Lab 5 – Combinational Shift Logic: Up/Down/Arithmetic/Circular/Barrel

Robert Bara

Section #: 003

Date: 2/20/2020

Summary/Abstract (10 points)

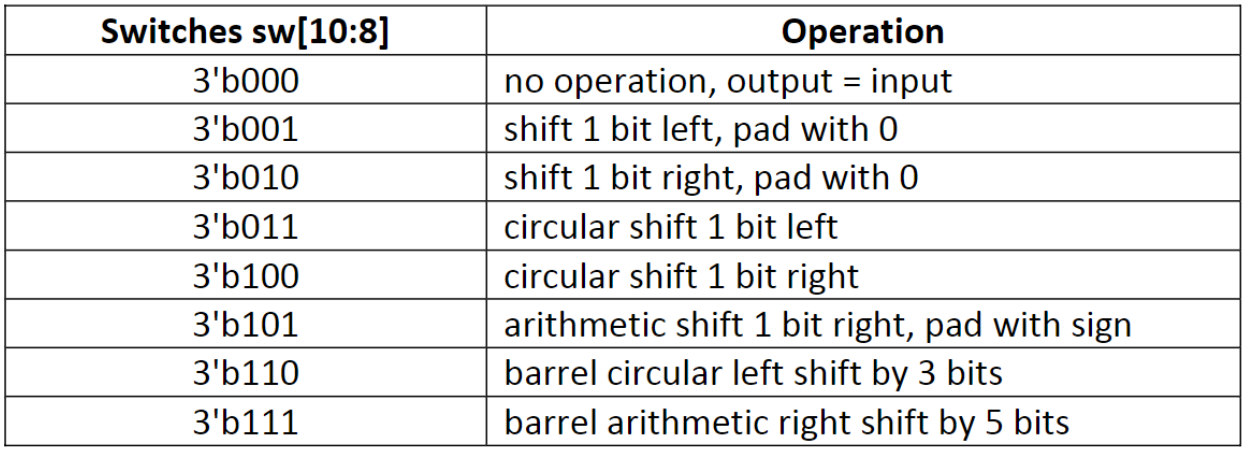
Lab 5 uses combinational shift logic to take an 8bit input on the Basys3 board from switches [7:0] that correspond to LEDs [7:0] and shift them in various combination patterns depending on the 3bit input from the selector switches [10:8].

Introduction (10 points)

Some relevant background information for this lab includes understanding different kinds of combinational logic such as the arithmetic shift, the barrel shift, and circular shift. The procedure will explain more on how some of these shifts work. It is essential to know how signed vs unsigned binary operates based on the most significant bit because this will change the algorithm of how the input is shifted. Using combinational logic is important and practical to software design because if you are designing hardware that takes in an input and needs to be updated, such as something that counts up like a timer, combinational shifts can move the original input over and pad bits with the updated bits. Concatenation as well as the arithmetic shift operators are essential Verilog syntax for this lab, and a case statements simplifies the conditions and implements the corresponding truth table.

Procedure (15 points)

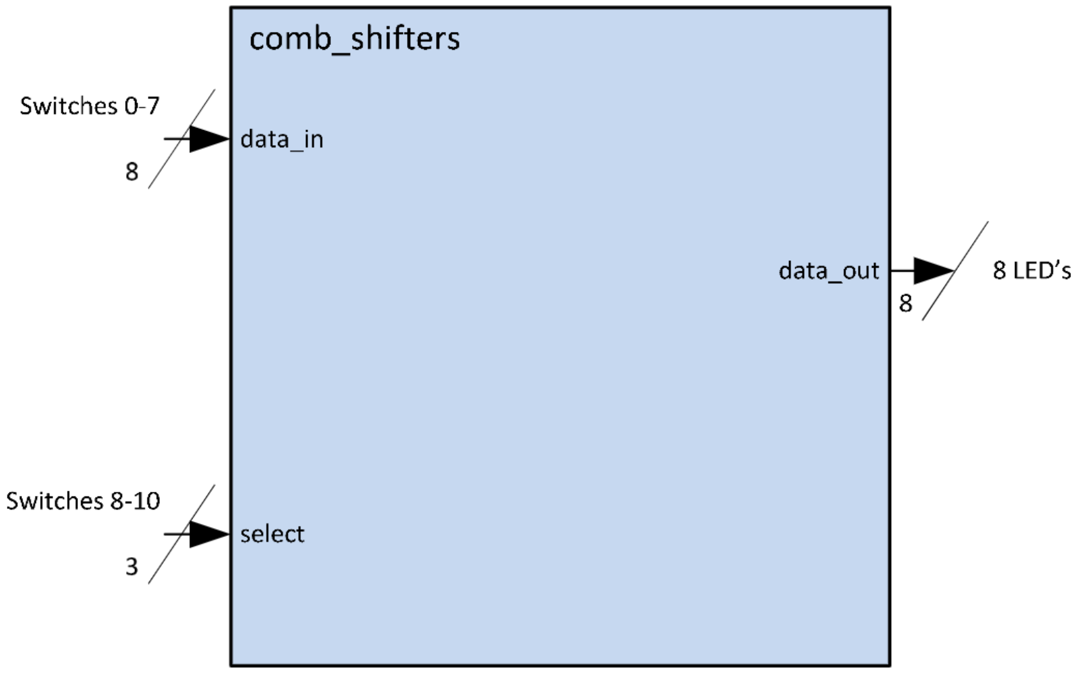
The first thing to understanding this lab is to understand combinational logic shifts. Arithmetic shifts will simply move the input over left or right to start at a different bit and will pad the bits that get cut off with the most or least significant bit unless stated otherwise. Barrel Arithmetic shifts are when more than one bit is shifted arithmetically. A Circular shift will act similar to arithmetic shift and move bits either individually or barreled, but it will pad the shifted positions with the remaining bits that would get cut off in an arithmetic shift. The truth table below explains how the selector switches will shift the original input:



Switches sw[10:8]	Operation
3'b000	no operation, output = input
3'b001	shift 1 bit left, pad with 0
3'b010	shift 1 bit right, pad with 0
3'b011	circular shift 1 bit left
3'b100	circular shift 1 bit right
3'b101	arithmetic shift 1 bit right, pad with sign
3'b110	barrel circular left shift by 3 bits
3'b111	barrel arithmetic right shift by 5 bits

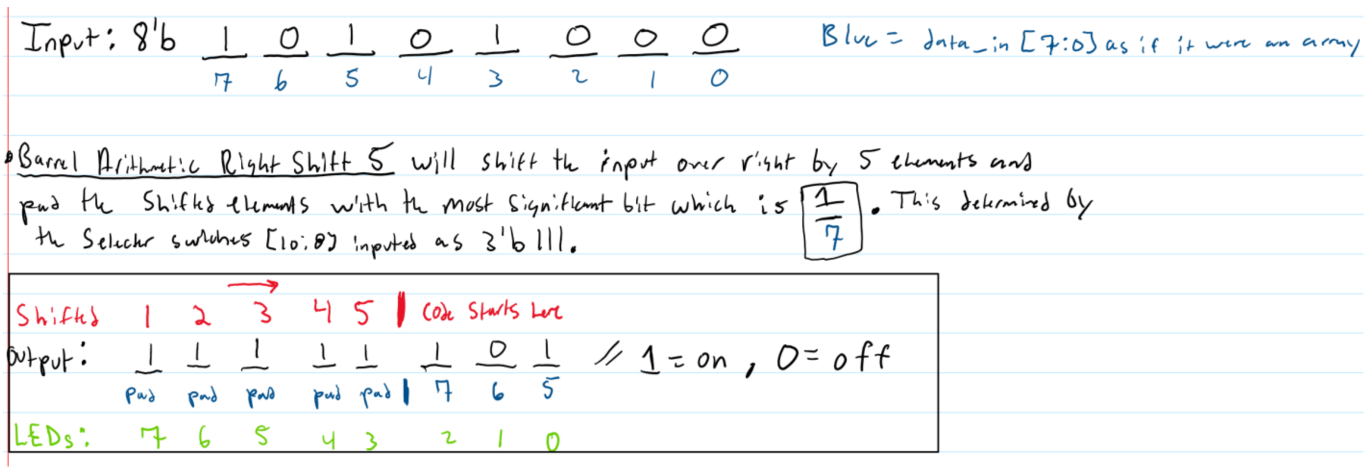
[Figure 1: Truth Table from Lab Manual]

The module comb_shifters will be created and instantiated based on the block diagram below, which will take an input on switches (0:7) and light up the corresponding LEDs, and then take in an input on switches (10:8) that will correspond to the truth table and shift the bits. Even though switches (10:8) will shift the input bits to different LEDs, switches (0:7) will still correspond to the original input, so for example if the original bit that is on in position data_in[4], then switch [4] will be turned on, but if that bit changes LEDs and is shifted into the position of LED [1], switch [4] will still control if that bit is turned on or off, which means in that case, switch [4] would turn on or off LED [1], unless the selector switches [10:8] are changed.



[Figure 1b: Block Diagram from Lab Manual]

To help understand the logic behind coding, an example of how the cases for a barrel arithmetic right shift by 5 and circular arithmetic left shit by 3 is drawn out below, if the given input is 8'b10101000:



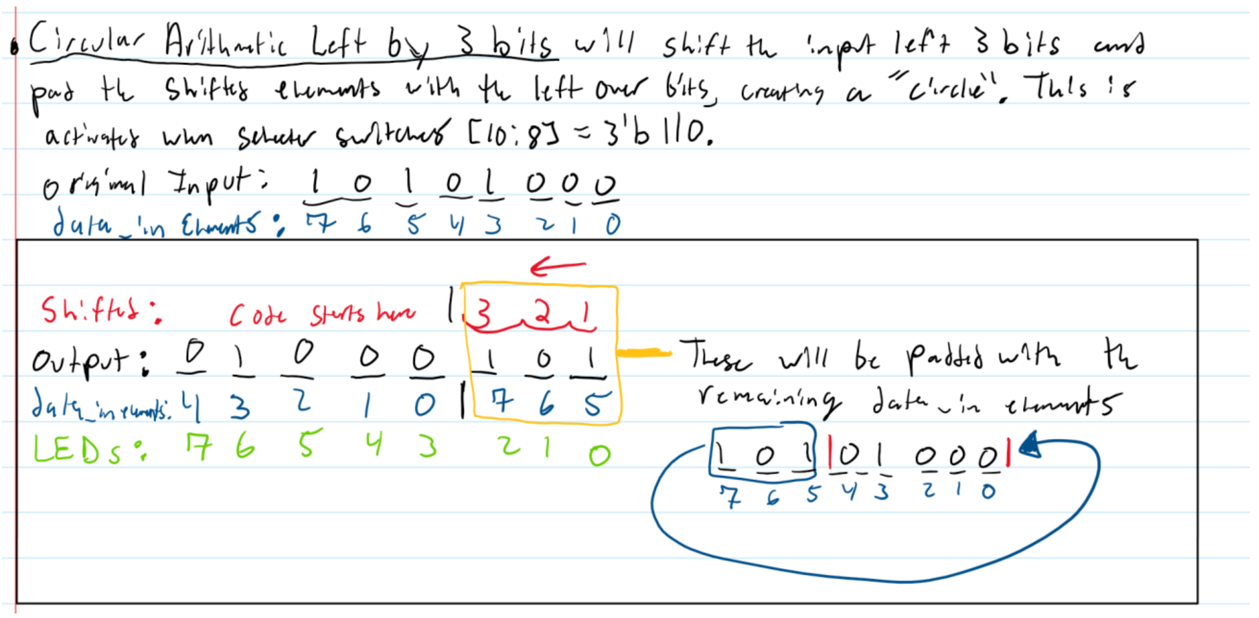
Input: 8'b 1 0 1 0 1 0 0 0

Blue = data_in [7:0] as if it were an array

• Barrel Arithmetic Right Shift 5 will shift the input over right by 5 elements and pad the Shifted elements with the most Significant bit which is $\frac{1}{7}$. This determined by the Selector switches [10:8] inputed as 3'b 111.

7 6 5 4 3 2 1 0
Shifts 1 2 3 4 5 | Code Starts here
Output: 1 1 1 1 1 | 1 0 1 // 1 = on, 0 = off
Pad Pad Pad Pad Pad | 7 6 5
LEDs: 7 6 5 4 3 2 1 0

[Figure 2: Logic Behind Case 3'b111]



• Circular Arithmetic Left by 3 bits will shift the input left 3 bits and pad the Shifted elements with the left over bits, creating a "circle". This is activated when Selector switches [10:8] = 3'b 110.

original Input: 1 0 1 0 1 0 0 0
data_in Elements: 7 6 5 4 3 2 1 0
Shifts: Code Starts here | 3 2 1
Output: 0 1 0 0 0 | 1 0 1
data_in elements: 4 3 2 1 0 | 7 6 5
LEDs: 7 6 5 4 3 2 1 0
These will be padded with the remaining data_in elements
1 0 1 | 0 1 0 0 0
7 6 5 4 3 2 1 0

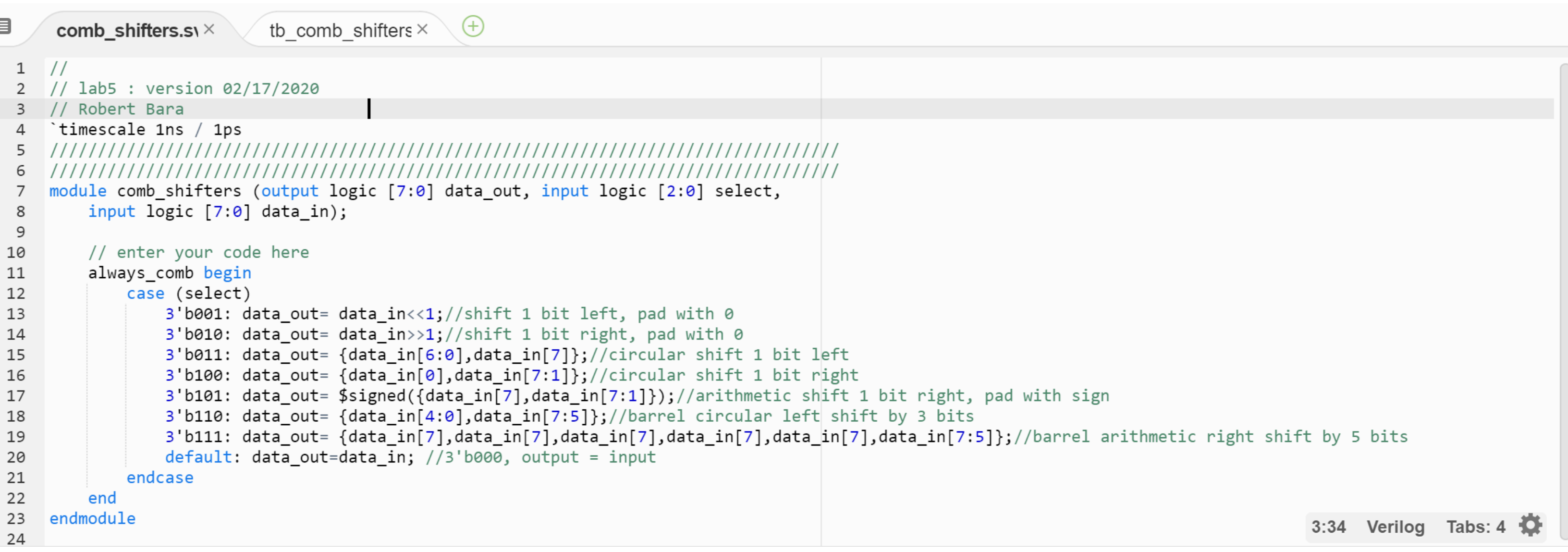
[Figure 2b: Logic Behind Case 3'b110]

Further cases such as the padding with sign can be drawn similarly by finding the decimal of the input and the decimal number of the shift which will determine whether the padding will be a 1'b0 or 1'b1 due to signed binary to decimal conversion.

Results (45 points – see sub-sections)

The hardest part about understanding this lab was understanding the logic behind it. The syntax for coding for not too difficult, but I had to draw out what exactly was being shifted so that way I knew how to index the switches of data_in within the case statement. I was able to achieve a simulation with no mismatches by creating a case statement that corresponds to the truth table and uses concatenation as well Verilog's unsigned shift operators to shift the given input.

Design Code (15 points)

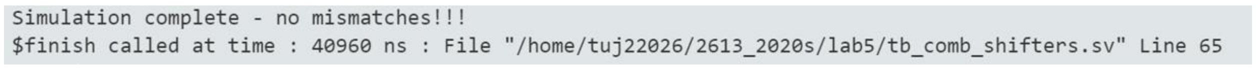


```
//
// lab5 : version 02/17/2020
// Robert Bara
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////
module comb_shifters (output logic [7:0] data_out, input logic [2:0] select,
    input logic [7:0] data_in);

    // enter your code here
    always_comb begin
        case (select)
            3'b001: data_out= data_in<<1;//shift 1 bit left, pad with 0
            3'b010: data_out= data_in>>1;//shift 1 bit right, pad with 0
            3'b011: data_out= {data_in[6:0],data_in[7]};//circular shift 1 bit left
            3'b100: data_out= {data_in[0],data_in[7:1]};//circular shift 1 bit right
            3'b101: data_out= $signed({data_in[7],data_in[7:1]});//arithmetic shift 1 bit right, pad with sign
            3'b110: data_out= {data_in[4:0],data_in[7:5]};//barrel circular left shift by 3 bits
            3'b111: data_out= {data_in[7],data_in[7],data_in[7],data_in[7],data_in[7],data_in[7:5]};//barrel arithmetic right shift by 5 bits
            default: data_out=data_in; //3'b000, output = input
        endcase
    end
endmodule
```

[Figure 3: module comb_shifters code]

Simulation Results (15 points)



```
Simulation complete - no mismatches!!!
$finish called at time : 40960 ns : File "/home/tuj22026/2613_2020s/lab5/tb_comb_shifters.sv" Line 65
```

[Figure 3b: Simulation of comb_shifters]

Hardware Implementation (10 points)

My design was demonstrated to Franka on Thursday 2/20/2020 at 2pm during the lab period.

Conclusion (10 points)

Using combination logic, I was able to create a case statement corresponding to the truth table which depending on the selector switches [10:8], the input was shifted in a unique way. To complete the code, I broke down the logic by writing out the input and sorting it based on the logic and data_in element position they were in. I padded the remaining bits depending on which type of shift was done either by padding with the most significant or least significant bit or padding with the remaining bits by creating a circular shift. Implementing this in Verilog, I used the shifting operators for linear arithmetic shifting, and used concatenation statements to accomplish barrel shifting and specific padding. For case 3'b101 I also padded with a sign by type casting with Verilog's syntax for signed binary. Uploading this into the Basys3 board was successful and I was able to shift the LEDs depending on the select switches and turn the LEDs on or off by using switches [7:0] to input the given input.