Lab 12-Processor Datapath

Name: Robert Bara

Section #: 003

Date: 4/16/2020

Summary/Abstract (10 points)

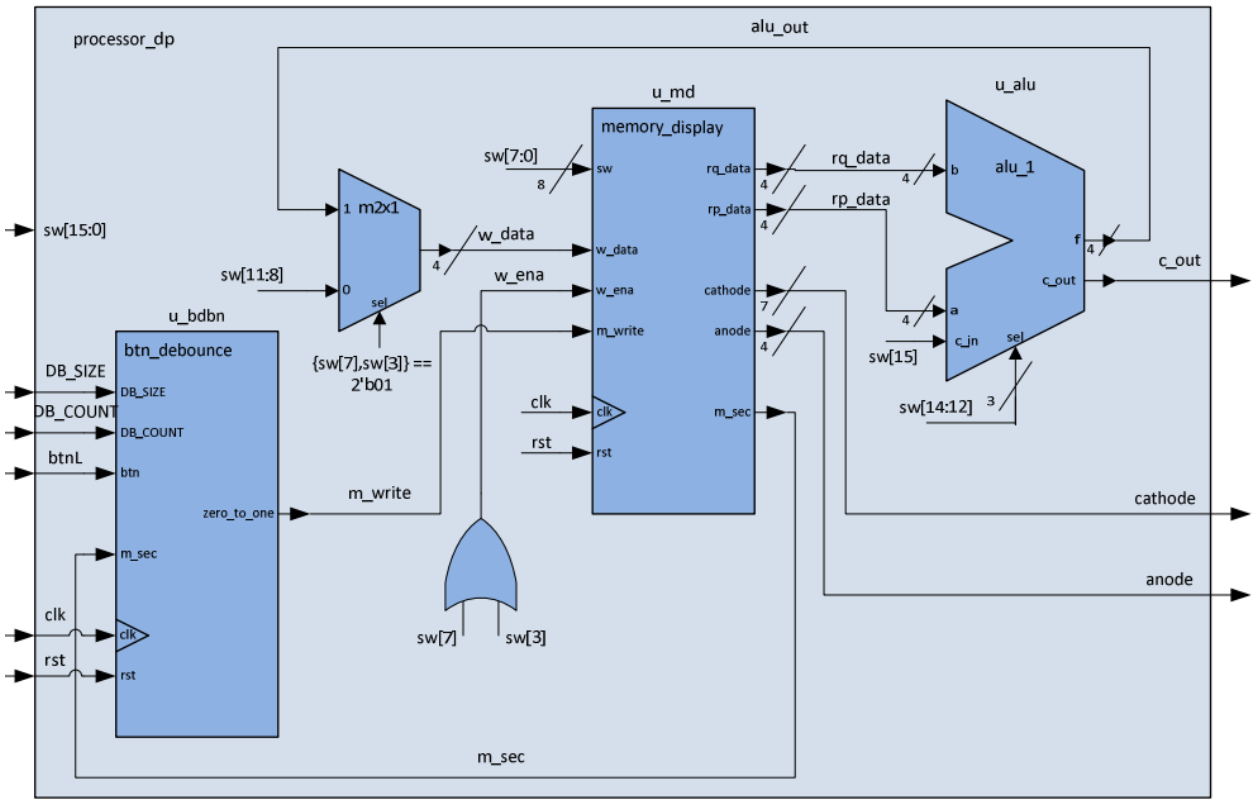
This lab takes the previous ALU created in Lab6, with the 2-read, 1-write register file from lab 11 to create a Datapath for a 4-bit microprocessor. The processor will then be uploaded to the Basys3 Board and will store my TU ID to perform various arithmetic.

Introduction (10 points)

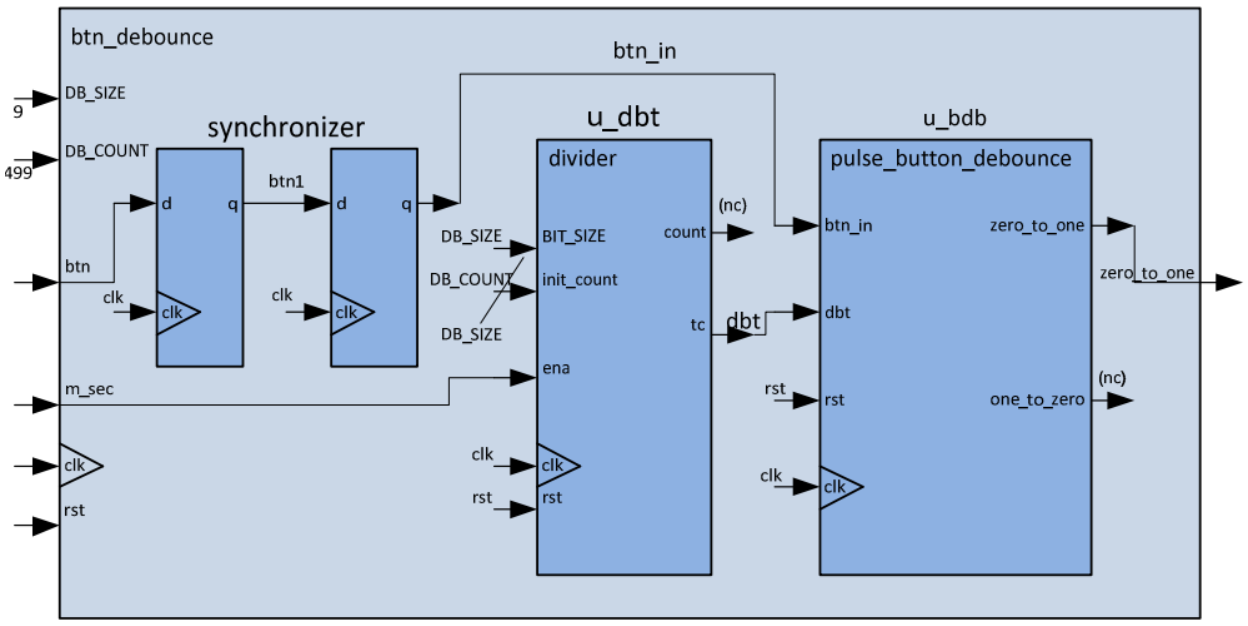
Processors are the backbone of computer memories so learning how to create a simple microprocessor serves as an introduction to Processors which is essential to ECE work. This lab will use Verilog instantiations and an assign statement to create a multiplexer. Essentially the processor follows the basic Datapath of reding from the register to a sign extend which will then use the ALU to write to data to the memory and update the register. This processor will perform operations with addresses P and Q and update the memory into P.

Procedure (15 points)

The processor block diagram below shows how the modules from previous labs will be instantiated in Verilog. Btn_debounce is essentially a FSM module that will be provided, along with its own block diagram. The btn_debounce will be connected to the left pushbutton and signals a single memory write by providing a one-clock pulse width output for a zero to one and/or a one to zero push button event. The output signal will then drive the 4-digit seven segment display and a carry out signal from the ALU which will be connected to LED[15]. The divider module is also used to divide m_sec by 500 to achieve a 0.5 second debounce time, further clarification of this can be seen using GTKWave. A 2x1 multiplexer will also be used to input write data into the memory display, this can be done using an assign statement, and an OR gate can also be used to determine the w_ena input to the divider module. Personally, I chose to use another assign statement over an OR gate, although this makes no difference.



[Figure 1A: Processor_dp block diagram, the datapath]



[Figure 1B: Btn_debounce block diagram]

The table below will provide further detail about the ALU operation and hardware instantiation:

Function: reset	
reset	center pushbutton (***rst*** input signal in ***memory_display***)
Function: write data to memory	
Inputs:	
Select write function	switch 3 off and switch 7 – on (***w_ena*** input signal in ***memory_display***)
Write address	3-bits: switches 2-0
Write data	4-bits: switches 11-8 (***w_data*** input signal in ***memory_display***)
Command: write data to address	left pushbutton (***m_write*** input signal in ***memory_display***)
Outputs:	
Digit 0	3-bits: hexadecimal, address to write to (switches 2-0)
Digit 1	4-bits: hexadecimal, data to write (switches 11-8)
Digit 2	(same as read function below)
Digit 3	(same as read function below)
Function: read data from memory	
Inputs:	
Select read function	switch 3 off and switch 7 – off (***w_ena*** input signal in ***memory_display***)
Read address, port p	3-bits: switches 2-0
Read address, port q	3-bits: switches 6-4
Outputs:	
Digit 0	3-bits: hexadecimal, address p
Digit 1	4-bits: hexadecimal, data read from address p
Digit 2	3-bits: hexadecimal, address q
Digit 3	4-bits: hexadecimal, data read from address q
Function: ALU operation – read data from memory while updating with ALU result	
Inputs:	
Select read register and update/write using ALU function	switch 3 – on and switch 7 – off
Read address, port p; and destination write address	3-bits: switches 2-0
Read address, port q	3-bits: switches 6-4
Command: update ALU - result to register pointed to by port p	Left button
Command: ALU function	switches [14:12] (alu_sel – see your ALU design) - 000 -> f = a (no operation) - 001 -> f = a + b + c_in - 010 -> f = a + b with no carry - 011 -> f = a – b – c_in - 100 -> f = a – b with no borrow - 101-> f = a + 1 - 110 -> f = a – 1 - 111-> f = a & b (bit by bit logical AND)
Outputs:	
Digit 0	3-bits: hexadecimal, address p; and destination write address
Digit 1	4-bits: hexadecimal, data read from address p
Digit 2	3-bits: hexadecimal, address q
Digit 3	4-bits: hexadecimal, data read from address q

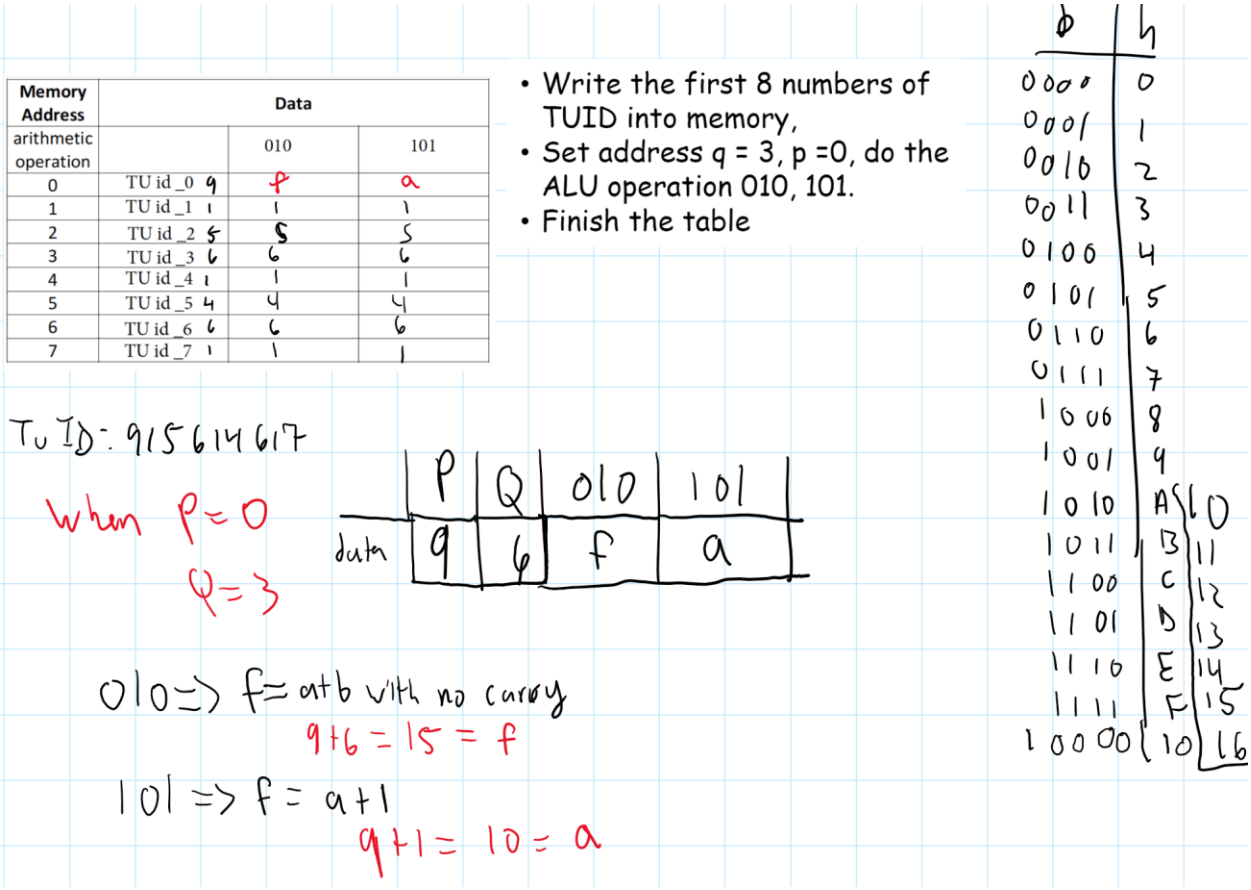
[Figure 2A: Functionality of the Basys3 board with processor instantiation]

Using this table, it is very easy to read and write to the board. I uploaded my TU ID using switch 7 to write, switches [2:0] to select which memory address to write to, and switches[11:8] to write data to the

selected address, with the left push button saving to the selected address. Then to perform the ALU operation, switch 7 is turned off and switch 3 is turned on. Switches [14:12] controls which ALU operation will be done. Switches [2:0] control what is stored in P and will display the address in digit 0, with the hexadecimal data in digit 1. Likewise, switches [6:4] controls Q with digit 2 displaying the address and digit 3 displaying the data. After pushing left push button, the address will be updated.

Results (45 points – see sub-sections)

I found this lab to be quite simple, the only part that threw me for a loop was designing the 2x1 multiplexer since in past labs I usually created multiplexers using if or case statements. This lab required an assign statement using the '?' conditional, after I corrected this I was able to achieve no mismatches. From this, I used the following logic below to import my 9 digit TU ID: 915614617 and figure out what will be updated into the 8bit memory.



Memory Address	Data		
arithmetic operation		010	101
0	TU id_0 9	f	a
1	TU id_1 1	1	1
2	TU id_2 5	5	5
3	TU id_3 6	6	6
4	TU id_4 1	1	1
5	TU id_5 4	4	4
6	TU id_6 6	6	6
7	TU id_7 1	1	1

[Figure 2B: Logic behind the hardware implementation/simulation]

After figuring this out I was able to get my board to match my results and get checked off.

Design Code (15 points)

processor_dp.sv

```
//
// lab12 : version 04/13/2020
// Robert Bara
module processor_dp #(parameter DB_SIZE = 9, DB_COUNT = 499)
    (output logic [6:0] cathode, output logic [3:0] anode,
    output logic c_out, input logic clk, input logic rst, input logic [15:0] sw,
    input logic btnL);
    logic m_write, m_sec, w_ena;
    logic [3:0]rp_data,rq_data,w_data,alu_out;

    assign w_ena=(sw[7]||sw[3]); //or gate

    //instantiations
    memory_display u_md(.m_sec,.anode,.cathode,.rp_data(rp_data),.rq_data(rq_data),.sw(sw[7:0]),.w_data(w_data),.m_write,.w_ena,.clk,.rst);
    alu_1 u_alu(.c_out(c_out),.f(alu_out),.b(rq_data),.a(rp_data),.c_in(sw[15]),.sel(sw[14:12]));
    btn_debounce#(.DB_SIZE(DB_SIZE),.DB_COUNT(DB_COUNT)) u_bdln(.zero_to_one(m_write),.btn(btnL),.m_sec(m_sec),.clk,.rst);
    //2x1 multpilexer
    assign w_data=({sw[7],sw[3]}==2'b01)?alu_out:sw[11:8];
endmodule
```

Simulation Results (15 points)

```
Simulation complete - no mismatches!!!
$finish called at time : 136000005 ns : File "/home/tuj22026/2613_2020s/lab12/tb_processor_dp.sv" Line 145
```

Hardware Implementation (10 points)

My design was checked off by Yamin on 4/16/2020 at 1:25pm during the lab period.

Conclusion (10 points)

This lab took together a lot of what was worked on throughout the semester and provides a brief introduction to Dr. Heferty's Processor course. Overall, I was able to use past modules that use logic designs such as a seven-segment decoder, parameterized divider using synchronous logic, and finite state machine to create a datapath for a microprocessor. The rest of the syntax was accomplished using assign statements and I was able to achieve no mismatches and successfully implement my design to the Basys3 board.