1.

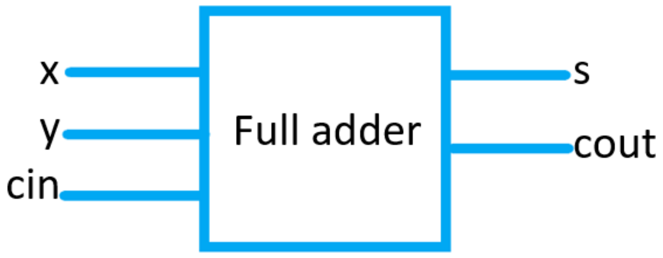
Emulate exp1 in lab1 (a full adder *s*+*cout*=*x*+*y*+*cin*) with the following parameters.

| **I/O** | *x* | *y* | *cin* | *s* | *cout* |
|---|---|---|---|---|---|
| **LOC** | V17 | V16 | W16 | U16 | E19 |

## Design Specification

Input: x, y, cin
Output: s, cout

Block diagram:



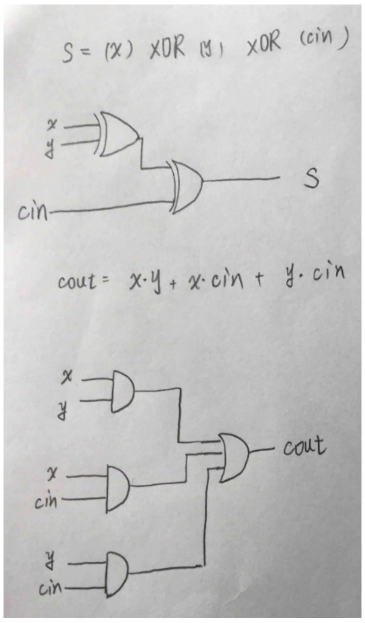
## Design Implementation

Logic equation:

因為 s 表示個位數，所以當 3 個 input (x, y, cin)中奇數個 1，s 則會為 1。所以 s = (x) XOR (y) XOR (cin) = s ^ y ^ cin。

cout 表示進位，當 3 個 input (x, y, cin)中有 2 個或 2 個以上為 1，則 cout 為 1。所以 cout = x·y + x·cin + y·cin。

Logic diagram:



I/O pin assignment

x — V17

y — V16

cin — W16

s — U16

cout — E19

## Discussion:

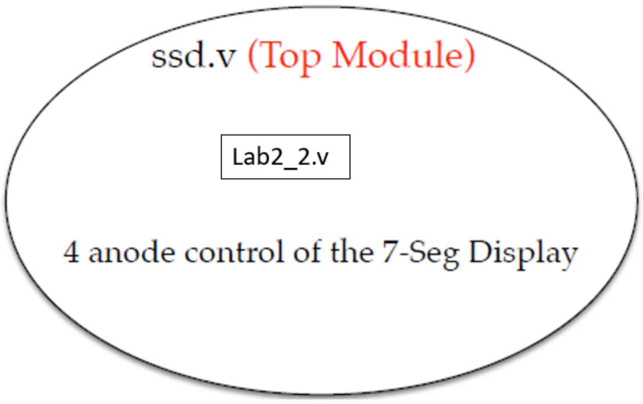
First of all, I started to think when s = 1 and when cout = 1. Hence, I thought I can use XOR to check the number of 1's in the input. If the number of 1's is odd, the XOR result (s) would be 1, otherwise, the result (s) is 0. And I then thought there need two or more inputs equal to 1 to produce carry out. Therefore, I used AND to check. Finally, if s/cout = 1, their corresponded LED on FPGA board will turn on, otherwise, LED is off.

It's single bit full adder with 3 inputs and 2 outputs. So there are $2^3$ = 8 types of input patterns. I used the switch on FPGA to check each pattern and all of them work as I expected.

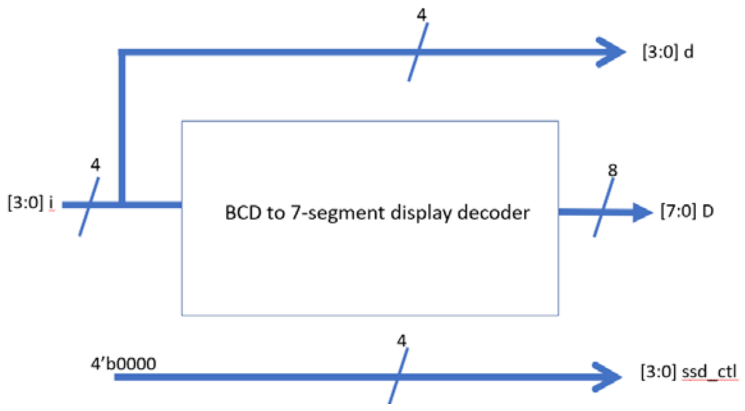
2. Derive a BCD (***i***[3:0]) to 7-segment display decoder (***D_ssd***[7:0]), and also use four LEDs (***d***[3:0]) to monitor the 4-bit BCD number. (Other values of ***i*** outside the range will show F).

## Design Specification

System Hierarchy



Input: [3:0] i
Output: [3:0]d, [7:0]D, [3:0] ssd_ctl
Block diagram:



## Design Implementation

Because there are many possible patterns on 7-segment display and only 16 kinds of input pattern to decide what the display would look like, I decide each bit of output ([7:0] D) to show the corresponded pattern according to each input as shown below.

SS_0→8'b0000_0011

SS_1→8'b1001_1111

SS_2→8'b0010_0101

SS_3→8'b0000_1101

SS_4→8'b1001_1001

SS_5→8'b0100_1001

SS_6→8'b0100_0001

SS_7→8'b0001_1111

SS_8→8'b0000_0001

SS_9→8'b0000_1001

otherwise→8'b0111_0001;

For example, When I find input is $5_{10}$, I set 8 bits of 7-segment display accordingly to show 5 on 7-segment. For the input >= $10_{10}$, I just set 8 bits of 7-segment display to show F.

As for ssd_ctl[3:0], it's always equal to 0 as shown on the block diagram to show patterns on 4 7-segments at the same time.

About d[3:0], it's LED representing input. Therefore, it's just equal to i[3:0].

I/O pin assignment:

ssd_ctl[3] — W4
ssd_ctl[2] — V4
ssd_ctl[1] — U4
ssd_ctl[0] — U2

D[7] — W7
D[6] — W6
D[5] — U8
D[4] — V8
D[3] — U5
D[2] — V5
D[1] — U7

D[0] — V7

d[3] — V19
d[3] — U19
d[3] — E19
d[3] — U16

i[3] — W17
i[2] — W16
i[1] — V16
i[0] — V17

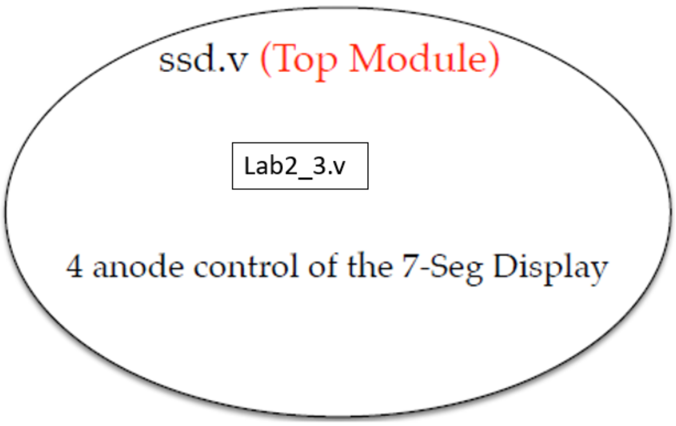
## Discussion:

- I think it’s hard to express each bit of output by input using logic function. So, I assign each bit of output directly according to input.
- So, according to my rule for each bit of bit of output (decide on/off of each segment of 7-segment) explained in implementation, the output shows the correct the pattern on FPGA according to input.

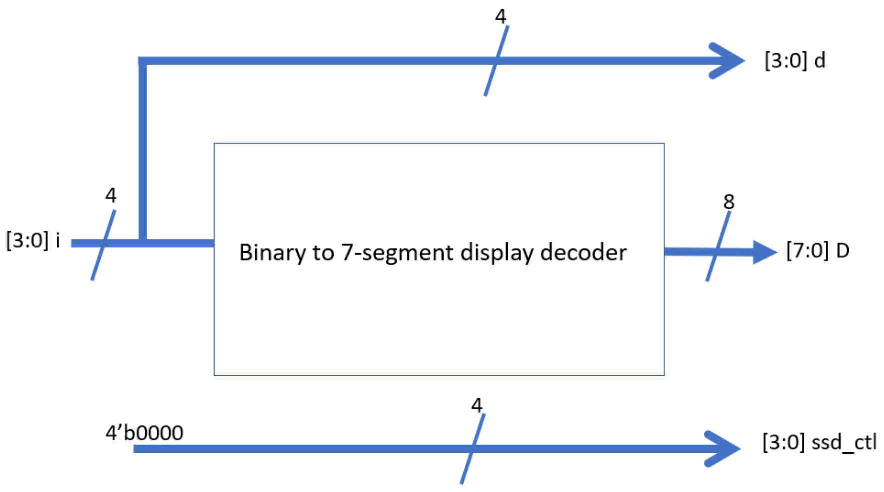
3 Derive a binary (***i***[3:0], 0-9, a, b, c, d, e, f) to 7-segment display decoder (***D***[7:0]), and also use four LEDs (***d***[3:0]) to monitor the 4-bit binary number.

## Design Specification

System Hierarchy



Input: [3:0] i
Output: [7:0] D, [3:0]d, ssd_ctl;
Block diagram:



## Design Implementation

Basically, this problem is similar as lab2_2. The only difference is there are more relation between input and output as input now is binary. As what I did in lab2_2, I assign each bit of output ([7:0] D) according to input ([3:0] i) to show the corresponded the pattern. The relation between input and output is as below.

SS_0→8'b0000_0011

SS_1→8'b1001_1111

SS_2→8'b0010_0101

SS_3→8'b0000_1101

SS_4→8'b1001_1001

SS_5→8'b0100_1001

SS_6→8'b0100_0001

SS_7→8'b0001_1111

SS_8→8'b0000_0001

SS_9→8'b0000_1001

SS_A→8'b0001_0001

SS_b→8'b1100_0001

SS_c→8'b1110_0101

SS_d→8'b1000_0101

SS_E→8'b0110_0001

SS_F→8'b0111_0001

There is no problem to display 0~9 on 7-segment display. The problem is on how to display a, b, c, d, e, f on 7-segment display correctly. To make sure I can understand these characters, I display a in upper case, b in lower case, c in lower case, d in lower case, E in upper case, F in upper case.

As for ssd_ctl[3:0], it's always equal to 0 as shown on the block diagram to show patterns on 4 7-segments at the same time.

About d[3:0], it's LED representing input. Therefore, it's just equal to i[3:0].

I/O pin assignment:

ssd_ctl[3] — W4

ssd_ctl[2] — V4

ssd_ctl[1] — U4

ssd_ctl[0] — U2

D[7] — W7

D[6] — W6

D[5] — U8

D[4] — V8

D[3] — U5

D[2] — V5

D[1] — U7

D[0] — V7

d[3] — V19

d[3] — U19

d[3] — E19

d[3] — U16

i[3] — W17
i[2] — W16
i[1] — V16
i[0] — V17

## Discussion:

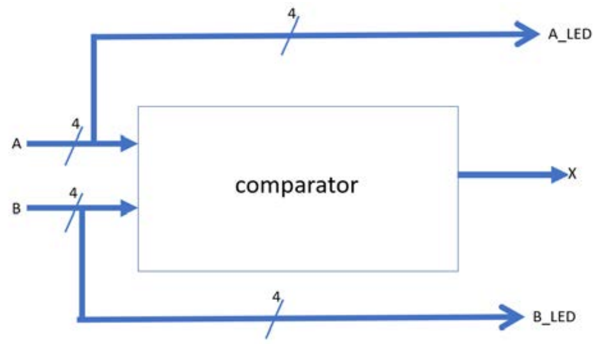
As mentioned in lab2_2, I assigned each bit of output according to input directly as the relation between each bit and input is complicated.

When I want the one of segments of 7-segment to light up, I set the bit representing this segment to 0, otherwise it's zero. So the result is correct as I expected.

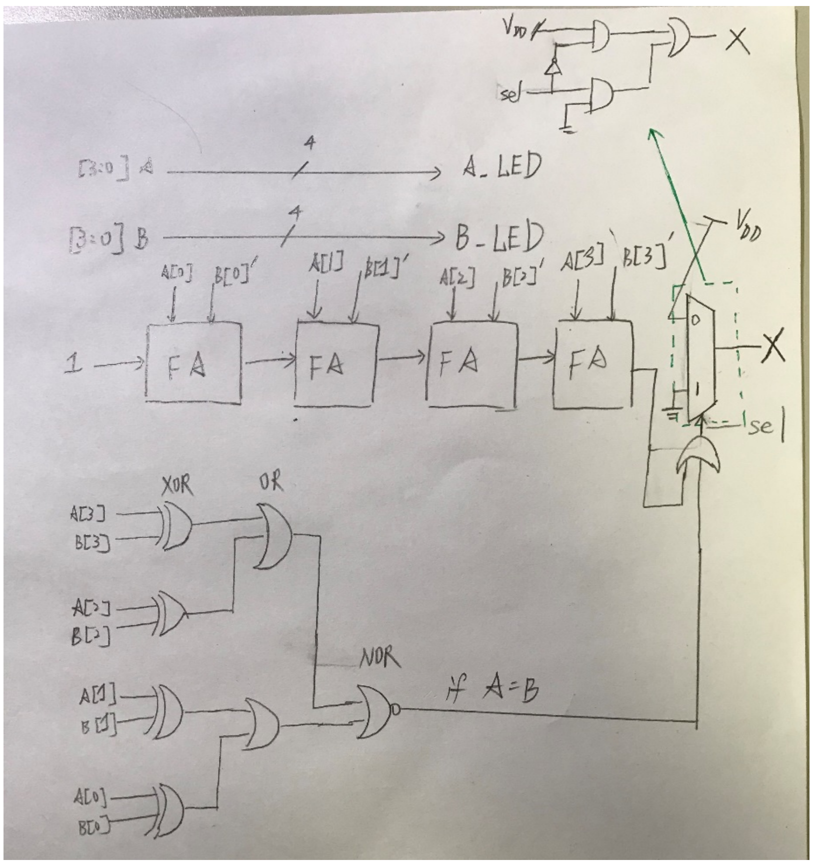
4 (Bonus) Design a combinational circuit that compares two 4-bit unsigned numbers A and B to see whether A is greater than B. The circuit has one output X such that X = 0 if A ≤ B and X = 1 if A > B. (let A[3:0], B[3:0] be controlled by 8 DIP switches, the binary numbers are displayed on 8 LEDs. The result X is on another LED.)

## Design Specification

Input: [3:0]A, B
Output: [3:0]A_LED, B_LED
Output: X
Block diagram:



## Design Implementation



Detail of FA (full adder) has been explained in Lab1!

I/O pin assignment:

A[3]—W17
A[2]—W16
A[1]—V16
A[0]—V17

A_LED[3]—V19
A_LED[2]—U19
A_LED[1]—E19
A_LED[0]—U16

B[3]—W13
B[2]—W14
B[1]—V15
B[0]—W15

B_LED[3]—V14
B_LED[2]—U14
B_LED[1]—U15
B_LED[0]—W18

X—L1

## Discussion:

First of all, I used 4-bit full adder to perform A-B. And I check it's carry out. If it is 0, it means A >= B. If it's 1, it means A < B. However according to the requirement of the problem, X = 0 when A <= B and X = 1 when A > B. So, I need to check if A is equal to B to do multiplexer. I used [(carry out) OR (A == B)] to serve as selection condition for multiplexer. For example, if carry out = 0 and A is equal to B, sel = 0 OR 1 = 1, therefore, X = 0.

As for [3:0]A_LED, B_LED, because it represent A and B respectively, A and B are directly assigned to A_LED and B_LED respectively. (i.e. A_LED = A, B_LED = B)

But in Verilog, I only need to use if…else… statement to express the complex logic architecture, which is easier and straightforward. Thus, the result is correct as I expected.

## Conclusion for Lab2:

In this lab, I have learned how to interact with FPGA board with Verilog code, such as control the LED and 7-segment display. And I know 0 means **on** (light up) for 7-segment but **off** for LED display.

**Reference:** the handout given by professor, from which I have learned how to control 7-segment display and LED.