Noah Khan

April 22, 2022

8-Bit Full Adder

Description:

For this lab, I used multiplexers to implement combinational logic. I organized my logic using bit vectors (buses) in Verilog. This circuit added 0-3 to an 8-bit binary number and displayed the sum on the two right most digits on the seven-segment display. The logic utilized switches 0 – 7, btnL and btnC. Where the switches display the 8-bit digit and btnL/btnC adds to the 8-bit number.

Methods:

I began by creating the 3 multiplexer modules: x8 2-to-1, 4-to-1, and an 8-to-1. All of these modules used similar logic. Where S_0 represents the select lines and the F represents the output depending on the select line value.

Next, I created the full adder module. This module inherited the 4-to-1 module twice. Similar to Lab 2. The first instance took {cin, ~cin, ~cin, cin} as the input, {a, b} for the select lines, and s for the output. The second instance took {1'b1, cin, cin, 1'b0} as the input, {a, b} for the select lines, and cout for the output. This, I believe create a single bit adder. The diagram used can be found in appendix B (notes).

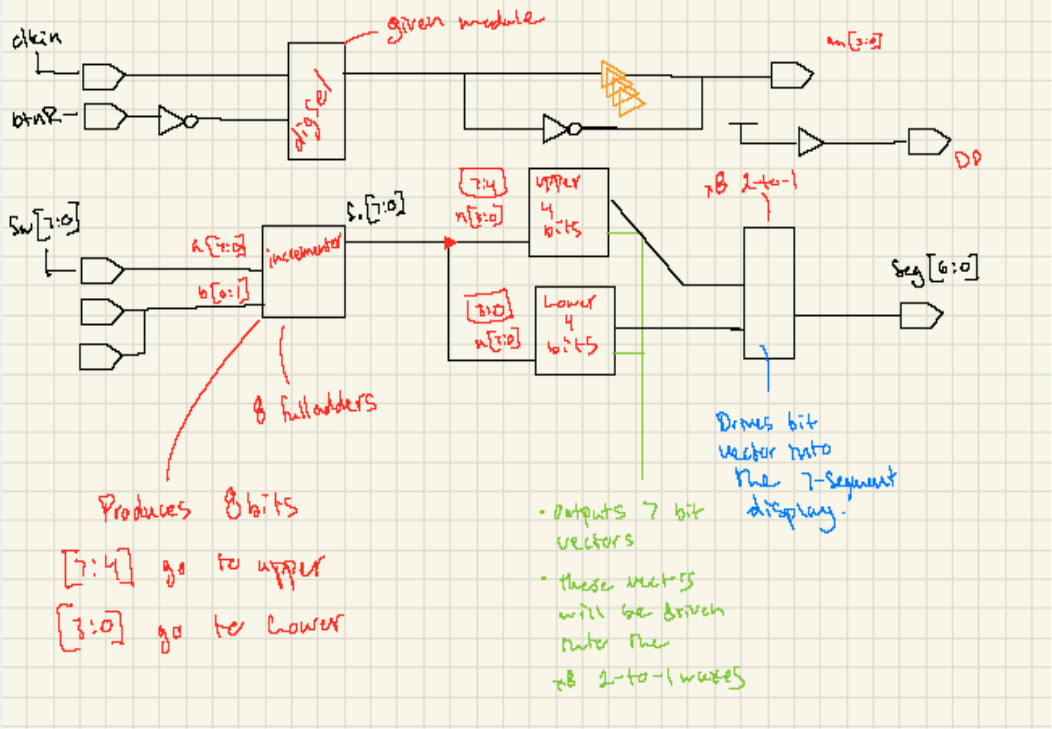
Next, I created the incrementer module. This module carried the task of being a ripple carry 8-bit adder. Where the full adder module was instantiated 8 times. Each of their "a" input was a bit from the 8-bit "a" vector. For the first 2 instances, b was b[0] and b[1] for the respective adders. For the first adder, cin was a 0 bit and for the rest it was cout form the preceding adder. Each adder output a bit into the 8-bit s vector, the bit's index corresponded with the number of the adder.

The second to last module created was the hex7seg. This module was responsible for the LEDs displayed on the seven-segment. The truth table used was the same as the truth table in Lab 2.

The final module was the top-level module, which calls the incrementer, hex7seg twice, and the 8x 2-to-1 modules. In addition, this module was in charge of turning on the appropriate LEDs on the seven-segment board.

Results:

Design:



2-to-1 Truth Table

S_0	F
0	I_0
1	I_2

Boolean Expression

$$I_0 \sim S + I_1 S$$

This is the logical expression for the 2-to-1 truth table. Where the ~ represents “not.” The other modules followed the same logic. The number of inputs will always be 2 to the power of the number of select lines. The logic is similar with the higher input multiplexers.

Hex7seg Truth Table

n_3	n_2	n_1	n_0	CA	CB	CC	CD	CE	CF	CG
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	1	0	0
1	0	1	0	0	0	0	1	0	0	0
1	0	1	1	1	1	0	0	0	0	0
1	1	0	0	0	1	1	0	0	0	1
1	1	0	1	1	0	0	0	0	1	0
1	1	1	0	0	1	1	0	0	0	0
1	1	1	1	0	1	1	1	0	0	0

Where n_{3-1} were used for the select lines and n_0 was used to determine whether the segment was on or off for a certain digit. Diagrams can be found in appendix B (notes).

I designed a single module at a time and got inspiration from Lab 2.

Testing & Simulation:

I tested my design by creating a testbench that tests the top-level module, similar to Lab 2. For this testbench, I created several test cases to test whether the switches output the appropriate number. I chose these inputs because they covered all the cases from 0000 0000 – FFFF FFFF. I believe there were a few corner cases, these cases include the btnL and btnC buttons. I checked to see the buttons worked individually, together and if they generated overflow. There were no problems that I can recall.

Lab Questions:

The dig_sel was osciallting at a frequency of 3.05 kHz.

I did not notice any flickering on the seven-segment display.

Conclusion:

From this lab, I learned how to use multiplexers for combinational logic, utilize bit vectors and display numbers on the seven-segment display. I encountered a few difficulties while doing this lab. I had an issue with my logic for my full adder, hex7seg and my incrementer. For my full adder, I originally used "a" for the input instead of c_{in}. For the incrementer I had the wrong inputs and outputs. I did not properly utilize all the inputs and outputs for this module at first. I later went back and changed the logic around. For the hex7seg module, I had the right idea for the logic, but it was backwards. I had to rearrange the inputs for the 8-to-1 mux. If I could redo this lab, I would test each module individually. Doing this would have saved me time. There aren't any components that come to mind when I think of optimization.

Appendix A

x8 2-to-1 Multiplexer

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/11/2022 07:38:24 PM
// Design Name:
// Module Name: m2_1
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m2_1(
    input [7:0] in0,
    input [7:0] in1,
    input sel,
    output [7:0] o
    );

    assign o[0] = (in0[0] & ~sel) | (in1[0] & sel);
    assign o[1] = (in0[1] & ~sel) | (in1[1] & sel);
    assign o[2] = (in0[2] & ~sel) | (in1[2] & sel);
    assign o[3] = (in0[3] & ~sel) | (in1[3] & sel);
    assign o[4] = (in0[4] & ~sel) | (in1[4] & sel);
    assign o[5] = (in0[5] & ~sel) | (in1[5] & sel);
    assign o[6] = (in0[6] & ~sel) | (in1[6] & sel);
    assign o[7] = (in0[7] & ~sel) | (in1[7] & sel);


endmodule
```

4-to-1 Multiplexer

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/11/2022 07:10:50 PM
// Design Name:
// Module Name: m4_1
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m4_1(
    input [3:0] in ,
    input [1:0] sel,
    output o
    );

    assign o = (in[0] & ~sel[1] & ~sel[0]) | (in[1] & ~sel[1] & sel[0]) | (in[2] &
sel[1] & ~sel[0]) | (in[3] & sel[1] & sel[0]);

endmodule
```

8-to-1 Multiplexer

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/11/2022 07:10:50 PM
// Design Name:
// Module Name: m8_1
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m8_1(

    input [7:0] in,
    input [2:0]sel,
    output o

    );

    assign o = (in[0] & ~sel[2] & ~sel[1] & ~sel[0]) | (in[1] & ~sel[2] & ~sel[1] &
sel[0]) | (in[2] & ~sel[2] & sel[1] & ~sel[0]) | (in[3] & ~sel[2] & sel[1] & sel[0])
| (in[4] & sel[2] & ~sel[1] & ~sel[0]) | (in[5] & sel[2] & ~sel[1] & sel[0]) |
(in[6] & sel[2] & sel[1] & ~sel[0]) | (in[7] & sel[2] & sel[1] & sel[0]);

endmodule
```

Full Adder

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/11/2022 10:39:10 PM
// Design Name:
// Module Name: fulladder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module fulladder(
    input a,
    input b,
    input cin,
    output cout,
    output s
    );

    m4_1 m1 ( .in({cin, ~cin, ~cin, cin}), .sel({a, b}), .o(s) );
    m4_1 m2 ( .in({1'b1, cin, cin , 1'b0}), .sel({a, b}), .o(cout) );

endmodule
```

Incrementer

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/11/2022 07:28:59 PM
// Design Name:
// Module Name: Incrementer
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Incrementer(
    input [7:0] a0,
    input [1:0] b0,
    output [7:0] s0
    );

    wire [7:0] c0;

    fulladder fa0 (.a(a0[0]), .b(b0[0]), .cin(1'b0), .cout(c0[0]), .s(s0[0]) );
    fulladder fa1 (.a(a0[1]), .b(b0[1]), .cin(c0[0]), .cout(c0[1]), .s(s0[1]) );
    fulladder fa2 (.a(a0[2]), .b(1'b0), .cin(c0[1]), .cout(c0[2]), .s(s0[2]) );
    fulladder fa3 (.a(a0[3]), .b(1'b0), .cin(c0[2]), .cout(c0[3]), .s(s0[3]) );
    fulladder fa4 (.a(a0[4]), .b(1'b0), .cin(c0[3]), .cout(c0[4]), .s(s0[4]) );
    fulladder fa5 (.a(a0[5]), .b(1'b0), .cin(c0[4]), .cout(c0[5]), .s(s0[5]) );
    fulladder fa6 (.a(a0[6]), .b(1'b0), .cin(c0[5]), .cout(c0[6]), .s(s0[6]) );
    fulladder fa7 (.a(a0[7]), .b(1'b0), .cin(c0[6]), .cout(c0[7]), .s(s0[7]) );

endmodule
```

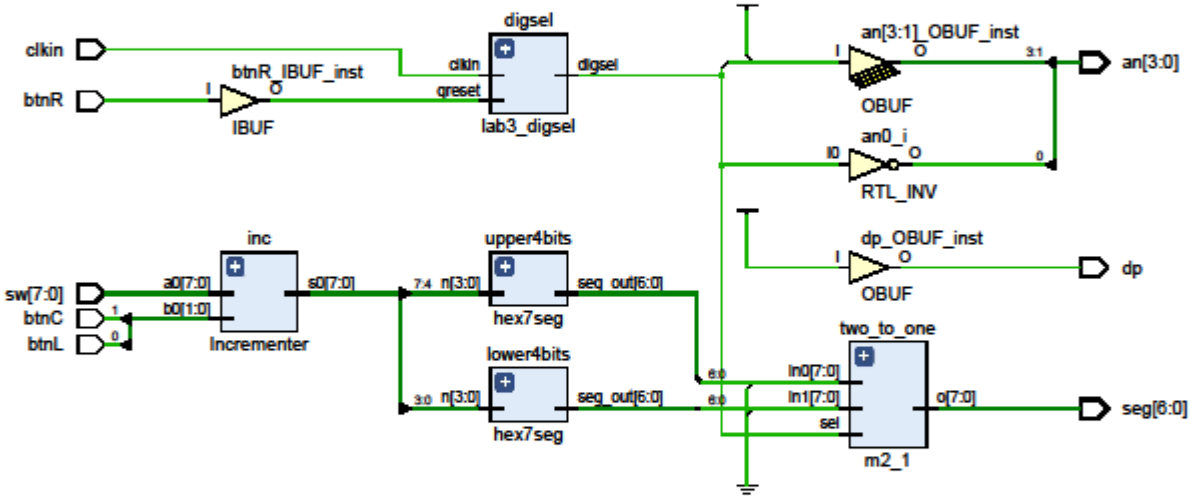
Hex7seg

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/12/2022 01:47:15 PM
// Design Name:
// Module Name: hex7seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module hex7seg(
    input [3:0] n,
    output [6:0] seg_out

    );


    wire not_n0;
    assign not_n0 = ~n[0];

    m8_1 sevensegment0(.sel(n[3:1]), .in({1'b0, n[0], n[0], 1'b0, 1'b0, not_n0,
1'b0, n[0]}), .o(seg_out[0]));
    m8_1 sevensegment1(.sel(n[3:1]), .in({1'b1, not_n0, n[0], 1'b0, not_n0, n[0],
1'b0, 1'b0}), .o(seg_out[1]));
    m8_1 sevensegment2(.sel(n[3:1]), .in({1'b1, not_n0, 1'b0, 1'b0, 1'b0, 1'b0,
not_n0, 1'b0}), .o(seg_out[2]));
    m8_1 sevensegment3(.sel(n[3:1]), .in({n[0], 1'b0, not_n0, n[0], n[0], not_n0,
1'b0, n[0]}), .o(seg_out[3]));
    m8_1 sevensegment4(.sel(n[3:1]), .in({1'b0, 1'b0, 1'b0, n[0], n[0], 1'b1, n[0],
n[0]}), .o(seg_out[4]));
    m8_1 sevensegment5(.sel(n[3:1]), .in({1'b0, n[0], 1'b0, 1'b0, n[0], 1'b0, 1'b1,
n[0]}), .o(seg_out[5]));
    m8_1 sevensegment6(.sel(n[3:1]), .in({1'b0, not_n0, 1'b0, 1'b0, n[0], 1'b0,
1'b0, 1'b1}), .o(seg_out[6]));
```

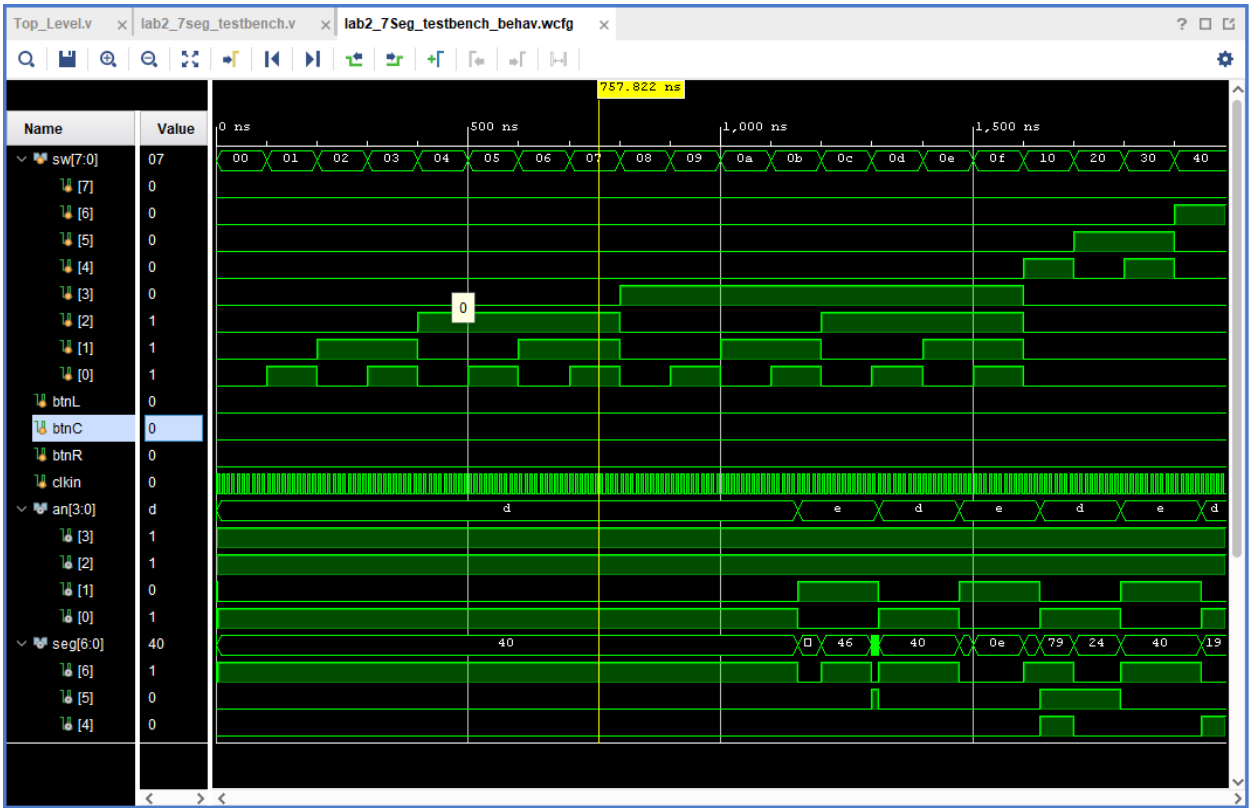
Top-Level

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/11/2022 08:45:23 PM
// Design Name:
// Module Name: Top_Level
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Top_Level(
    input [7:0] sw,
    input btnL,
    input btnC,
    input btnR,
    input clkin,
    output [6:0] seg,
    output dp,
    output [3:0] an
    );


    wire dig_sel;
    wire [7:0] inc_out;
    wire [6:0] seg_out1;
    wire [6:0] seg_out2;

    assign an[3] = 1'b1;
    assign an[2] = 1'b1;
    assign dp = 1'b1;
    assign an[0] = ~dig_sel;
    assign an[1] = dig_sel;
```

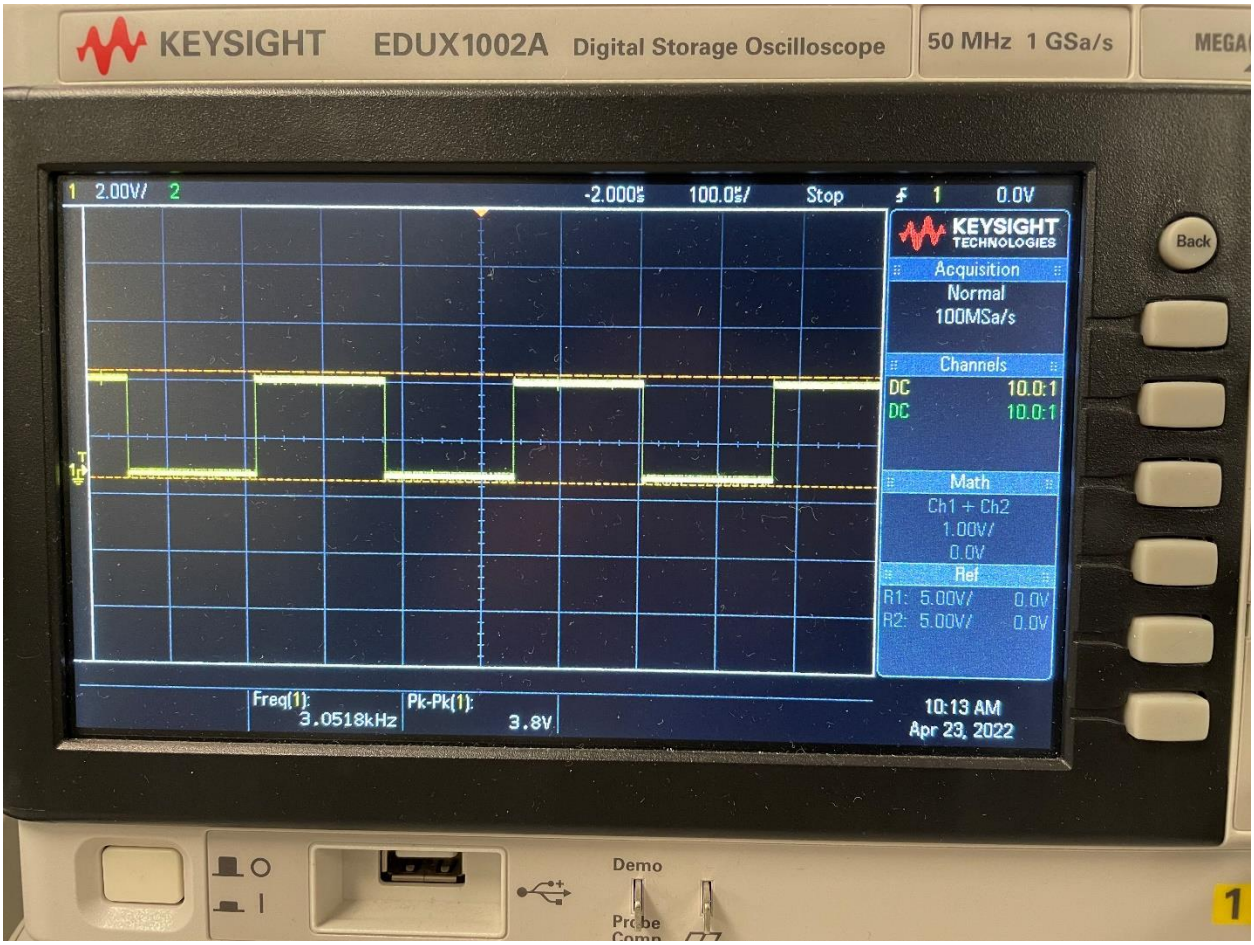
Elaborated Design


clkin
btnR
btnR_IBUF_inst
I
O
IBUF
digsel
clkin
greset
digsel
lab3_digsel
an[3:1]_OBUF_inst
I
O
3:1
OBUF
an[3:0]
an0_i
I0
O
0
RTL_INV
dp_OBUF_inst
I
O
OBUF
dp
inc
sw[7:0]
btnC
btnL
a0[7:0]
b0[1:0]
s0[7:0]
Incrementer
upper4bits
7:4
n[3:0]
seq_out[6:0]
hex7seq
lower4bits
3:0
n[3:0]
seq_out[6:0]
hex7seq
two_to_one
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_1
seg[6:0]

Simulation


Top_Level.v
lab2_7seg_testbench.v
lab2_7Seg_testbench_behav.wcfg
757.822 ns
Name
Value
0 ns
500 ns
1,000 ns
1,500 ns
sw[7:0]
07
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
btnL
btnC
btnR
clkin
an[3:0]
[3]
[2]
[1]
[0]
seg[6:0]
[6]
[5]
[4]

Waveform



Appendix B (notes):

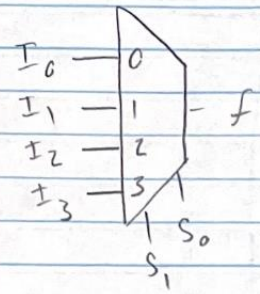
Lab 3 16 Apr 22 12:56 AM 0.6

1000

lab 3

btnL and btnR were switched

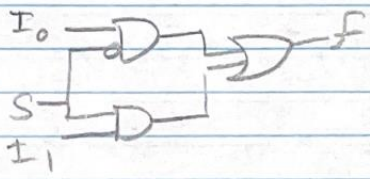
4-to-1 multiplexer T-table



S_1	S_0	f
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

2-to-1 Logic Gates (mux)

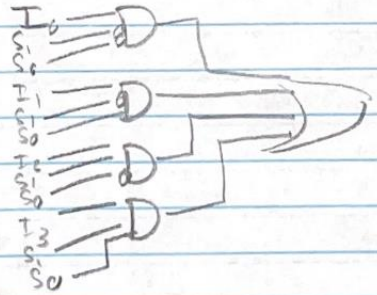
$I_0 \bar{S} + I_1 S$



$I_0 \cdot \bar{S} + I_1 S$

what do I need?

4-to-1 Logic Gates (Mux) Index[]



$S_1 S_0 = 00$

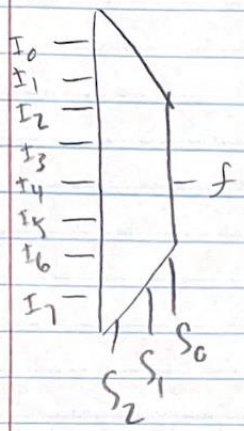
$S_1 S_0 = 01$

$S_1 S_0 = 10$

$S_1 S_0 = 11$

Most Significant bit and index is on the left

8-to-1 Mux



S_2, S_1, S_0

S_2	S_1	S_0	f
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

$$I_0\bar{S}_2\bar{S}_1\bar{S}_0 + I_1\bar{S}_2\bar{S}_1S_0 + I_2\bar{S}_2S_1\bar{S}_0 + I_3\bar{S}_2S_1S_0$$

$$I_4S_2\bar{S}_1\bar{S}_0 + I_5S_2\bar{S}_1S_0 + I_6S_2S_1\bar{S}_0 + I_7S_2S_1S_0$$

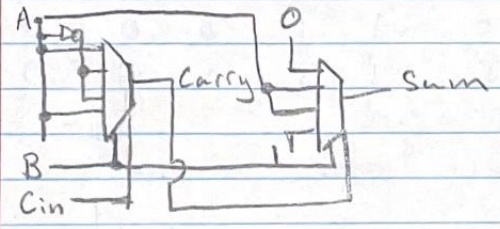
- Every instance has same port name
- Top level uses different wires to connect to ports

4-to-1 boolean Expression

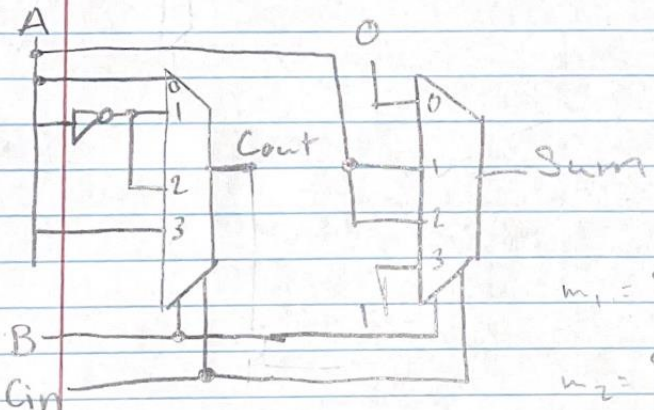
$I_0\bar{S}_1\bar{S}_0 + I_1\bar{S}_1S_0 + I_2S_1\bar{S}_0 + I_3S_1S_0$

8-bit Adder obtain Sum S[7:0]

- Ripple Carry using Full adders
  - 8 Full adders
  - Full Adder
    - 2 4-to-1 multiplexers "M4_1"
    - One inverter
    - 0 and 1 as needed

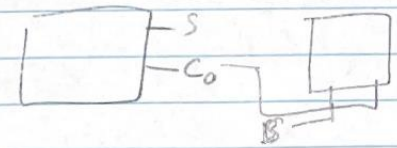


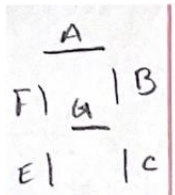
geeksforgeek.org



$m_1 = \{a, \sim a, \sim a, a\}$

$m_2 = \{0, a, a, 1\}$





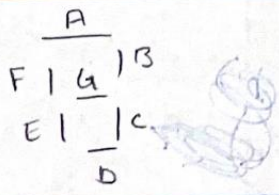
7-Segment Always active low

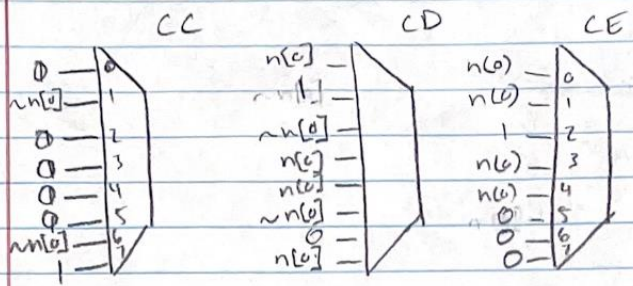
	n_3	n_2	n_1	n_0	CA	CB	CC	CD	CE	CF	CG
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	1	1	0	0
10	1	0	1	0	0	0	0	1	0	0	0
11	1	0	1	1	1	1	0	0	0	0	0
12	1	1	0	0	0	1	1	0	0	0	1
13	1	1	0	1	1	0	0	0	0	1	0
14	1	1	1	0	0	1	1	0	0	0	0
15	1	1	1	1	0	1	1	1	0	0	0

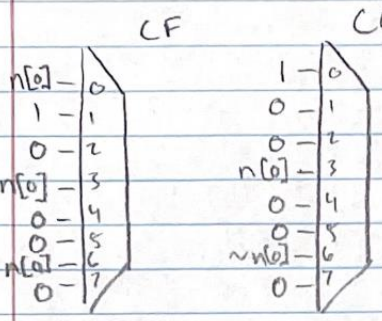
- Why so many multiplexers?
- How are the multiplexers suppose to be connected?
- what are the inputs?

1

Active low
meaning
0 is ON







m0_1 [6:0] (.in(n[0]),

Created my 8 inputs for the 7 muxes, is there an easier way to drive them?
Can I do it in a single line of code rather than 7 different ones?

Full Adder

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1