

ECEN 160L Final Project

Four-bit Processor

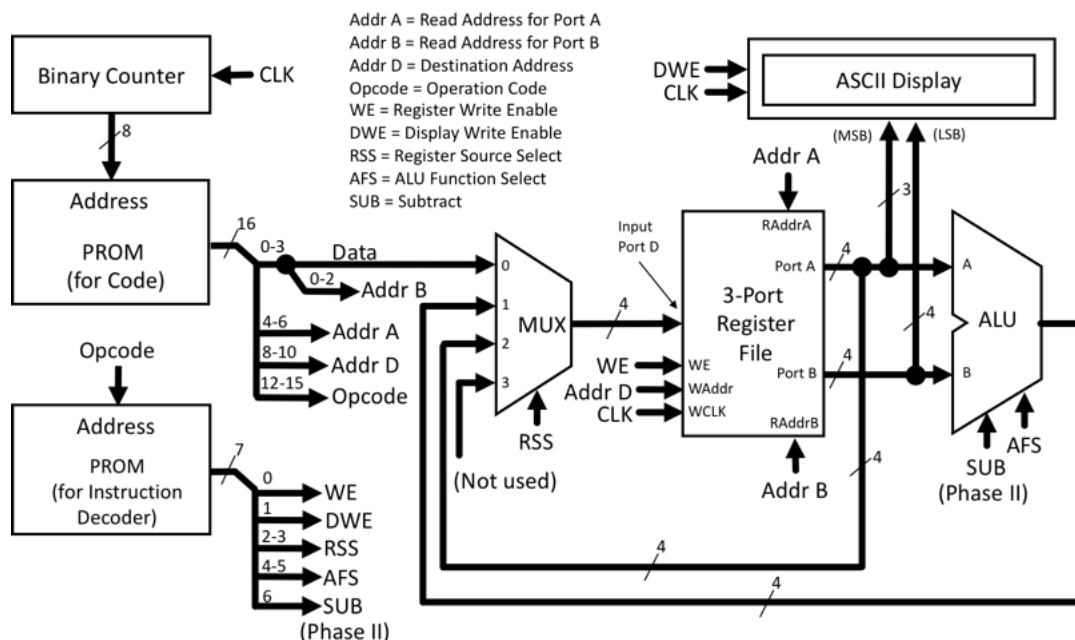
Purpose:

1. Review Verilog coding constructs learned throughout the semester
2. Be able to program a simple 4-bit processor in machine language
3. Enjoy the excitement of having created a simple 4-bit processor

Overview

For ECEN 160, the final project consists of designing and implementing a simple four-bit processor. Your final project for ECEN 160L will be to implement a similar processor using Verilog and the Digilent Basys 3 board that has an Atrix-7 FPGA (part xc7a35tcpg236-1).

The ECEN 160 version is described in the “Final Project Design Notes” item in I-Learn. The block diagram as given in ECEN 160 is:



You will be given a Vivado project that includes the following modules: Program_ROM, Instruction_Decoder, debouncer, and 7segment. The project also includes an outline of the other needed modules as well as code that provides a clock for the project that may be set at various speeds. This variable-clock code is built into the finalProject module. To make use of this clock you are required to use the signal called “clock” as the clock for your CPU.

You are *not* required to use the provided project. If you start a new project and want to use the display controller and debouncer modules, you may unzip the zipped-up project available on I-Learn and copy the sources of the modules you wish to use.

A video of a functioning ECEN 160L final project is available on I-Learn.

Grading

Grading of the final project will be done based on correct operation (90%) and construction quality (10%). [Note: this may be different than the ECEN 160 grading percentages.] Each individual student needs to demonstrate a working circuit to their instructor, or a qualified lab assistant, to receive full credit for correct operation. Construction quality will be based on code submitted using the following criteria:

- Is the code structured well?
- Does the code use good style?
- Is it the work of a professional?

Requirements and changes from ECEN 160 project

Implement the same functionality as the ECEN 160 project with the following exceptions:

- The display will be the four seven segment displays on the Basys3 board.
- Print “Hi, B/SX” instead of “Hello, B/SX” (Example: “Hi, BJ” for Brother Jones).
- You may do the $4-3=1$ subtraction problem in both software and hardware but you are only required to do one or the other. See the ‘Optional’ section at the end of this document.
- The project is to have an asynchronous reset button, which at a minimum, sets the program counter to address 0.

Notes and helps

- There will be two clocks in this project. The 100 MHz clock is used for the display, the debouncer and to generate a slower speed clock that has various speed settings. The lowest speed of the variable clock (inputs of 0 0 0 on the switches) is just a debounced version of the signal created by pushing the ‘Clk’ button (BTNR). This would usually be called the ‘single step’ clock. Use the output of the variable speed clock, which has the single step clock as an option, for the clock signal shown in the block diagram above.
- The modules given to you should be reviewed to understand how to use them.
- Depending on how you do the implementation of the Verilog version of the four-bit processor, the program you made for Logisim will work correctly with few or no modifications.
- Using the LEDs to display the contents of the register file can help in debugging your code.
- *Pay close attention to the synthesis warnings!*
- The Instruction_Decoder module allocates the top three bits of the decoder ROM output to ALU function select lines (see ‘Optional’). The “output [2:0] ALUFuncSel,” line shown below:

```
module Instruction_Decoder(  
    input [3:0] opcode,  
    output [2:0] ALUFuncSel,  
    output [1:0] regSourceSel,  
    output dispWE, regWE  
);
```

If you only use two bits for selecting ALU functions, as was done for the ECEN 160 final project, you will get a warning. In this particular case, the warning is OK to have. In general, you should review very carefully any of these types of warnings. They may cause the optimizer to optimize out all most all of your design!

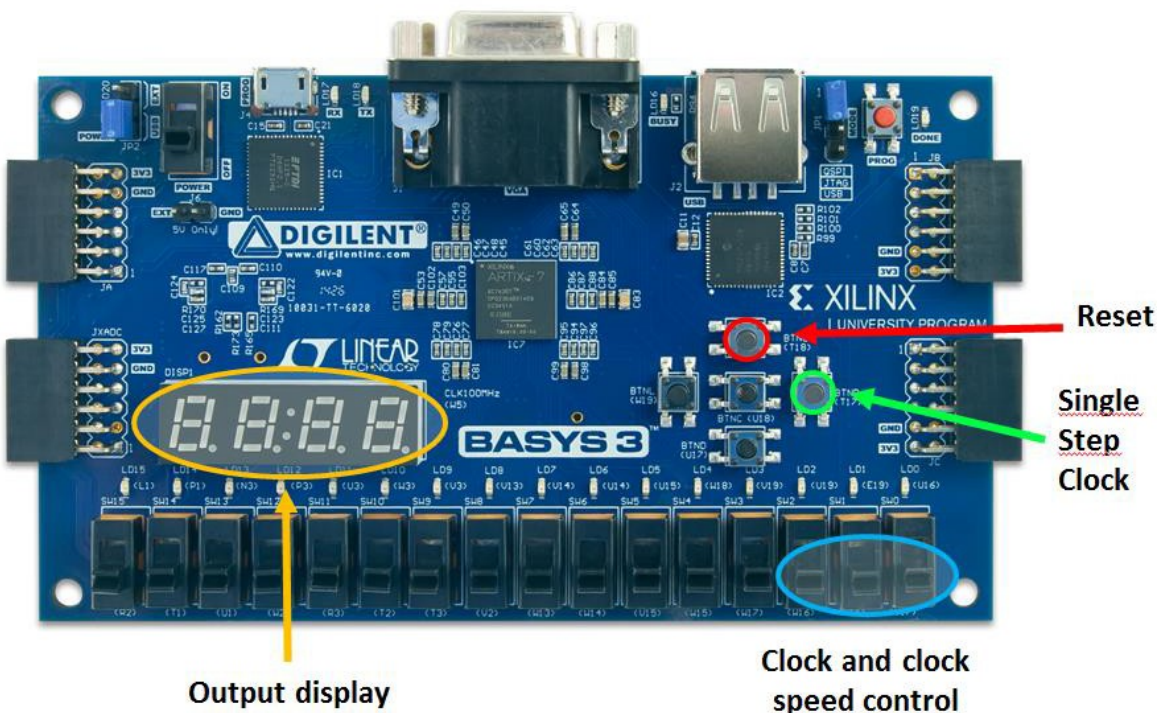
- **Warning!** If you synthesize your design with only zeros in the Program_ROM and Instruction_Decoder, the optimizer is so good that it will optimize out most all of your hardware (very little logic in the schematic view of the project)
- Depending on what you have in the ROM and decoder, the optimizer may just optimize out one or two bits! *Check for synthesis warnings!*

The following table provides the input and output assignments for the final project. A diagram with input and output locations is also given.

Input	Switch Assignment
Clk	BTNR
Reset	BTNU
3 switches for clock speeds	SW2 – SW0

Output	LED Assignment
Seven segment displays	Seg[6:0], an[3:0], dp

Final Project



[Handout continued on the next page.]

```

module finalProject(
    input clk,                // Push button for single stepping processor - button R
    input reset,              // Push button for asynchronous reset - button U
    input [2:0] clkSpeed,     // Lower three switches determine clock to use
    input clk100MHz,          // The 100 MHz on-board clock
    output [6:0] seg,         // Outputs to use the seven segment displays
    output [3:0] an,
    output dp
);

// Wires, busses and registers
.
.
.
reg [26:0] clkDiv;           // A 27 bit counter to divide down the 100 MHz
reg clock;
wire deBncClk;

// Debounce the single step push button clock input
debouncer DB0(.clk100MHz(clk100MHz), .rst(1'b0), .btn(clk), .debBtn(deBncClk));

// Use the counter to divide down the 100 MHz clock
always @(posedge clk100MHz)
    clkDiv <= clkDiv + 1;

// Select the clock speed to use for the CPU, based on the three lower switches.
// When all switches are off, select the pushbutton switch.
always @(clkSpeed, deBncClk, clkDiv)
    case(clkSpeed)
        0: clock = deBncClk;
        1: clock = clkDiv[26];
        2: clock = clkDiv[25];
        3: clock = clkDiv[24];
        4: clock = clkDiv[23];
        5: clock = clkDiv[22];
        6: clock = clkDiv[21];
        7: clock = clkDiv[20];
    endcase

// Instantiate all modules
Program_counter PCNT(clock, );
//Add port names that correspond with the ports in your Binary Counter
Program_ROM PRGROM( );
//Add port names that correspond with the ports in the program PROM module
Instruction_Decoder InstDec( );
//Add port names that correspond with the ports in the instr. decoder module
alu ALU( );
//Add port names that correspond with the ports in your ALU module
RegFile Reg( );
//Add port names that correspond with the ports in your RegFile module
sevenSegment SEVSEG({ ... , ... }, ... , clock, clk100MHz, seg, an, dp);
//You can concatenate Port A and Port B to create
//your ASCII code to send to the display

```

```

// Add your code for the Register input MUX

endmodule

// Sequential logic for Binary Counter
module Program_counter(
    input clk,
    ...
);

    always @ (posedge clk ...)
    ...

endmodule

// Combinational logic for ALU
module alu(
    ...
);

    ...

endmodule

// 3-port Register File
module RegFile(
    ...
);

    ...


endmodule

```

[Handout continued on the next page.]

Seven Segment Display Controller Characters

					Upper Data Nibble															
					DB7	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
					DB6	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
					DB5	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
					DB4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
Lower Data Nibble	XXXXX	0000	null																	
	XXXXX	0001																		
	XXXXX	0010																		
	XXXXX	0011																		
	XXXXX	0100																		
	XXXXX	0101																		
	XXXXX	0110																		
	XXXXX	0111																		
	XXXXX	1000	BS																	
	XXXXX	1001	tab																	
	XXXXX	1010	LF																	
	XXXXX	1011	VT																	
	XXXXX	1100	FF																	
	XXXXX	1101	CR																	
	XXXXX	1110																		
	XXXXX	1111																		

Note: Sending the code for any of the grey spaces will display the following invalid character symbol: 

Optional

You might wish to modify the ALU to do a hardware subtract function. If you do, it is suggested that you implement eight ALU functions. The additional four functions would be: subtract, ~ (bit wise negation), ! (logical NOT) and multiplication.