032”	"&&"	“000020”	---	Passed
	x64	x32	0011	“000064”	“000032”	"||"	“000076”	---	Passed
CCR Tests	x0	x0	0000	“000000”	“000000”	"++"	“000000”	-z--	Passed
	x1	xFFFFFF	0000	“000001”	“FFFFFF”	"++"	“000000”	-zoc	Passed
	x7FFFFF	x1	0000	“7FFFFF”	“000001”	“++”	“800000”	n-o-	Passed
	xFFFFFF	x1	0000	“FFFFFF”	“000001”	“++”	“000000”	-zoc	Passed
Mem Tests	xFFFFFF	x1	1001	“FFFFFF”	“000001”	“,,”	“000000”	----	Passed
	x16	x1	1010	“000016”	“000001”	“..”	“000000”	----	Passed
	x16	x1	1001	“000016”	“000001”	“,,”	“000016”	----	Passed

Table 1: Shows the test results of all test based on various inputs.

UCF File

The UCF for both the counter and ALU has been emailed to Dr. Fortier.

Conclusion

Through this lab we learned a lot by familiarizing ourselves with VHDL syntax and the XILINX software, and understanding the connection between components at the top level. Furthermore, we learned how to program and interact with hardware such as switches, LEDs, push buttons and LCD.

We implemented the counter with the clock divider, where the results were as expected in both the hardware and the simulation testing. By having the clock divider we were able to see the count change through the LED

The results for the ALU were as expected, where we used the switches and the push buttons for inputs and the LCD to verify the outputs. We were able to implement fully the hardware testing for the UMD ALU by inputting 24bits for A and B and 4bits for the op-code. Moreover, displaying the inputs along with the results of the operation with the CCR.

Reflection

Overall this lab was interesting and fun project. Even though this is the first time we are actually exposed to VHDL. It was a great experience of learning the concepts/syntax's of the code. At first the code for just the counter seemed daunting to use but we pushed through and learned how it works. After putting in 40+ hours on research and study, we were confident that we have the basic concepts. Most of the hours was spent on reading various tutorials like 6 and 7 which Dr. Fortier supplied in the source directory of the class drive.

One of the problems we faced at the beginning stage of the project is using the push buttons. Whenever the push button was pressed, its corresponding value was increased by more than one, where it was supposed to increase by one only. The solution to that was to add a de-bounce component that is connected to the input. Another difficulty that we faced is understanding when to use a select statement or if statements inside a process and an event.

The LCD was the most difficult part to implement but after understanding the concepts and adapt to the board. It is a vital method for debugging later on. One of the most difficult problems of the LCD was actually one of the simplest part which was just to jump to the next line on the LCD. We tried many different ways but all did not work. After a while, we found that we had to set LCD_RS to a 0 in order to change the pointer of the LCD to hex 40 of the next line which was X"C0".

With learning the code there is still a lot that can be done in order to improve the functions we coded. One of the main features that we would like to incorporate was a central lookup table that the LCD relies on and improve the decoder of the LCD. The next was to incorporate more priority encoders just to remove excess used LUTS on the system.

ECE 368
Digital Design
Spring 2014

Lab 1 – Binary Counter & UMD ALU

Design Layout
Section

Counter – clock_toplevel

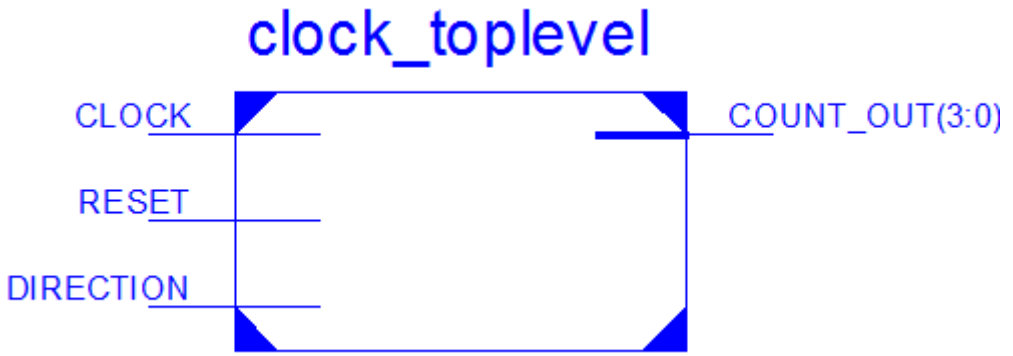
Inputs:

Clock <STD_LOGIC>,
Reset <STD_LOGIC>,
Direction <STD_LOGIC>

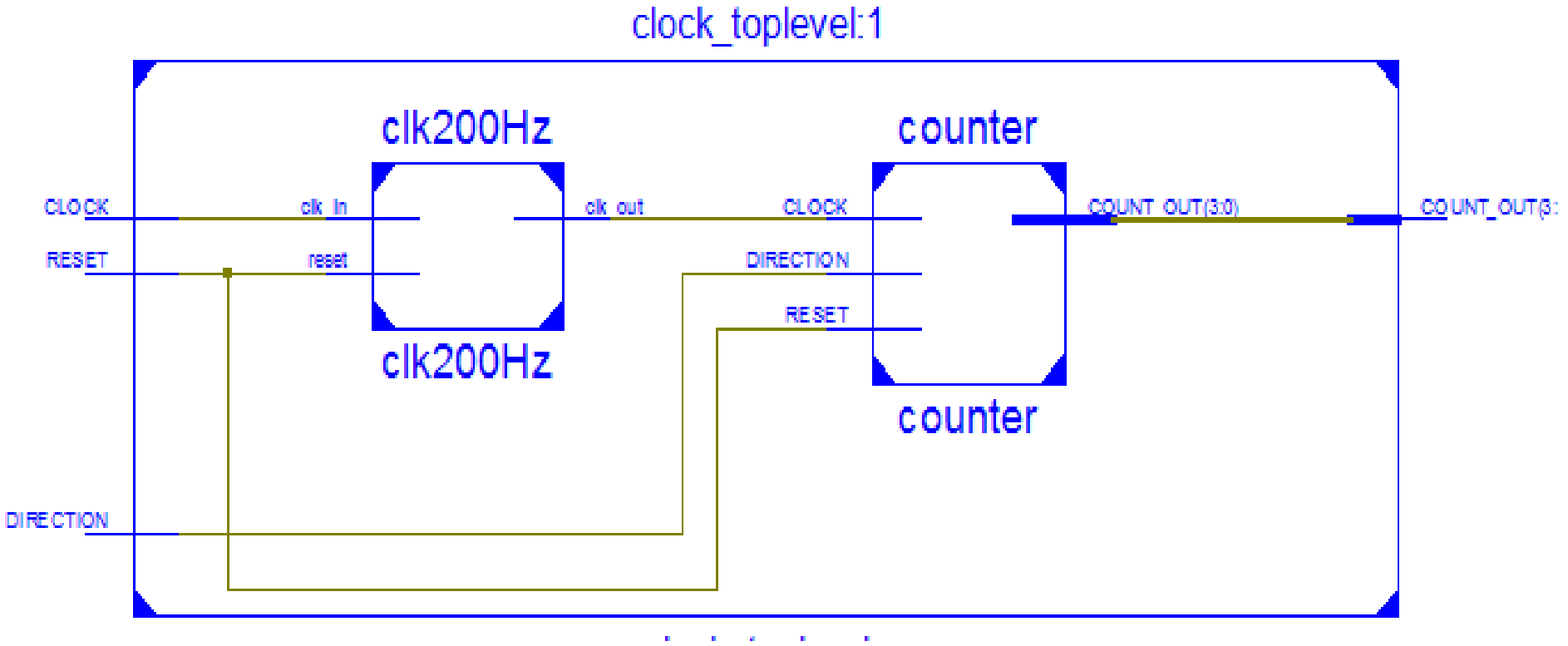
Outputs:

Count_out <STD_LOGIC_VECTOR (3 downto 0)>

Functionality: This is the top level for the counter. Based on a clock signal, the counter gets incremented and displayed through the LEDs as output. It connects its componets: counter and clock200Hz.



Counter – clock_toplevel



Counter – clock200Hz

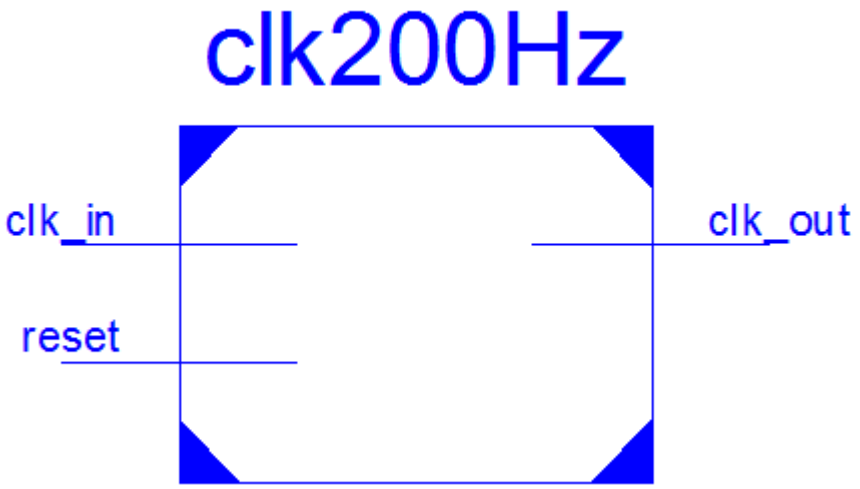
Inputs:

clk_in <STD_LOGIC> ,
reset <STD_LOGIC>,

Outputs:

clk_out <STD_LOGIC>

Functionality: This component decreases the input frequency. It increments an integer value on the rising edge of the clock until the value reaches 25000000. Once the value has reached, a high value signal is passed. This scales a 50MHz clock down to 2Hz.



Counter – clock_toplevel

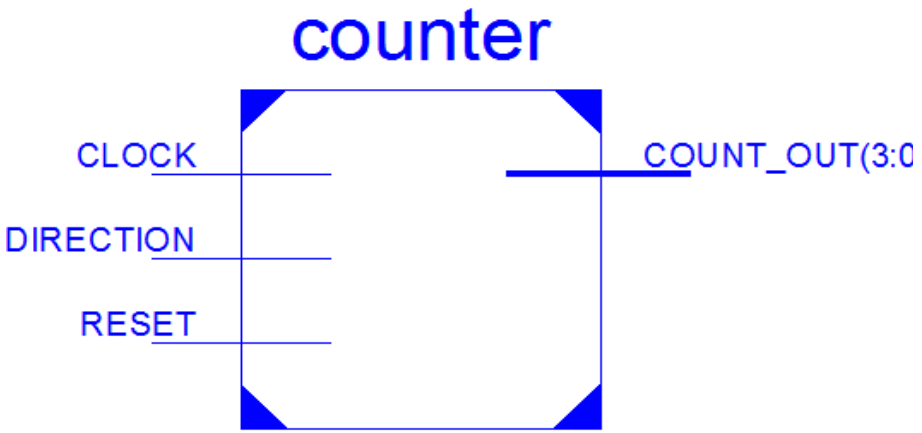
Inputs:

Clock <STD_LOGIC> ,
Reset <STD_LOGIC>,
Direction <STD_LOGIC>

Outputs:

Count_out <STD_LOGIC_VECTOR (3 downto 0)>

Functionality: Starting from 0 4-bit value gets incremented if the is high and decremented if it low.



ALU – input_toplevel

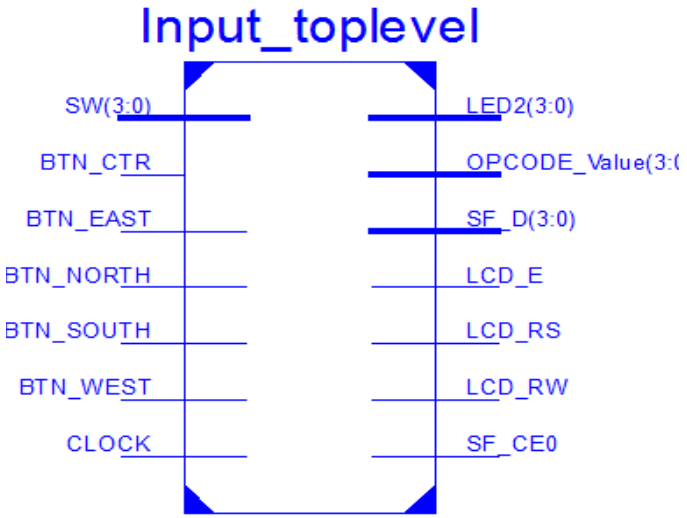
Inputs:

SW <STD_LOGIC_VECTOR (3 downto 0) >
BTN_CTR, BTN_EAST, BTN_NORTH, BTN_SOUTH, BTN_WEST <STD_LOGIC>,
Clock <STD_LOGIC>

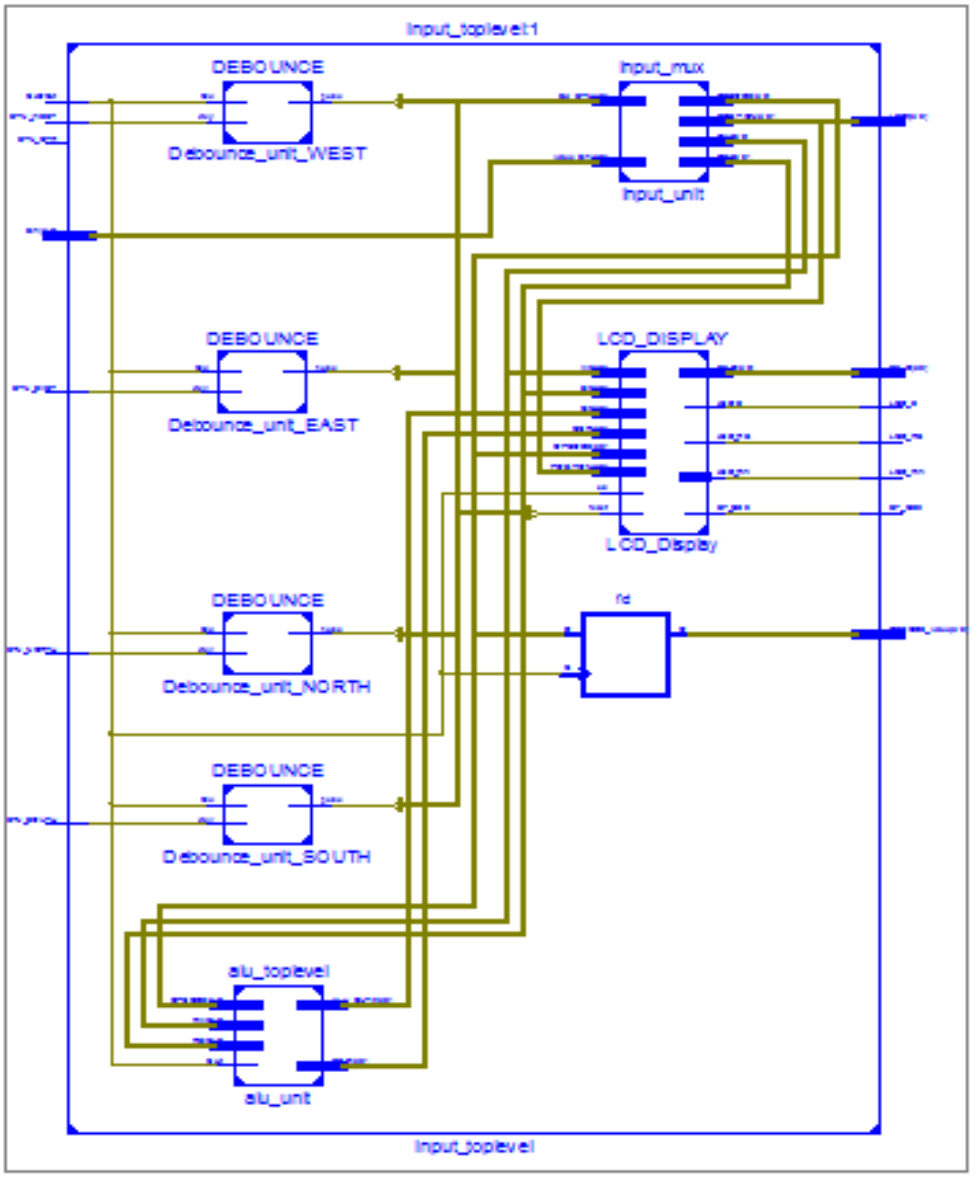
Outputs:

LED2, OPCODE_Value, SF_D <STD_LOGIC_VECTOR (3 downto 0) >,
LCD_E, LCD_RS, LCD_RW, SF_CE0 <STD_LOGIC>

Functionality: This is the top level for the ALU. It connects its components and passes the input data to its components. Its inputs are connected to the switches, push button and the clock. The output of this component is connected to the LCD. Any other outputs is for debugging purposes



ALU – input_toplevel



ALU – Debounce

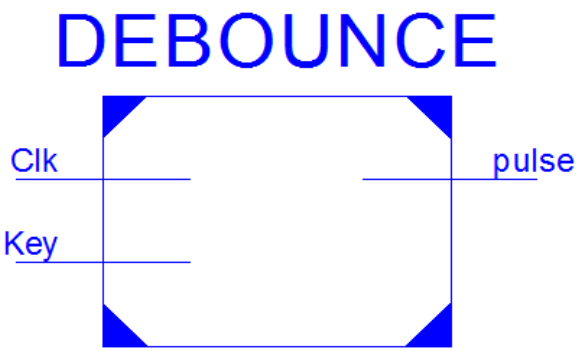
Inputs:

Clk, Key <STD_LOGIC>

Outputs:

pulse <STD_LOGIC>

Functionality: It is used for the purposes of push button de-boucing. An integer gets incremented 1000000 so that the results gets stable.



ALU – input_mux

Inputs:

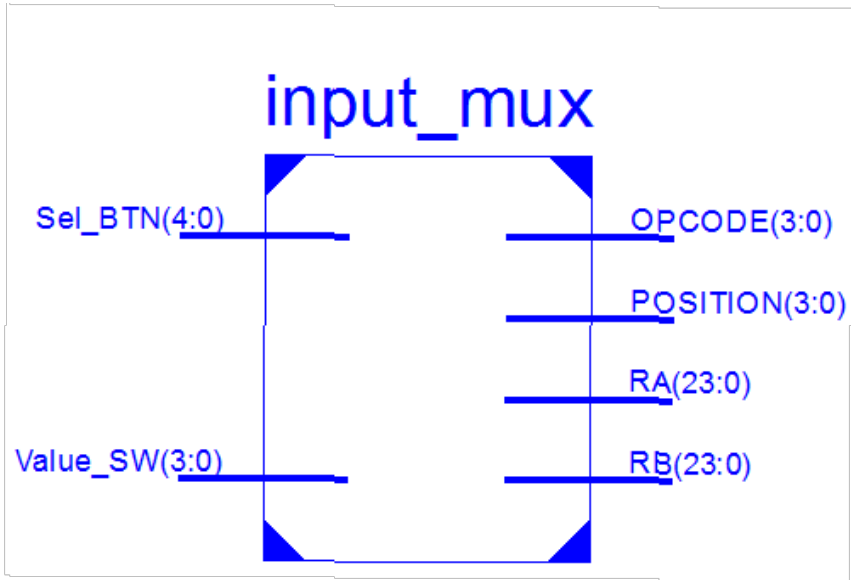
Sel_BTN, Value_SW <STD_LOGIC_VECTOR (3 downto 0) >

Outputs:

Position, Opcode <STD_LOGIC_VECTOR (3 downto 0) >

RA, RB <STD_LOGIC_VECTOR (23 downto 0) >

Functionality: This component is used to convert the switch input based on the push buttons to the corresponding value. Sel_BTN(0) for RA. Sel_BTN(1) for RB input . Sel_BTN(2) for position. Sel_BTN(3) for opcode input.



ALU – LCD_DISPLAY

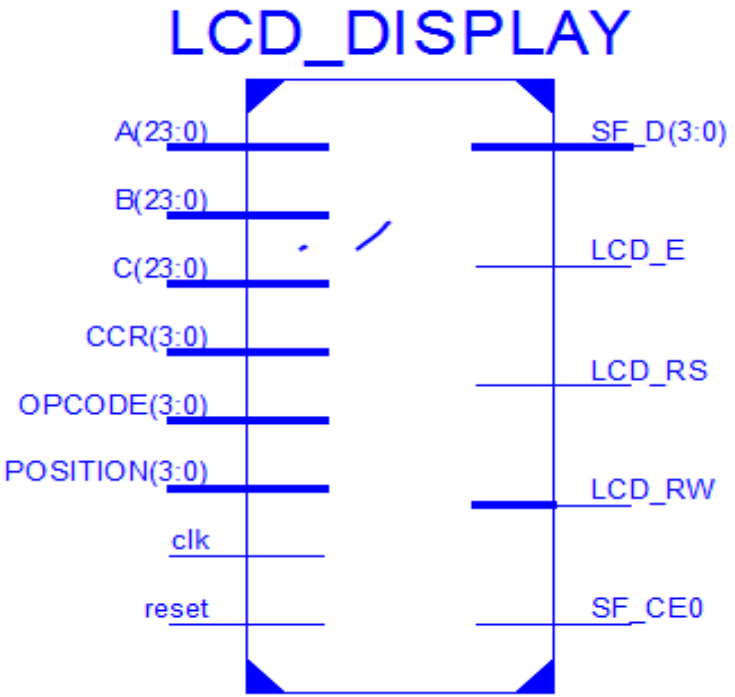
Inputs:

Position, Opcode, CCR <STD_LOGIC_VECTOR (3 downto 0) >
A, B, C, <STD_LOGIC_VECTOR (23 downto 0) >
Clk, reset <STD_VECTOR>

Outputs:

SF_D <STD_LOGIC_VECTOR (3 downto 0) >,
LCD_E, LCD_RS, LCD_RW, SF_CE0 <STD_LOGIC>

Functionality: This component is used to display the A, B, Opcode, CCR, and the result of the operation on the LCD



ALU – alu_toplevel

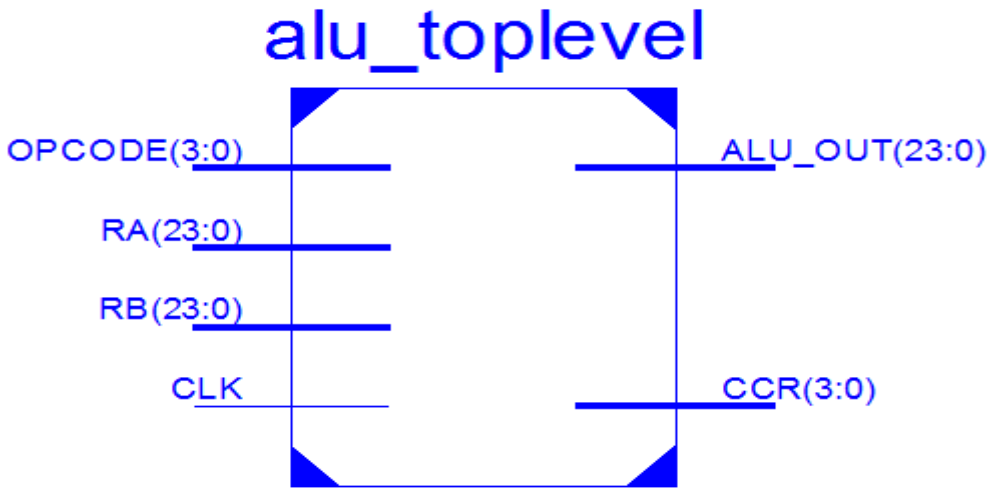
Inputs:

OPCODE, Value_SW <STD_LOGIC_VECTOR (3 downto 0) >
RA, RB <STD_LOGIC_VECTOR (23 downto 0) >
Clk <STD_LOGIC>

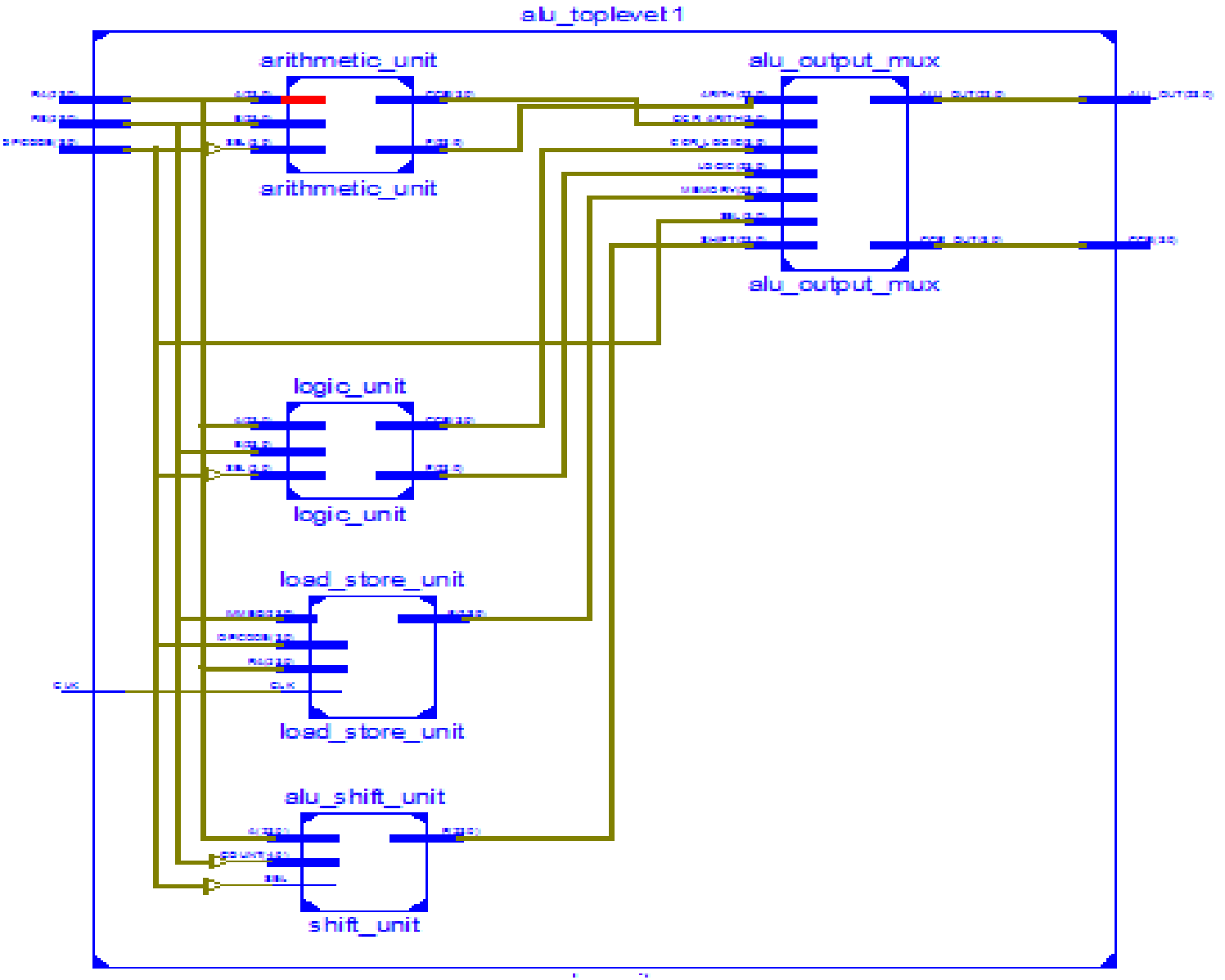
Outputs:

CCR <STD_LOGIC_VECTOR (3 downto 0) >
ALU_OUT <STD_LOGIC_VECTOR (23 downto 0) >

Functionality: It performs operation on RA and RB based on the opcode. The result of the operation is output in the ALU_OUT and the CCR (N,Z,V,C).



ALU – alu_toplevel



alu_toplevel – arithmetic-unit

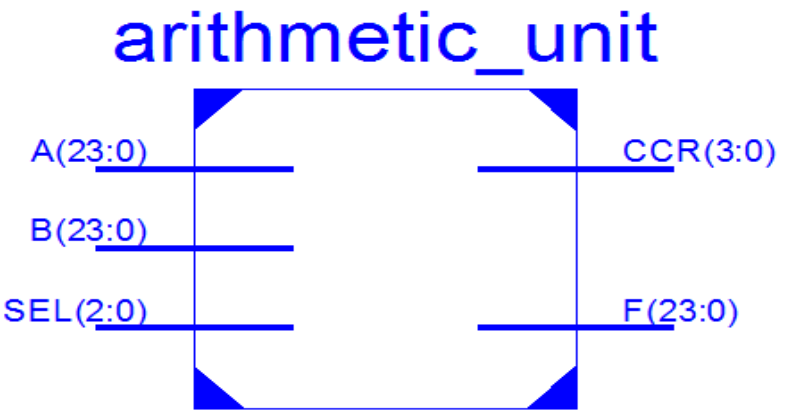
Inputs:

OPCODE <STD_LOGIC_VECTOR (3 downto 0) >
RA, RB <STD_LOGIC_VECTOR (23 downto 0) >
SEL <STD_LOGIC_VECTOR (2 downto 0)>

Outputs:

CCR <STD_LOGIC_VECTOR (3 downto 0) >
ALU_OUT <STD_LOGIC_VECTOR (23 downto 0) >

Functionality: It performs add/subtract on RA and RB based on the opcode. The result of the operation is output in the ALU_OUT and the CCR (N,Z,V,C).



alu_toplevel – logical-unit

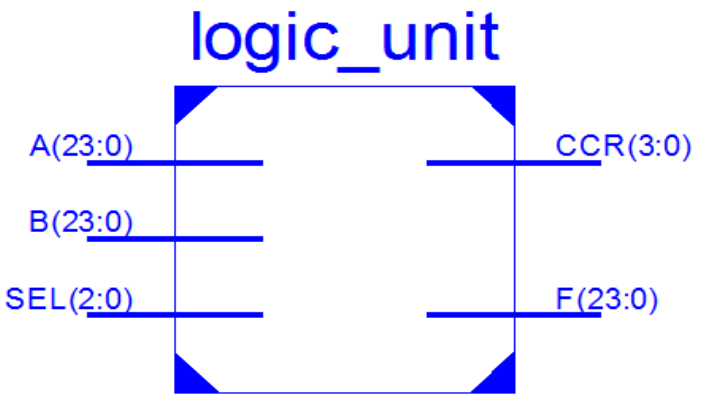
Inputs:

A, B <STD_LOGIC_VECTOR (23 downto 0) >
SEL <STD_LOGIC_VECTOR (2 downto 0)>

Outputs:

CCR <STD_LOGIC_VECTOR (3 downto 0) >
F <STD_LOGIC_VECTOR (23 downto 0) >

Functionality: It performs the logical operation: AND, OR, CMP, ANDI subtract on RA and RB based on the opcode. The result of the operation is output in the ALU_OUT and the CCR (N,Z,V,C).



alu_toplevel – load_store_unit

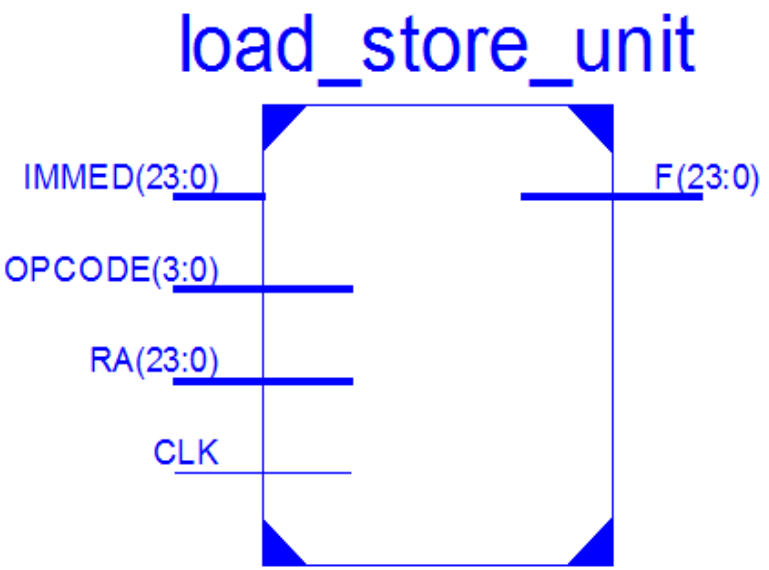
Inputs:

OPCODE <STD_LOGIC_VECTOR (3 downto 0) >
RA, IMMED <STD_LOGIC_VECTOR (23 downto 0) >
CLK <STD_LOGIC>

Outputs:

F <STD_LOGIC_VECTOR (23 downto 0) >

Functionality: It performs the performs reading/writing 'A' to memory 'IMMED'.



alu_toplevel – load_store_unit

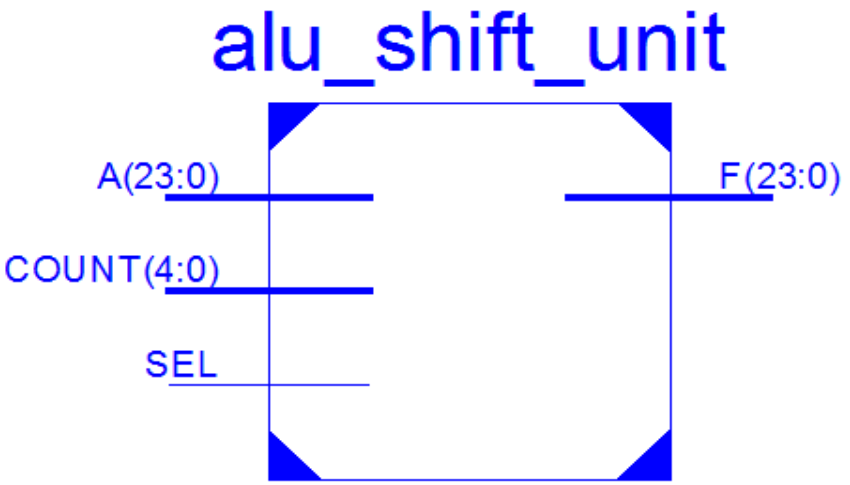
Inputs:

COUNT <STD_LOGIC_VECTOR (4 downto 0) >
A <STD_LOGIC_VECTOR (23 downto 0) >
SEL <STD_LOGIC>

Outputs:

F <STD_LOGIC_VECTOR (23 downto 0) >

Functionality: It shift the input A to the left/right based on the sel and it shifts by the amount in the count. The result is outputted to F.



ECE 368
Digital Design
Spring 2014

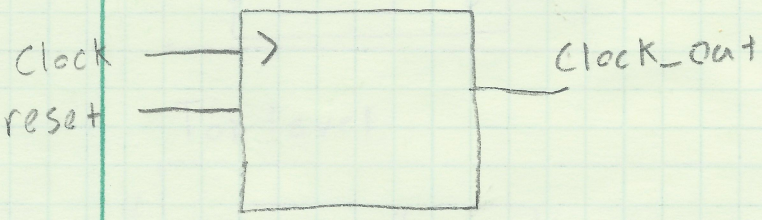
Lab 1 – Binary Counter & UMD ALU

Original Design
Section

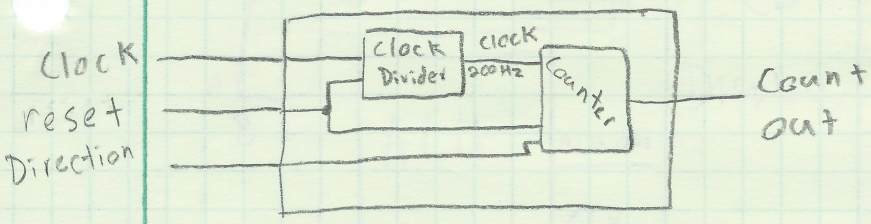
EE2368
Lab#1 RTL Block diagram | Original Design

Counter

Clock divider to 200 Hz

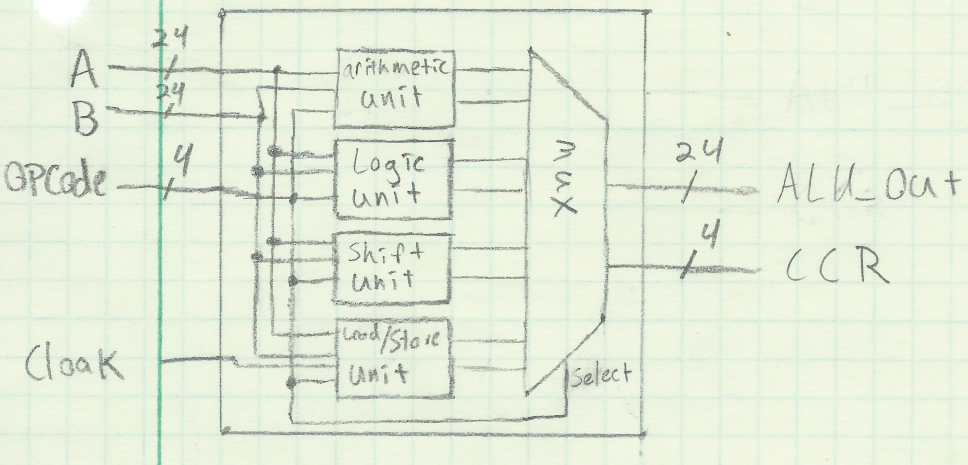


Clock_Toplevel

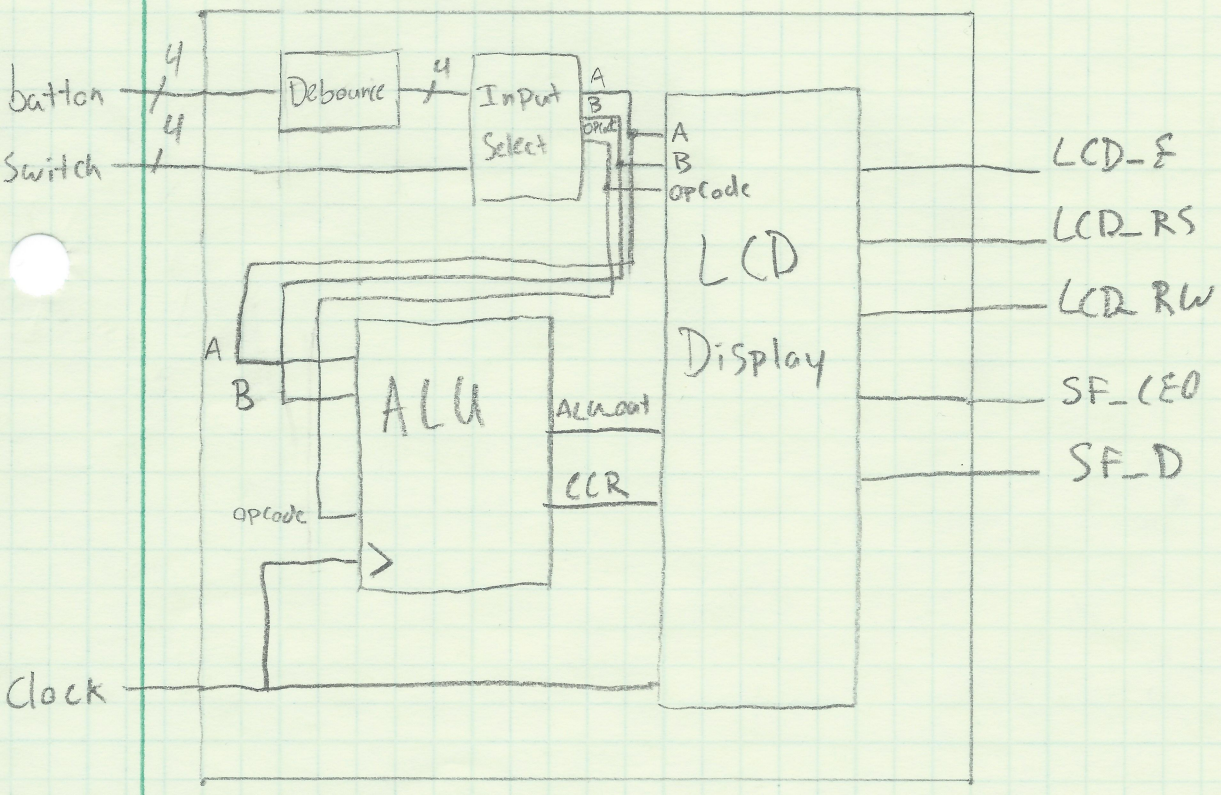


ECE 368
Lab #1 RTL block Diagram | Original Design

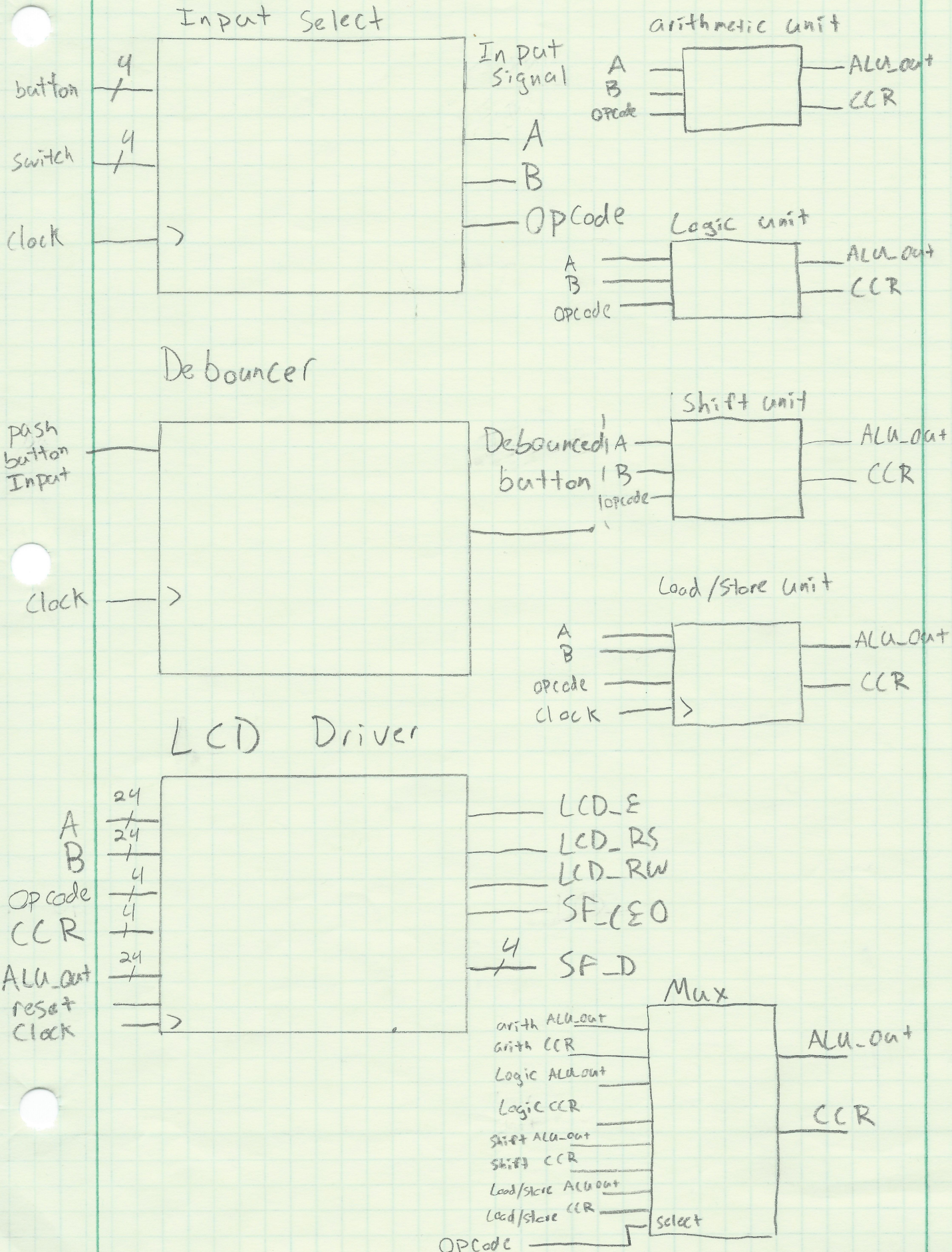
ALU



Top level



ECE 368
Lab # 1 RTL Block Diagrams | Original Design



ECE 368
Digital Design
Spring 2014

Lab 1 – Binary Counter & UMD ALU

Counter Code
Section

```
----------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
--
-- Create Date:    SPRING 2014
-- Module Name:    UMD_ALU_FPU
-- Project Name:   UMD-RISC 24
-- Target Devices: Spartan-3E
-- Tool versions:        Xilinx ISE 14.7
-- Description: Clock Divider
--      Lower the Clock frequency from
--      50 Mhz to 2 hz
--      50Mhz = 50,000,000/25,000,000 = 2 Hz
--              2Hz/2 => 1 second
----------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity clk2Hz is
    Port ( clk_in : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           clk_out : out  STD_LOGIC);
end clk2Hz;

architecture Behavioral of clk2Hz is
        signal temporal: STD_LOGIC:='0';
   signal counter : integer range 0 to 25000000 := 0;

begin
    frequency_divider: process (reset, clk_in) begin
        if (reset = '1') then
            temporal <= '0';
            counter <= 0;
        elsif rising_edge(clk_in) then
            if (counter = 25000000) then
                                        if(temporal='0') then
                                                temporal <= '1';
                                         else
                                                temporal <= '0';
                                        end if;
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

    clk_out <= temporal;
end Behavioral;
```

```
######################################
## School: University of Massachusetts Dartmouth
## Department: Computer and Electrical Engineering
## Class: ECE 368 Digital Design
## Engineer: Daniel Noyes
##                              Massarrah Tannous
##
##
## Create Date:    SPRING 2014
## Module Name:    UMD_ALU_FPU
## Project Name:   UMD-RISC 24
## Target Devices: Spartan-3E
## Tool versions:        Xilinx ISE 14.7
## Description: Clock toplevel user constraints
##      Locations of each pin output
######################################
#Created by Constraints Editor (xc3s500e-fg320-4) - 2014/02/03

NET "CLOCK" TNM_NET = "CLOCK";
TIMESPEC TS_CLOCK = PERIOD "CLOCK" 40 ns HIGH 50 %;
OFFSET = IN 10 ns BEFORE "CLOCK";
OFFSET = OUT 10 ns AFTER "CLOCK";

# PlanAhead Generated physical constraints

NET "COUNT_OUT[3]" LOC = F11;
NET "COUNT_OUT[2]" LOC = E11;
NET "COUNT_OUT[1]" LOC = E12;
NET "COUNT_OUT[0]" LOC = F12;
NET "CLOCK" LOC = C9;
NET "DIRECTION" LOC = L13;
NET "RESET" LOC = D18;
```

```
----------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
--
-- Create Date:    SPRING 2014
-- Module Name:    UMD_ALU_FPU
-- Project Name:   UMD-RISC 24
-- Target Devices: Spartan-3E
-- Tool versions:        Xilinx ISE 14.7
-- Description: Clock toplevel
--      Top level diagram of the Clock
--
----------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.all;

entity clock_toplevel is
    Port ( CLOCK : in  STD_LOGIC;       -- 50 MHz Oscillator
           RESET : in  STD_LOGIC;
           DIRECTION : in  STD_LOGIC;
           COUNT_OUT : out  STD_LOGIC_VECTOR (3 downto 0));
end clock_toplevel;

architecture Structural of clock_toplevel is
        -- CLOCK
        signal CLOCK_DIVIDER    : STD_LOGIC  := '0'; -- Clock2Hz Output
        --signal COUNT  : STD_LOGIC_VECTOR (3 downto 0) := (OTHERS => '0'); -- Counter Output
begin

        ----- Structural Components: -----
        clk2Hz: entity work.clk2Hz
        port map(clk_in => CLOCK,                                                       -- clk2Hz input
                                reset           =>
RESET,                                                          -- clk2Hz input
                                clk_out         => CLOCK_DIVIDER                                -- clk2Hz
output
                                );

        counter: entity work.counter
        port map(CLOCK          => CLOCK_DIVIDER,                               -- From the clk2, counter
input
                                DIRECTION       => DIRECTION,                                   --
counter input
                                RESET           =>      RESET,
                                COUNT_OUT       => COUNT_OUT                                    -- Route
the output of clock
                                );

        ----- End Structural Components -----


end Structural;
```

```
----------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
--
-- Create Date:    SPRING 2014
-- Module Name:    UMD_ALU_FPU
-- Project Name:   UMD-RISC 24
-- Target Devices: Spartan-3E
-- Tool versions:        Xilinx ISE 14.7
-- Description: Clock Divider
--      Lower the Clock frequency from
--      50 Mhz to 2 hz
--      50Mhz = 50,000,000/25,000,000 = 2 Hz
--              2Hz/2 => 1 second
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
----------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY counter_tb IS
END counter_tb;

ARCHITECTURE behavior OF counter_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT clock_toplevel
    PORT(
         CLOCK : IN  std_logic;
         DIRECTION : IN  std_logic;
         RESET: IN STD_LOGIC;
                     COUNT_OUT : OUT  std_logic_vector(3 downto 0)
                );
    END COMPONENT;


   --Inputs
   signal CLOCK : std_logic := '0';
   signal DIRECTION : std_logic := '0';
        signal RESET : std_logic := '0';
        --Outputs
   signal COUNT_OUT : std_logic_vector(3 downto 0);

        --Constants
        constant period : time := 20 ns;

BEGIN
        --Instantiate the unit under test (UUT)
        uut: clock_toplevel PORT MAP(
        CLOCK => CLOCK,
```

```
        DIRECTION => DIRECTION,
        RESET =>RESET,
        COUNT_OUT => COUNT_OUT
        );
        --GENERATE The clock
        m50MHZ_Clock: process
        begin
                clock <= '0'; wait for period;
                clock <= '1'; wait for period;
        end process m50MHZ_Clock;

        --STIMULUS Process
        tb : process
        begin
                wait for 20 ns;
                report "Start counter Test Bench" severity NOTE;
                DIRECTION <='0';
                wait for 40 ns; assert COUNT_OUT ="1111" report "ERROR: expected 1111 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1110" report "ERROR: expected 1110 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1101" report "ERROR: expected 1101 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1100" report "ERROR: expected 1100 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1011" report "ERROR: expected 1011 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1010" report "ERROR: expected 1010 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1001" report "ERROR: expected 1001 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1000" report "ERROR: expected 1000 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0111" report "ERROR: expected 0111 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0110" report "ERROR: expected 0110 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0101" report "ERROR: expected 0101 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0100" report "ERROR: expected 0100 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0011" report "ERROR: expected 0011 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0010" report "ERROR: expected 0010 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0001" report "ERROR: expected 0001 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0000" report "ERROR: expected 0000 for COUNT_OUT"
severity ERROR;
                DIRECTION <= '1';
                wait for 40 ns; assert COUNT_OUT ="0001" report "ERROR: expected 0001 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0010" report "ERROR: expected 0010 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0011" report "ERROR: expected 0011 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0100" report "ERROR: expected 0100 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0101" report "ERROR: expected 0101 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0110" report "ERROR: expected 0110 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="0111" report "ERROR: expected 0111 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1000" report "ERROR: expected 1000 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1001" report "ERROR: expected 1001 for COUNT_OUT"
```

```
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1010" report "ERROR: expected 1010 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1011" report "ERROR: expected 1011 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1100" report "ERROR: expected 1100 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1101" report "ERROR: expected 1101 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1110" report "ERROR: expected 1110 for COUNT_OUT"
severity ERROR;
                wait for 40 ns; assert COUNT_OUT ="1111" report "ERROR: expected 1111 for COUNT_OUT"
severity ERROR;

                report "Counter Test done." severity note;
                wait; -- END Test
        end process;

END;
```

```
----------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
--
-- Create Date:    SPRING 2014
-- Module Name:    UMD_ALU_FPU
-- Project Name:   UMD-RISC 24
-- Target Devices: Spartan-3E
-- Tool versions:        Xilinx ISE 14.7
-- Description: Counter
--      Will increase the counter(output) ever time
--              the clock does a rising action
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter is
    Port ( CLOCK : in  STD_LOGIC;
           DIRECTION : in  STD_LOGIC;
           RESET : in STD_LOGIC;
                        COUNT_OUT : out  STD_LOGIC_VECTOR (3 downto 0));

end counter;

architecture Behavioral of counter is

        signal count_int : std_logic_vector(0 to 3) := "0000";
        begin
                process (CLOCK, RESET)
                begin
                        if (RESET = '1') then
                                        count_int <= "0000";
                        elsif CLOCK = '1' and CLOCK'event then
                                if DIRECTION = '1' then
                                        count_int <= count_int + 1;
                                else
                                        count_int <= count_int - 1;
                                end if;
                        end if;
                end process;
        COUNT_OUT <= count_int;
end Behavioral;
```

ECE 368
Digital Design
Spring 2014

Lab 1 – Binary Counter & UMD ALU

ALU Code
Section

```
----------------------------------------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
--
-- Create Date:    SPRING 2014
-- Module Name:    UMD_ALU_FPU
-- Project Name:   UMD-RISC 24
-- Target Devices: Spartan-3E
-- Tool versions:        Xilinx ISE 14.7
-- Description: Arithmetic Logic Unit - Arithmetic Unit.
--      Performs all arithmetic operations.
--      Performs all shift operations.
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity arithmetic_unit is

        Port (  A       : in    STD_LOGIC_VECTOR (23 downto 0); -- Input 1
                        B     : in    STD_LOGIC_VECTOR (23 downto 0); -- Input 2
           SEL : in  STD_LOGIC_VECTOR (2 downto 0);     -- Select ALU Operation
                        CCR : out STD_LOGIC_VECTOR (3 downto 0);      -- Condition Codes (N,Z,V,C)
           F : out  STD_LOGIC_VECTOR  (23 downto 0));   -- Output

end arithmetic_unit;

architecture Combinational of arithmetic_unit is
        signal A2, B2: STD_LOGIC_VECTOR (24 downto 0) := (OTHERS => '0'); -- Expanded 25 bit inputs
        signal arith: STD_LOGIC_VECTOR (24 downto 0) := (OTHERS => '0');        -- Expanded 25 bit
arithmetic register to hold result
begin

        -- Expand to 25 bit inputs to monitor carry and overflow
        A2 <= '0' & A;
        B2 <= '0' & B;

        ----- Arithmetic Unit: -----

        -- Perform 3 arithmetic operations with expanded 25-bit inputs.
        --      Calculates in two's complement.
        with sel select
                arith <=        A2 + B2                                 when "000",             -- ADD
REG A, REG B
                                        A2 - B2                                 when "001",             --
SUB     REG A, REG B
                                        A2 + B2                                 when OTHERS;    -- ADDI
        REG A, IMMED
        ----- End Arithmetic Unit -----

        ----- Set Condition Code Register (N,Z,V,C): -----
        ccr(3) <= arith(23);
                                                                                                -- N - Negative
        ccr(2) <= '1' when arith(23 downto 0)=x"0000" else '0'; -- Z - Zero
        ccr(1) <= arith(24) xor arith(23);                                                      --
V - Overflow
        ccr(0) <= arith(24);
                                                                                                -- C - Carry/
Borrow
        ----- End Set Condition Code Register -----

        -- Trim to 24-bits to output arithmetic result
```

```
        F <= arith(23 downto 0);

end Combinational;
```

```
----------------------------------------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
--
-- Create Date:    SPRING 2014
-- Module Name:    UMD_LOGIC_FPU
-- Project Name:         UMD-RISC 24
-- Target Devices: Spartan-3E
-- Tool versions:        Xilinx ISE 9.2i
-- Description: Arithmetic Logic Unit - Logic Unit.
--      Performs all logic operations.
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity logic_unit is

        Port (  A       : in    STD_LOGIC_VECTOR (23 downto 0); -- Input 1
                        B     : in    STD_LOGIC_VECTOR (23 downto 0); -- Input 2
           SEL : in  STD_LOGIC_VECTOR (2 downto 0);     -- Select Logic Operation
                        CCR : out STD_LOGIC_VECTOR (3 downto 0);      -- Condition Codes (N,Z,V,C)
           F : out  STD_LOGIC_VECTOR  (23 downto 0));   -- Output

end logic_unit;

architecture Combinational of logic_unit is
        signal ccr_cmp: STD_LOGIC_VECTOR (3 downto 0) := (OTHERS => '0'); -- CCR Compare
begin

        ----- Logic Unit: -----
        with sel select
                F <=    A and B         when "010",     -- AND  REG A, REG B
                                A or B          when "011",     -- OR   REG A, REG B
                                x"000000"       when "100",     -- CMP  REG A, REG B
                                A and B         when OTHERS;-- ANDI     REG A, IMMED
        ----- End Logic Unit -----

        -- Set CCR for CMP operation
        ccr_cmp(3) <= '1' when a<b else '0';    -- N when s<r
        ccr_cmp(2) <= '1' when a=b else '0';    -- Z when s=r

        -- Choose CCR output
        with sel select
                ccr <= ccr_cmp when "100",      -- CCR for CMP
                                "0000" when OTHERS;    -- All flags cleared for other logic operations

end Combinational;
```

```
----------------------------------------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
-- Create Date:    SPRING 2014
-- Module Name:    UMD_MUX_FPU
-- Project Name:         UMD-RISC 24
-- Target Devices: Spartan-3E
-- Tool versions:        Xilinx ISE 14.7
-- Description: ALU Output Multiplexer.
--      Selects ALU output based on current instruction.
--
--              Sets condition code register output for each instruction.
--              Operations that do not set any condition codes clear all
--              flags for simplicity. You may want to change this to suit
--              your specific design.
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity alu_output_mux is

        Port (  SEL                     : in STD_LOGIC_VECTOR   (3 downto 0);   -- Mux Input
                                ARITH           : in STD_LOGIC_VECTOR   (23 downto 0);  -- ALU Arith Input
                                LOGIC           : in STD_LOGIC_VECTOR   (23 downto 0);  -- ALU Logic Input
                                SHIFT           : in STD_LOGIC_VECTOR   (23 downto 0);  -- ALU Shift Input
                                MEMORY          : in STD_LOGIC_VECTOR   (23 downto 0);  -- ALU Memory
Input
                                CCR_ARITH       : in STD_LOGIC_VECTOR   (3 downto 0);   -- CCR Arith Input
                                CCR_LOGIC       : in STD_LOGIC_VECTOR   (3 downto 0);   -- CCR Logic Input

                                ALU_OUT                         : out STD_LOGIC_VECTOR  (23 downto 0);  --
ALU Mux Output
                                CCR_OUT                         : out STD_LOGIC_VECTOR  (3 downto 0)    --
CCR Mux Output
                        );
end alu_output_mux;

architecture Combinational of alu_output_mux is

begin

        ----- Select Outputs: -----
        with sel select
                ALU_OUT <=      arith   when "0000",    -- ADD
                                        arith   when "0001",    -- SUB
                                        logic   when "0010",    -- AND
                                        logic   when "0011",    -- OR
                                        logic   when "0100",    -- CMP
                                        arith   when "0101",    -- ADDI
                                        logic   when "0110",    -- ANDI
                                        shift   when "0111",    -- SL
                                        shift   when "1000",    -- SR
                                        memory  when "1001",    -- LW
                                        memory  when OTHERS;    -- SW
        with sel select
                CCR_OUT <=      ccr_arith when "0000",  -- ADD
                                        ccr_arith when "0001",  -- SUB
                                        ccr_logic when "0010",  -- AND
                                        ccr_logic when "0011",  -- OR
                                        ccr_logic when "0100",  -- CMP
                                        ccr_arith when "0101",  -- ADDI
```

```
                                                ccr_logic when "0110",  -- ANDI
                                                "0000"    when OTHERS;   -- All flags cleared for other
logic operations

Modify for your machine as necessary
        ----- End Select Outputs -----

end Combinational;
```

```
----------------------------------------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
--
-- Create Date:    SPRING 2014
-- Module Name:    UMD_SHIFT_FPU
-- Project Name:         UMD-RISC 24
-- Target Devices: Spartan-3E
-- Tool versions:        Xilinx ISE 14.7
-- Description: Arithmetic Logic Unit - Shift Unit.
--      Performs all shift operations.
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity alu_shift_unit is

        Port (  A               : in    STD_LOGIC_VECTOR (23 downto 0); -- Register A Input
                        COUNT : in    STD_LOGIC_VECTOR (4 downto 0);  -- Number of Shifts
           SEL  : in  STD_LOGIC;                                                   -- Select
Shift Operation
           F            : out STD_LOGIC_VECTOR  (23 downto 0)   -- Shift Output
                        );

end alu_shift_unit;

architecture Combinational of alu_shift_unit is
        signal shift_left, shift_right : std_logic_vector (23 downto 0) := (OTHERS => '0'); -- Shift
operations
begin

        ----- Shift Unit: -----
        shift_left <= to_stdlogicvector(to_bitvector(a) sll conv_integer(count));
        shift_right <= to_stdlogicvector(to_bitvector(a) srl conv_integer(count));
        ----- End Shift Unit -----

        F <= shift_left when sel='0' else shift_right;

end Combinational;
```

```
--------------------------------------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
-- Create Date:    Spring 2014
-- Module Name:    UMD_ALU_FPU_tb
-- Project Name:         UMD-RISC 24
-- Target Devices: Spartan-3E
-- Tool versions:        Xilinx ISE 14.7
-- Description: VHDL Test Bench for ALU
--
-- Notes:
-- Verifies simple ALU operations.
----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY alu_tb_vhd IS
END alu_tb_vhd;

ARCHITECTURE behavior OF alu_tb_vhd IS

        -- Component Declaration for the Unit Under Test (UUT)
        COMPONENT alu_toplevel
        PORT(
                CLK : IN std_logic;
                RA : IN std_logic_vector(23 downto 0);
                RB : IN std_logic_vector(23 downto 0);
                OPCODE : IN std_logic_vector(3 downto 0);
                CCR : OUT std_logic_vector(3 downto 0);
                ALU_OUT : OUT std_logic_vector(23 downto 0)
                );
        END COMPONENT;

        --Inputs
        SIGNAL CLK : std_logic := '0';
        SIGNAL RA :  std_logic_vector(23 downto 0) := (others=>'0');
        SIGNAL RB :  std_logic_vector(23 downto 0) := (others=>'0');
        SIGNAL OPCODE :  std_logic_vector(3 downto 0) := (others=>'0');

        --Outputs
        SIGNAL CCR :  std_logic_vector(3 downto 0);
        SIGNAL ALU_OUT :  std_logic_vector(23 downto 0);

        -- Constants
        -- constant period : time := 20 ns; -- 25 MHz =(1/20E-9)/2
        constant period : time := 10 ns; -- 50 MHz =(1/10E-9)/2
        -- constant period : time := 5 ns; -- 100 MHz =(1/10E-9)/2

        --Condition Codes
        SIGNAL N : std_logic := '0';
        SIGNAL Z : std_logic := '0';
        SIGNAL V : std_logic := '0';
        SIGNAL C : std_logic := '0';

BEGIN

        -- Instantiate the Unit Under Test (UUT)
        uut: alu_toplevel PORT MAP(
                CLK => CLK,
                RA => RA,
                RB => RB,
```

```
                OPCODE => OPCODE,
                CCR => CCR,
                ALU_OUT => ALU_OUT
        );

        -- Assign condition code bits
        N <= CCR(3); -- N - Negative
        Z <= CCR(2); -- Z - Zero
        V <= CCR(1); -- V - Overflow
        C <= CCR(0); -- C - Carry/Borrow

        -- Generate clock
        m50MHZ_Clock: process
        begin
                clk <= '0'; wait for period;
                clk <= '1'; wait for period;
        end process m50MHZ_Clock;

        tb : PROCESS
        BEGIN

                -- Wait 100 ns for global reset to finish
                wait for 100 ns;

                report "Start ALU Test Bench" severity NOTE;

                ----- Register-Register Arithmetic Tests -----
                RA <= "00000000000000000000000101"; -- 5
                RB <= "00000000000000000000000011"; -- 3

                OPCODE <= "0000";  wait for period;
                assert (ALU_OUT = 8)  report "Failed ADD 1. ALU_OUT=" & integer'image(to_integer(unsigned
(ALU_OUT))) severity ERROR;
                assert (CCR = "0000")  report "Failed ADD 1 - CCR. CCR=" & integer'image(to_integer
(unsigned(CCR))) severity ERROR;
                OPCODE <= "0001";  wait for period;
                assert (ALU_OUT = 2)  report "Failed SUB 1. ALU_OUT=" & integer'image(to_integer(unsigned
(ALU_OUT))) severity ERROR;
                assert (CCR = "0000")  report "Failed SUB 1 - CCR. CCR=" & integer'image(to_integer
(unsigned(CCR))) severity ERROR;
                OPCODE <= "0010";  wait for period;
                assert (ALU_OUT = 1) report "Failed AND 1. ALU_OUT=" & integer'image(to_integer(unsigned
(ALU_OUT))) severity ERROR;
                assert (CCR = "0000")  report "Failed AND 1 - CCR. CCR=" & integer'image(to_integer
(unsigned(CCR))) severity ERROR;
                OPCODE <= "0011";  wait for period;
                assert (ALU_OUT = 7)  report "Failed OR 1. ALU_OUT=" & integer'image(to_integer(unsigned
(ALU_OUT))) severity ERROR;
                assert (CCR = "0000")  report "Failed OR 1 - CCR. CCR=" & integer'image(to_integer
(unsigned(CCR))) severity ERROR;

                RA <= "00000000000000000001100100"; -- 100
                RB <= "00000000000000000000110010"; -- 50

                OPCODE <= "0000";  wait for period;
                assert (ALU_OUT = 150)  report "Failed ADD 2. ALU_OUT=" & integer'image(to_integer
(unsigned(ALU_OUT))) severity ERROR;
                assert (CCR = "0000")  report "Failed ADD 2 - CCR. CCR=" & integer'image(to_integer
(unsigned(CCR))) severity ERROR;
                OPCODE <= "0001";  wait for period;
                assert (ALU_OUT = 50)  report "Failed SUB 2. ALU_OUT=" & integer'image(to_integer(unsigned
(ALU_OUT))) severity ERROR;
                assert (CCR = "0000")  report "Failed SUB 2 - CCR. CCR=" & integer'image(to_integer
(unsigned(CCR))) severity ERROR;
                OPCODE <= "0010";  wait for period;
                assert (ALU_OUT = "0000000000100000") report "Failed AND 2. ALU_OUT=" & integer'image
(to_integer(unsigned(ALU_OUT))) severity ERROR;
```

```
                assert (CCR = "0000")  report "Failed AND 2 - CCR. CCR=" & integer'image(to_integer
(unsigned(CCR))) severity ERROR;
                OPCODE <= "0011";  wait for period;
                assert (ALU_OUT = "0000000001110110")  report "Failed OR 2. ALU_OUT=" & integer'image
(to_integer(unsigned(ALU_OUT))) severity ERROR;
                assert (CCR = "0000")  report "Failed OR 2 - CCR. CCR=" & integer'image(to_integer
(unsigned(CCR))) severity ERROR;
                ----- END Arithmetic Tests -----

                ----- CCR Tests -----
                RA <= "000000000000000000000000";
                RB <= "000000000000000000000000";

                OPCODE <= "0000";  wait for period;
                assert (CCR(2) = '1')  report "Failed CCR 1 (Z). CCR=" & integer'image(to_integer(unsigned
(CCR))) severity ERROR;

                RA <= "000000000000000000000001";
                RB <= "111111111111111111111111";

                OPCODE <= "0000";  wait for period;
                assert (Z = '1')  report "Failed CCR 2 (Z). CCR=" & integer'image(to_integer(unsigned
(CCR))) severity ERROR;
                assert (C = '1')  report "Failed CCR 3 (C). CCR=" & integer'image(to_integer(unsigned
(CCR))) severity ERROR;

                RA <= "000000000000000000000000";
                RB <= "000000000000000000000001";

                OPCODE <= "0001";  wait for period;
                assert (N = '1')  report "Failed CCR 4 (N). CCR=" & integer'image(to_integer(unsigned
(CCR))) severity ERROR;

                RA <= "011111111111111111111111";
                RB <= "000000000000000000000001";

                OPCODE <= "0000";  wait for period;
                assert (V = '1')  report "Failed CCR 5 (V). CCR=" & integer'image(to_integer(unsigned
(CCR))) severity ERROR;

                RA <= "111111111111111111111111";
                RB <= "000000000000000000000001";

                OPCODE <= "0000";  wait for period;
                assert (C = '1')  report "Failed CCR 6 (C). CCR=" & integer'image(to_integer(unsigned
(CCR))) severity ERROR;
                ----- END CCR Tests -----

                -- Mem Test --

                OPCODE <= "1001";  wait for period;
                assert (ALU_OUT = 0) report "Failed MEMORY READ(1) ALU_OUT=" & integer'image(to_integer
(unsigned(CCR))) severity ERROR;

                RA <= X"000016";
                OPCODE <= "1010";  wait for period;
                assert (ALU_OUT = 0) report "Failed MEMORY WRITE ALU_OUT=" & integer'image(to_integer
(unsigned(CCR))) severity ERROR;

                OPCODE <= "1001";  wait for period;
                assert (ALU_OUT = X"16") report "Failed MEMORY READ(2) ALU_OUT=" & integer'image
(to_integer(unsigned(CCR))) severity ERROR;

                -- END Mem Test --

                report "Finish ALU Test Bench" severity NOTE;
```

```
                wait; -- will wait forever
        END PROCESS;

END;
```

```
----------------------------------------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
--
-- Create Date:    SPRING 2014
-- Module Name:    UMD_ALUTOP_FPU
-- Project Name:         UMD-RISC 24
-- Target Devices: Spartan-3E
-- Tool versions:        Xilinx ISE 14.7
-- Description: Arithmetic Logic Unit Top Level.
--      Structural defines all ALU modules.
--
--              Provides a sample solution for designing the UMD-RISC 24 machine's functional
--              processing unit (FPU).
--              Does not include any hardware testing. Integrate into your own machine as
--              desired.
--
--              Separates the shift and arithmetic units into separate entities for simplicity
--              and testing purposes.
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.all;

entity alu_toplevel is

        Port (  CLK     : in    STD_LOGIC;                                                      --
50 MHz Oscillator
                          RA            : in    STD_LOGIC_VECTOR (23 downto 0); -- Input 1
                          RB            : in    STD_LOGIC_VECTOR (23 downto 0); -- Input 2
           OPCODE : in  STD_LOGIC_VECTOR (3 downto 0);  -- Current opcode
                          CCR                           : out   STD_LOGIC_VECTOR (3 downto 0);  --
Condition Codes (N,Z,V,C)
           ALU_OUT                      : out STD_LOGIC_VECTOR (23 downto 0));  -- Output from ALU

end alu_toplevel;

architecture Structural of alu_toplevel is

        -- 24-bit Outputs
        signal arith    : STD_LOGIC_VECTOR (23 downto 0) := (OTHERS => '0'); -- Arithmetic Unit Output
        signal logic    : STD_LOGIC_VECTOR (23 downto 0) := (OTHERS => '0'); -- Logic Unit Output
        signal shift    : STD_LOGIC_VECTOR (23 downto 0) := (OTHERS => '0'); -- Shift Unit Output
        signal memory   : STD_LOGIC_VECTOR (23 downto 0) := (OTHERS => '0'); -- Load Store Unit Output

        -- 4-bit Condition Codes
        signal ccr_arith        : STD_LOGIC_VECTOR (3 downto 0) := (OTHERS => '0'); -- Arithmetic Unit CCR
        signal ccr_logic        : STD_LOGIC_VECTOR (3 downto 0) := (OTHERS => '0'); -- Logic Unit CCR

begin

        ----- Structural Components: -----
        -- ALU Units
        arithmetic_unit: entity work.arithmetic_unit
        port map(a                      => ra,                                                  -- ALU Input
                                 b                      => rb,                                                  --
ALU Input
                                 sel            => opcode(2 downto 0),  -- Only need least significant 3
bits

                                 ccr            => ccr_arith,                           -- Route to ALU
```

```
Output Multiplexer
                                f                       =>
arith                                           -- Route to ALU Output Multiplexer
                                );
        logic_unit: entity work.logic_unit
        port map(a                      => ra,                                  -- ALU Input
                                b                       => rb,                                  --
ALU Input
                                sel             => opcode(2 downto 0),  -- Only need least significant 3
bits

                                ccr             => ccr_logic,                   -- Route to ALU
Output Multiplexer
                                f                       =>
logic                                           -- Route to ALU Output Multiplexer
                                );
        shift_unit: entity work.alu_shift_unit
        port map(a                      => ra,                                  -- ALU Input
                                count   => rb(4 downto 0),      -- ALU Input
                                sel             => opcode(3),                   -- Only 2 shift ops
(opcode="0111", "1000")

                                f                       => shift                                --
Route to ALU Output Multiplexer
                                );
        load_store_unit: entity work.load_store_unit
        port map(clk            => clk,         -- ALU Input
                                ra                      => ra,          -- ALU Input
                                immed           => rb,          -- ALU Input
                                opcode  => opcode,      -- Need entire opcode to isolate SW instruction

                                f                       => memory       -- Route to ALU Output Multiplexer
                                );
        -- Multiplexers
        alu_output_mux: entity work.alu_output_mux
        port map(sel                    => opcode,              -- ALU Input
                                arith           => arith,               -- From arithmetic_unit
                                logic           => logic,               -- From logic_unit
                                shift           => shift,               -- From shift_unit
                                memory          => memory,              -- From load_store_unit
                                ccr_arith       => ccr_arith,   -- From arithmetic_unit
                                ccr_logic       => ccr_logic,   -- From logic_unit

                                alu_out                 => alu_out,             -- ALU Output
                                ccr_out                 => ccr                  -- ALU Output
                                );
        ----- End Structural Components -----

end Structural;
```

```
----------------------------------------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
--
-- Create Date:    SPRING 2014
-- Module Name:    UMD_ALU_FPU
-- Project Name:   UMD-RISC 24
-- Target Devices: Spartan-3E
-- Tool versions:        Xilinx ISE 14.7
-- Description: Debounce
--      Debounce switches
----------------------------------------
Library ieee;
USE ieee.STD_LOGIC_1164.all;
USE ieee.STD_LOGIC_UNSIGNED.all;

ENTITY DEBOUNCE IS
PORT (
  Clk       : IN STD_LOGIC;             -- reduced system clock
  Key       : IN STD_LOGIC;             -- active low input
  pulse     : OUT STD_LOGIC);           --debounced output
END DEBOUNCE;

ARCHITECTURE clean_pulse OF DEBOUNCE IS
  SIGNAL cnt       :integer range 0 to 1000000 := 0;
BEGIN
  PROCESS (Clk,Key,cnt)
  BEGIN
    IF Key = '1' THEN
      cnt <= 0;
    ELSIF (clk'EVENT AND Clk = '1') THEN
      IF (cnt /= 1000000) THEN cnt <= cnt + 1; END IF;
    END IF;
    IF (cnt = 999999) AND (Key = '0') THEN pulse <= '1'; ELSE pulse <= '0'; END IF;
  END PROCESS;
END clean_pulse;
```

```
----------------------------------------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
-- Create Date:    SPRING 2014
-- Design Name:
-- Module Name:    input_mux - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Revision 0.02 - Created a central process for the buttons
-- Revision 0.03 - Swaped reset from rotar button to a push button due to input error
--                                          From the push buttom double clicking.
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;



entity input_mux is
        Port (
                                Sel_BTN : std_logic_vector (4 downto 0);
                                Value_SW : std_logic_vector (3 downto 0);
                                --test :out STD_LOGIC_VECTOR    (3 downto 0);
                                RA                      : OUT STD_LOGIC_VECTOR  (23 downto 0);  -- ALU
Mux Output
                                RB                      : OUT STD_LOGIC_VECTOR  (23 downto 0);  -- CCR
Mux Output
                                OPCODE  : OUT STD_LOGIC_VECTOR  (3 downto 0);   -- CCR Mux Output
                                POSITION : OUT STD_LOGIC_VECTOR (3 downto 0)
                                --LED_SEL       : BUFFER STD_LOGIC_VECTOR       (2 downto 0);
                                --LED_Value : BUFFER STD_LOGIC_VECTOR   (3 downto 0)
                        );
end input_mux;

architecture behavior of input_mux is
        signal Value : std_logic_vector (23 downto 0) := (OTHERS => '0'); -- Shift operations
        --signal position : integer range 0 to 6 := 0;

        signal RA0 : std_logic_vector (23 downto 0) := (OTHERS => '0');
        signal RA1 : std_logic_vector (23 downto 0) := (OTHERS => '0');

        signal RB0 : std_logic_vector (23 downto 0) := (OTHERS => '0');
        signal RB1 : std_logic_vector (23 downto 0) := (OTHERS => '0');


        signal OPCODE0 : std_logic_vector (3 downto 0) := (OTHERS => '0');
        signal OPCODE1 : std_logic_vector (3 downto 0) := (OTHERS => '0');


        signal pos : integer range 0 to 6 := 6;
type OUT_STATE is (RESET, STD);
signal OUTPUT_STATE : OUT_STATE := STD;
```

```
        --signal sel : std_logic_vector (3 downto 0) := (OTHERS => '0');
        --signal Value_SW : std_logic_vector (3 downto 0) := (OTHERS => '0');

begin

RA <= RA0;
RB <= RB0;
OPCODE <= OPCODE0;
POSITION <= conv_std_logic_vector(pos,4); --std_logic_vector(to_signed(pos, 4));

--
--      with OUTPUT_STATE select
--              RA0 <= RA1 when STD,
--                              X"000000" when RESET;
--
--      with OUTPUT_STATE select
--              RB0 <= RB1 when STD,
--                              X"000000" when RESET;
--
--      with OUTPUT_STATE select
--              OPCODE0 <= OPCODE1 when STD,
--                              X"0" when RESET;

--      process(OUTPUT_STATE)
--      begin
--              OUTPUT_


        process(Sel_BTN)
        begin
--              if  ( Sel_BTN(4) = '1' and Sel_BTN(4)'event ) then -- Seems to double or triple click
when changing push button.
--                      --OUTPUT_STATE <= STD;
--                      --RA <= RA + X"000001";
----                    if(pos < 6) then
----                            pos <= pos+1;
----                    else
--                              --pos <= 0;
--                              --RA0 <= x"000000";
--                              --RB0 <= x"000000";
----                    end if;
--
--                      --test <= conv_std_logic_vector(position, 3)(2 downto 0);
----                    Value((position * 4+3) downto (position * 4 )) <= Value_SW(3 downto 0);
----                    position <= (position + 1) ;
----
----                    if (position = 6) then
----                            position <= 0;
----                    end if;
--              end if;

                if  ( Sel_BTN(3) = '1' and Sel_BTN(3)'event ) then
                        OPCODE0 <= Value_SW;
                end if;

                if  ( Sel_BTN(2) = '1' and Sel_BTN(2)'event ) then
                        if(pos < 5) then
                                pos <= pos+1;
                        else
                                pos <= 0;
                        end if;
                end if;

                if  ( Sel_BTN(1) = '1' and Sel_BTN(1)'event ) then
                        RB0(23 - pos*4 downto 20 - pos*4) <= Value_SW;
                end if;
```

```
                if  ( Sel_BTN(0) = '1' and Sel_BTN(0)'event ) then
                        RA0(23 - pos*4 downto 20 - pos*4) <= Value_SW;
                end if;

        end process;
        ----- Create Values -----
--      process(Sel_BTN(3))
--      begin
--
--      end process;
--
--      process(Sel_BTN(2))
--      begin
--
--      end process;
--
--      process(Sel_BTN(1))
--      begin
--
--      end process;
--      process(Sel_BTN(0))
--      begin
--
--      end process;
--      with Sel_BTN select
--              RA              <= Value when "00001",  -- RA
--                                 x"A22222" when "01000",
--                                 RA when OTHERS;
--      with Sel_BTN select
--              RB                      <=      Value when "00010",     -- RB
--                                      X"B22222" when "01000",
--                                      RB when OTHERS;
--      with Sel_BTN select
--              OPCODE  <=      Value_SW when "00100", -- OPCODE
--                                       "0000" when "01000",
--                                       OPCODE  when OTHERS;
        --LED_SEL <= SEL;
        --LED_VALUE <= Value_SW;
        --test <= OPCODE;
end behavior;
```

```
## School: University of Massachusetts Dartmouth
## Department: Computer and Electrical Engineering
## Class: ECE 368 Digital Design
## Engineer: Daniel Noyes
##                                Massarrah Tannous
##
## Create Date:    SPRING 2014
##
#
# Period constraint for 50MHz operation
NET "CLOCK" TNM_NET = "CLOCK";
TIMESPEC TS_CLOCK = PERIOD "CLOCK" 40 ns HIGH 50 %;
#OFFSET = IN 10 ns BEFORE "CLOCK";
#OFFSET = OUT 10 ns AFTER "CLOCK";
#
#
# Soldered 50MHz Clock.
NET "CLOCK" LOC = "C9";

# Simple LEDs
# Require only 3.5mA.
##
NET "OPCODE_Value[0]" LOC = "F12" ;     # right to left
NET "OPCODE_Value[1]" LOC = "E12";
NET "OPCODE_Value[2]" LOC = "E11";
NET "OPCODE_Value[3]" LOC = "F11" ;
NET "LED2<0>" LOC = "C11";
NET "LED2<1>" LOC = "D11";
NET "LED2<2>" LOC = "E9" ;
NET "LED2<3>" LOC = "F9" ;

##
## Simple switches
##   Pull UP resistors used to stop floating condition during switching.
##
NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP;  # right to left
NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP;
NET "SW<2>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP;
NET "SW<3>" LOC = "N17" | IOSTANDARD = LVTTL | PULLUP;
##
##
##

##   Must have pull DOWN resistors to provide LOW when not pressed.
##
NET "BTN_SOUTH" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN;
NET "BTN_NORTH"  LOC = "V4"  | IOSTANDARD = LVTTL | PULLDOWN;   # btn_north
NET "BTN_EAST"  LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN;    # btn_east
NET "BTN_CTR"  LOC = "V16" | IOSTANDARD = LVTTL | PULLDOWN;     # btn_center
NET "BTN_WEST"  LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN;    # btn_west
NET "BTN_CTR" CLOCK_DEDICATED_ROUTE = FALSE;

##
## LCD DISPLAY PINS
##
NET "SF_CE0"  LOC = "D16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "LCD_E"   LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "LCD_RS"  LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "LCD_RW"  LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
# The LCD four-bit data interface is shared with the StrataFlash.
NET "SF_D<0>" LOC  = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "SF_D<1>" LOC  = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "SF_D<2>" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET "SF_D<3>" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
```

```
----------------------------------------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
-- Create Date:    22:52:01 02/03/2014
-- Design Name:
-- Module Name:    Input_toplevel - Sturctural
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Input_toplevel is
    Port (  CLOCK       : in    STD_LOGIC;                                                  --
50 MHz Oscillator
                            BTN_WEST : in  STD_LOGIC;
           BTN_NORTH : in  STD_LOGIC;
           BTN_EAST : in  STD_LOGIC;
           BTN_SOUTH : in  STD_LOGIC;
           BTN_CTR : in  STD_LOGIC;
           SW : in  STD_LOGIC_VECTOR (3 downto 0);
           --LD0 : out  STD_LOGIC;
                            --,LD1,LD2,LD3,LD4,LD5,LD6
                            --test : out STD_LOGIC_VECTOR (3 downto 0);
                            OPCODE_Value  : out STD_LOGIC_VECTOR (3 downto 0) ;
           --RA :buffer STD_LOGIC_VECTOR (23 downto 0);
                            --RB  :buffer STD_LOGIC_VECTOR (23 downto 0);
                            --LCD DISPLAY PORTS
                            SF_D : out STD_LOGIC_VECTOR(3 downto 0);
                            LCD_E, LCD_RS, LCD_RW, SF_CE0 : out STD_LOGIC ;

                            LED2 : out STD_LOGIC_VECTOR(3 downto 0)
                            );
end Input_toplevel;


architecture Sturctural  of Input_toplevel is

signal DEBOUNCE_BTN_CENTER : STD_LOGIC :='0';
signal DEBOUNCE_BTN_WEST : STD_LOGIC :='0';
signal DEBOUNCE_BTN_EAST : STD_LOGIC :='0';
```

```
signal DEBOUNCE_BTN_NORTH : STD_LOGIC :='0';
signal DEBOUNCE_BTN_SOUTH : STD_LOGIC :='0';

Signal CCR : STD_LOGIC_VECTOR (3 downto 0);     -- Condition Codes (N,Z,V,C)
Signal ALU_OUT  : STD_LOGIC_VECTOR (23 downto 0);       -- Output from ALU

        -- 24-bit Outputs
        signal RA       : STD_LOGIC_VECTOR (23 downto 0) ;--:= (OTHERS => x"123456"); -- Arithmetic Unit
Output
        signal RB       : STD_LOGIC_VECTOR (23 downto 0) ; -- Logic Unit Output
        signal OPCODE_SIG       : STD_LOGIC_VECTOR (3 downto 0); -- Operation Code
        signal POSITION : STD_LOGIC_VECTOR (3 downto 0); -- position location to change bit



begin
        --RA (3 downto 0) <= SW;
        --RB <= x"BBBBBB";
        --OPCODE_SIG <= "0000";
        LED2 <= POSITION;

        PROCESS (CLOCK)
        BEGIN
        If( CLOCK = '1' and CLOCK'event )then
                        OPCODE_Value <= OPCODE_SIG ;
        end if ;
        END PROCESS;
        ----- Structural Components: -----
        -- ALU Units
        debounce_unit : entity work.debounce
        port map (
                Clk     => CLOCK,               -- reduced system clock
                Key     => BTN_CTR,     -- active low input
                pulse   => DEBOUNCE_BTN_CENTER                  --debounced output
        );

        Debounce_unit_WEST : entity work.debounce
                port map (
                Clk     => CLOCK,               -- reduced system clock
                Key     => BTN_WEST,    -- active low input
                pulse   => DEBOUNCE_BTN_WEST            --debounced output
        );

        Debounce_unit_EAST : entity work.debounce
                port map (
                Clk     => CLOCK,               -- reduced system clock
                Key     => BTN_EAST,    -- active low input
                pulse   => DEBOUNCE_BTN_EAST            --debounced output
        );

        Debounce_unit_NORTH : entity work.debounce
                port map (
                Clk     => CLOCK,               -- reduced system clock
                Key     => BTN_NORTH,   -- active low input
                pulse   => DEBOUNCE_BTN_NORTH                   --debounced output
        );

        Debounce_unit_SOUTH : entity work.debounce
                port map (
                Clk     => CLOCK,               -- reduced system clock
                Key     => BTN_SOUTH,   -- active low input
                pulse   => DEBOUNCE_BTN_SOUTH                   --debounced output
        );


        input_unit : entity work.input_mux
        port map(
```

```
                                Sel_BTN(0) => DEBOUNCE_BTN_WEST,
            Sel_BTN(1) => DEBOUNCE_BTN_NORTH,
            Sel_BTN(2) => DEBOUNCE_BTN_EAST ,
            Sel_BTN(3) => DEBOUNCE_BTN_SOUTH,
            Sel_BTN(4) => DEBOUNCE_BTN_CENTER,
                                Value_SW => SW,
                           RA => RA,
                           RB => RB,
                           OPCODE => OPCODE_SIG,
                                POSITION => POSITION
                       );


        alu_unit : entity work.alu_toplevel
        port map(CLK                    => CLOCK,
                                RA                      => RA,                                  -- ALU
Input
                                RB                      => RB,                                  -- ALU
Input
                                OPCODE          => OPCODE_SIG,                          -- Only need
least significant 3 bits
                                CCR                     => CCR,                                 -- CCR
                                ALU_OUT         => ALU_OUT                              -- ALU_OUT
                                );

        LCD_Display : entity work.LCD_Display
        port map(CLK                    => CLOCK,
                                RESET                   => DEBOUNCE_BTN_SOUTH,
                                A                               => RA,
                                B                               => RB,
                                C                               => ALU_OUT,
                                OPCODE          => OPCODE_SIG,
                                CCR                     => CCR,
                                SF_D                    => SF_D,
                                LCD_E                   => LCD_E,
                                LCD_RS          => LCD_RS,
                                LCD_RW          => LCD_RW,
                                SF_CE0          => SF_CE0,
                                POSITION    => POSITION
                                );


end Sturctural ;
```

```
----------------------------------------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
-- Create Date:    SPRING 2014
-- Design Name:
-- Module Name:    LCD_DISPLAY - behavior
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description: This LCD driver was modified from the cosmiac tutorial 6 by Rahul Vora.
--                                      http://www.cosmiac.org/tutorial_6.html
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Revision 0.02 - Implemented a LCD buffer
-- Revision 0.03 - Implemented a hexnum to display converter
-- Revision 0.04 - Added a refresh cycle at 1 Hz
-- Revision 0.05 - Finished implementing Next Line Function
-- Revision 0.06 - Added the CCR registers into the LCD
-- Revision 0.07 - Convert design from bit to std_logic
-- Revision 0.08 - Cleaned up the test signals and set up for ALU
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.all;


entity LCD_DISPLAY is
        port(
                clk, reset : in STD_LOGIC;
                SF_D : out STD_LOGIC_vector(3 downto 0);
                LCD_E, LCD_RS, LCD_RW, SF_CE0 : out STD_LOGIC;
                A,B,C : in STD_LOGIC_vector(23 downto 0) ;
                OPCODE, CCR, POSITION: in STD_LOGIC_vector(3 downto 0)
                );
end LCD_DISPLAY;

architecture behavior of LCD_DISPLAY is

type tx_sequence is (high_setup, high_hold, oneus, low_setup, low_hold, fortyus, done);
signal tx_state : tx_sequence := done;
signal tx_byte : STD_LOGIC_vector(7 downto 0);
signal tx_init : STD_LOGIC := '0';

type init_sequence is (idle, fifteenms, one, two, three, four, five, six, seven, eight, done);
signal init_state : init_sequence := idle;
signal init_init, init_done : STD_LOGIC := '0';

signal i : integer range 0 to 750000 := 0;
signal i2 : integer range 0 to 2000 := 0;
signal i3 : integer range 0 to 82000 := 0;
signal ir : integer range 0 to 50000000 := 0; -- 50 MHz in which brings it to 1 second
signal DISP_INDEX : integer range 0 to 32 := 0;

signal SF_D0, SF_D1 : STD_LOGIC_vector(3 downto 0);
signal LCD_E0, LCD_E1 : STD_LOGIC;
signal mux : STD_LOGIC;
```

```
signal HEXNUM : STD_LOGIC_vector(4 downto 0); -- [special char][hex number]
signal NUM : STD_LOGIC_vector(7 downto 0);

signal LCD_BUFFER : STD_LOGIC_vector(127 downto 0) := (OTHERS => '0');

type display_state is (init, function_set, entry_set, set_display, clr_display, pause, set_addr, DISP,
NEWLINE, done);
signal cur_state : display_state := init;

--signal newlinecheck : bit_vector(7 downto 0) := (OTHERS => "01101101");

begin

        SF_CE0 <= '1'; --disable intel strataflash
        LCD_RW <= '0'; --write only

        --The following "with" statements simplify the process of adding and removing states.

        --when to transmit a command/data and when not to
        with cur_state select
                tx_init <= '0' when init | pause | done,
                        '1' when others;

        --control the bus
        with cur_state select
                mux <= '1' when init,
                        '0' when others;

        --control the initialization sequence
        with cur_state select
                init_init <= '1' when init,
                        '0' when others;

        --register select
        with cur_state select
                LCD_RS <= '0' when function_set|entry_set|set_display|clr_display|set_addr|NEWLINE,
                        '1' when others;

        --what byte to transmit to LCD
        --refer to datasheet for an explanation of these values
        with cur_state select
                tx_byte <= "00101000" when function_set,
                        "00000110" when entry_set,
                        "00001100" when set_display,
                        "00000001" when clr_display,
                        "10000000" when set_addr,
                        NUM               when DISP,
                        X"C0"             when NEWLINE,
                        "00000000" when others;

        --LCD decoder
        with HEXNUM select
                NUM <=
                "00110000" when "00000", -- 0
                "00110001" when "00001", -- 1
                "00110010" when "00010", -- 2
                "00110011" when "00011", -- 3
                "00110100" when "00100", -- 4
                "00110101" when "00101", -- 5
                "00110110" when "00110", -- 6
                "00110111" when "00111", -- 7
                "00111000" when "01000", -- 8
                "00111001" when "01001", -- 9
                "01000001" when "01010", -- A
                "01000010" when "01011", -- B
                "01000011" when "01100", -- C
```

```
		"01000100" when "01101", -- D
		"01000101" when "01110", -- E
		"01000110" when "01111", -- F
		"00101011" when "10000", -- ADD
		"00101101" when "10001", -- SUB
		"00100110" when "10010", -- AND
		"01111100" when "10011", -- OR
		"00111101" when "10100", -- CMP
		"00101011" when "10101", -- ADDI
		"00100110" when "10110", -- ANDI
		"01111111" when "10111", -- SL
		"01111110" when "11000", -- SR
		"00101100" when "11001", -- LW
		"00101110" when "11010", -- SW
		"01100011" when "11011", -- CCR(0): Carry
		"01101111" when "11100", -- CCR(1): OverFlow
		"01111010" when "11101", -- CCR(2): Zero
		"11101001" when "11110", -- CCR(3): Negative
		"11111110" when "11111", -- space
		--x"80" when "11111", -- hex 40 for Next Line
		"11111110" when OTHERS; -- ERROR so space

--Set the LCD buffer
LCD_Buff: process(clk)
begin

if(clk='1' and clk'event) then

		LCD_BUFFER(127 downto 104) <= A;
		LCD_BUFFER(87 downto 64) <= B;
		LCD_BUFFER(99 downto 96) <= OPCODE; -- op Codes
		LCD_BUFFER(95 downto 92) <= OPCODE; -- op Codes
		LCD_BUFFER(63 downto 40) <= C;
		LCD_BUFFER(23 downto 20) <= POSITION;
		LCD_BUFFER(15 downto 12) <= CCR;
		LCD_BUFFER(11 downto 8) <= CCR;
		LCD_BUFFER(7 downto 4) <= CCR;
		LCD_BUFFER(3 downto 0) <= CCR;

end if;

end process LCD_Buff;

--main state machine
display: process(clk, reset)
begin
		if(reset='1') then
			cur_state <= function_set;
		elsif(clk='1' and clk'event) then
			case cur_state is
				--refer to intialize state machine below
				when init =>
					if(init_done = '1') then
						cur_state <= function_set;
					else
						cur_state <= init;
					end if;

				--every other state but pause uses the transmit state machine
				when function_set =>
					if(i2 = 2000) then
						cur_state <= entry_set;
					else
						cur_state <= function_set;
					end if;

				when entry_set =>
```

```
                        if(i2 = 2000) then
                                cur_state <= set_display;
                        else
                                cur_state <= entry_set;
                        end if;

                when set_display =>
                        if(i2 = 2000) then
                                cur_state <= clr_display;
                        else
                                cur_state <= set_display;
                        end if;

                when clr_display =>
                        i3 <= 0;
                        if(i2 = 2000) then
                                cur_state <= pause;
                        else
                                cur_state <= clr_display;
                        end if;

                when pause =>
                        if(i3 = 82000) then
                                cur_state <= set_addr;
                                i3 <= 0;
                        else
                                cur_state <= pause;
                                i3 <= i3 + 1;
                        end if;

                when set_addr =>
                        if(i2 = 2000) then
                                DISP_INDEX <= 1;
                                HEXNUM(4) <= '0';
                                HEXNUM(3 downto 0) <= LCD_BUFFER(127 downto 124);
                                cur_state <= DISP;
                        else
                                cur_state <= set_addr;
                        end if;

                when NEWLINE =>
                if(i2 = 2000) then
                                cur_state <= DISP;
                        else
                                cur_state <= NEWLINE;
                        end if;


                when DISP =>
                        if(i2 = 2000) then
                                if(DISP_INDEX = 32) then
                                        cur_state <= done;
                                else
                                        DISP_INDEX <= DISP_INDEX + 1;
                                        if(DISP_INDEX> 5 and DISP_INDEX< 10) then
                                                if(DISP_INDEX = 6 or DISP_INDEX = 9) then
                                                        HEXNUM <= "11111"; -- Space
                                                else
                                                        HEXNUM(3 downto 0) <= LCD_BUFFER
((127-4*DISP_INDEX) downto (124-4*DISP_INDEX));

                                                        HEXNUM(4) <=

'1';


                                                end if;
                                        else
                                                        if(DISP_INDEX > 27) then
                                                                if(DISP_INDEX = 28) then
```

```
                                                                        if(LCD_BUFFER
(127-4*DISP_INDEX) = '1') then -- CCR(3)
                                                                                HEXNUM <=
"11110";
                                                                        else
                                                                                HEXNUM <=
"11111";
                                                                        end if;
                                                                end if;
                                                                if(DISP_INDEX = 29) then
                                                                        if(LCD_BUFFER
(127-4*DISP_INDEX - 1) = '1') then -- CCR(2)
                                                                                HEXNUM <=
"11101";
                                                                        else
                                                                                HEXNUM <=
"11111";
                                                                        end if;
                                                                end if;
                                                                if(DISP_INDEX = 30) then
                                                                        if(LCD_BUFFER
(127-4*DISP_INDEX - 2) = '1') then -- CCR(1)
                                                                                HEXNUM <=
"11100";
                                                                        else
                                                                                HEXNUM <=
"11111";
                                                                        end if;
                                                                end if;
                                                                if(DISP_INDEX = 31) then
                                                                        if(LCD_BUFFER
(127-4*DISP_INDEX - 3) = '1') then -- CCR(0)
                                                                                HEXNUM <=
"11011";
                                                                        else
                                                                                HEXNUM <=
"11111";
                                                                        end if;
                                                                end if;

                                                        else
                                                                if(DISP_INDEX> 21 and
DISP_INDEX< 28 and not (DISP_INDEX = 26)) then
                                                                        HEXNUM <= "11111";
                                                                else
                                                                        HEXNUM(4) <= '0';
                                                                        HEXNUM(3 downto
0) <= LCD_BUFFER((127-4*DISP_INDEX) downto (124-4*DISP_INDEX));
                                                                end if;
                                                        end if;
                                                end if;
                                        end if;
                                        if(Disp_INDEX = 16) then
                                                cur_state <=  NEWLINE;
                                        end if;
                                else
                                        cur_state <= DISP;
                                end if;

                        when done =>
                                if(ir = 50000000) then
                                ir <= 0;
                                cur_state <= function_set;
                                else
                                ir <= ir + 1;
                                cur_state <= done;
                                end if;
```

```
                end case;
        end if;
end process display;

with mux select
        SF_D <= SF_D0 when '0', --transmit
                SF_D1 when others;      --initialize
with mux select
        LCD_E <= LCD_E0 when '0', --transmit
                LCD_E1 when others; --initialize

--specified by datasheet
transmit : process(clk, reset, tx_init)
begin
        if(reset='1') then
                tx_state <= done;
        elsif(clk='1' and clk'event) then
                case tx_state is
                        when high_setup => --40ns
                                LCD_E0 <= '0';
                                SF_D0 <= tx_byte(7 downto 4);
                                if(i2 = 2) then
                                        tx_state <= high_hold;
                                        i2 <= 0;
                                else
                                        tx_state <= high_setup;
                                        i2 <= i2 + 1;
                                end if;

                        when high_hold => --230ns
                                LCD_E0 <= '1';
                                SF_D0 <= tx_byte(7 downto 4);
                                if(i2 = 12) then
                                        tx_state <= oneus;
                                        i2 <= 0;
                                else
                                        tx_state <= high_hold;
                                        i2 <= i2 + 1;
                                end if;

                        when oneus =>
                                LCD_E0 <= '0';
                                if(i2 = 50) then
                                        tx_state <= low_setup;
                                        i2 <= 0;
                                else
                                        tx_state <= oneus;
                                        i2 <= i2 + 1;
                                end if;

                        when low_setup =>
                                LCD_E0 <= '0';
                                SF_D0 <= tx_byte(3 downto 0);
                                if(i2 = 2) then
                                        tx_state <= low_hold;
                                        i2 <= 0;
                                else
                                        tx_state <= low_setup;
                                        i2 <= i2 + 1;
                                end if;

                        when low_hold =>
                                LCD_E0 <= '1';
                                SF_D0 <= tx_byte(3 downto 0);
                                if(i2 = 12) then
                                        tx_state <= fortyus;
                                        i2 <= 0;
```

```
                                else
                                        tx_state <= low_hold;
                                        i2 <= i2 + 1;
                                end if;

                        when fortyus =>
                                LCD_E0 <= '0';
                                if(i2 = 2000) then
                                        tx_state <= done;
                                        i2 <= 0;
                                else
                                        tx_state <= fortyus;
                                        i2 <= i2 + 1;
                                end if;

                        when done =>
                                LCD_E0 <= '0';
                                if(tx_init = '1') then
                                        tx_state <= high_setup;
                                        i2 <= 0;
                                else
                                        tx_state <= done;
                                        i2 <= 0;
                                end if;

                end case;
        end if;
end process transmit;

--specified by datasheet
power_on_initialize: process(clk, reset, init_init) --power on initialization sequence
begin
        if(reset='1') then
                init_state <= idle;
                init_done <= '0';
        elsif(clk='1' and clk'event) then
                case init_state is
                        when idle =>
                                init_done <= '0';
                                if(init_init = '1') then
                                        init_state <= fifteenms;
                                        i <= 0;
                                else
                                        init_state <= idle;
                                        i <= i + 1;
                                end if;

                        when fifteenms =>
                                init_done <= '0';
                                if(i = 750000) then
                                        init_state <= one;
                                        i <= 0;
                                else
                                        init_state <= fifteenms;
                                        i <= i + 1;
                                end if;

                        when one =>
                                SF_D1 <= "0011";
                                LCD_E1 <= '1';
                                init_done <= '0';
                                if(i = 11) then
                                        init_state<=two;
                                        i <= 0;
                                else
                                        init_state<=one;
                                        i <= i + 1;
```

```
        end if;

    when two =>
        LCD_E1 <= '0';
        init_done <= '0';
        if(i = 205000) then
                init_state<=three;
                i <= 0;
        else
                init_state<=two;
                i <= i + 1;
        end if;

    when three =>
        SF_D1 <= "0011";
        LCD_E1 <= '1';
        init_done <= '0';
        if(i = 11) then
                init_state<=four;
                i <= 0;
        else
                init_state<=three;
                i <= i + 1;
        end if;

    when four =>
        LCD_E1 <= '0';
        init_done <= '0';
        if(i = 5000) then
                init_state<=five;
                i <= 0;
        else
                init_state<=four;
                i <= i + 1;
        end if;

    when five =>
        SF_D1 <= "0011";
        LCD_E1 <= '1';
        init_done <= '0';
        if(i = 11) then
                init_state<=six;
                i <= 0;
        else
                init_state<=five;
                i <= i + 1;
        end if;

    when six =>
        LCD_E1 <= '0';
        init_done <= '0';
        if(i = 2000) then
                init_state<=seven;
                i <= 0;
        else
                init_state<=six;
                i <= i + 1;
        end if;

    when seven =>
        SF_D1 <= "0010";
        LCD_E1 <= '1';
        init_done <= '0';
        if(i = 11) then
                init_state<=eight;
                i <= 0;
        else
```

```
                                        init_state<=seven;
                                        i <= i + 1;
                                end if;

                        when eight =>
                                LCD_E1 <= '0';
                                init_done <= '0';
                                if(i = 2000) then
                                        init_state<=done;
                                        i <= 0;
                                else
                                        init_state<=eight;
                                        i <= i + 1;
                                end if;

                        when done =>
                                init_state <= done;
                                init_done <= '1';

                end case;

        end if;
    end process power_on_initialize;

end behavior;
```

```
----------------------------------------------------------------------------------
-- School: University of Massachusetts Dartmouth
-- Department: Computer and Electrical Engineering
-- Class: ECE 368 Digital Design
-- Engineer: Daniel Noyes
--                              Massarrah Tannous
--
--
-- Create Date:    SPRING 2014
-- Module Name:    UMD_LOAD_STORE_FPU
-- Project Name:         UMD-RISC 24
-- Target Devices: Spartan-3E
-- Tool versions:        Xilinx ISE 14.7
-- Description: Arithmetic Logic Unit - Load/Store Unit.
--      Performs all memory operations.
--
--              Example for memory read/write instructions using only one register.
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity load_store_unit is

        Port (  CLK             : in
STD_LOGIC;                                                      -- 50 MHz Oscillator
                          OPCODE        : in    STD_LOGIC_VECTOR (3 downto 0);  --      Opcode for
current instruction
                          RA            : in    STD_LOGIC_VECTOR (23 downto 0); -- Memory input 1
                          IMMED : in    STD_LOGIC_VECTOR (23 downto 0); -- Memory input 2 (Since we are
not indexing into memory, unused)

           F            : out STD_LOGIC_VECTOR  (23 downto 0)   -- 24-bit memory output
                          );

end load_store_unit;

architecture Behavioral of load_store_unit is

        -- Since this ALU memory block does not interface with the final machine's
        --      memory, we will use one 24-bit register to show the basic functionality
        --      of a load/store.
        signal memory_register : std_logic_vector (23 downto 0) := (OTHERS => '0');

        -- Decode opcode to determine when a load/store will occur.
        -- Since we are multiplexing the output, we are free to read from memory
        --      on all cycles other than SW. Only on SW do we want to write into memory.
        --
        -- '1' = write, '0' = read
        signal wen : std_logic := '0';
begin

        -- Assign write enable based on current opcode
        wen <= '1' when opcode="1010" else '0';

        -- Write to memory register during rising clock edge of SW instruction
        process(clk)
        begin
                if (clk'event and clk='1') then
                        if (wen = '1') then -- If current instruction is SW
                                memory_register <= RA;
                        end if;
                end if;
        end process;

        -- Output value in memory when we are not writing to it
```

```
        F <= memory_register;

end Behavioral;
```