ECE 272 Lab #2

Adders on an FPGA

Hyunjae Kim

07/06/2021

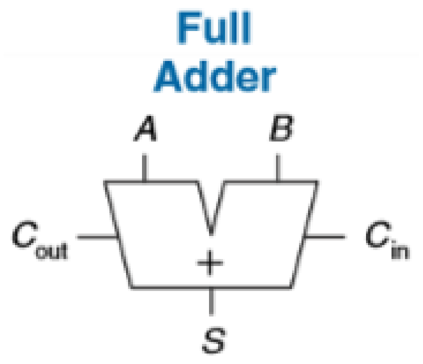
Matthew Shuman

1. Introduction

The general purpose of this lab is to design 1-bit full adder, and then expand it to 4-bit full adder by using Quartus software and FPGA hardware.
Before starting the lab, the student needs to learn about what is full adder and understand combinational logic design. Full adder is a combinational logic design which enable to do adding and subtracting the values which are in binary and accepts Carry in. In multi bit adders, there are Ripple-Carry, Carry-Lookahead, and Carry-save adders. In this section, the student will implement Ripple-Carry multi bit adder because it is easy to understand and implement, although it is slower than other two multi bit adders.
First of all, the student will implement one-bit full adder by using Quartus software. Then, the implemented one-bit full adders will be expanded to 4-bit full adder by connecting Carry out of a one-bit adder to Carry in of other one bit adder successively. Moreover, in the aspect of expanding the design, the student will learn about how to make new schematic symbol in Quartus software. Finally, the student will verify the implemented design by using programmed Max 10 FPGA. If the actual result is not equal to the expected result, then the student has to find what cause the difference and try to fix it in order to make the actual result equal to the expected result.

2. Design

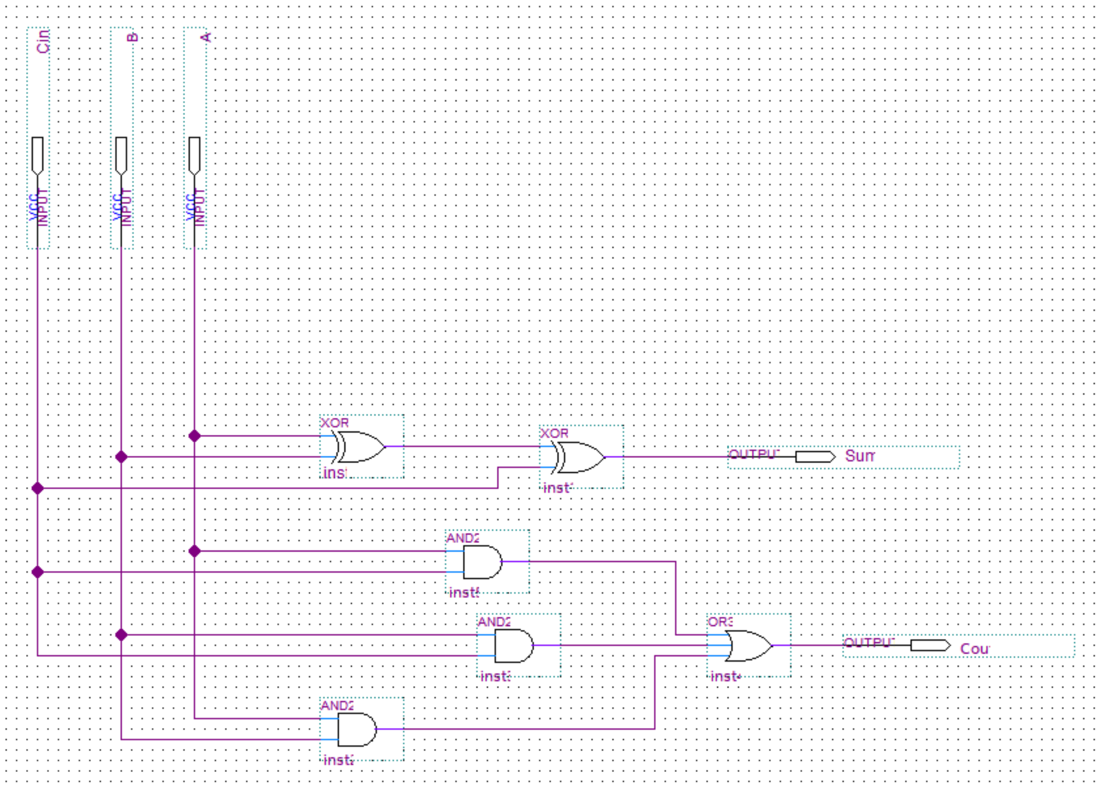
1)Designing one-bit full adder
In designing one-bit full adder, the block design of the one-bit adder is provided in textbook.



The Boolean algebra of SUM (or S) and Cout were provided.
SUM = A ⊕ B ⊕ Cin

Cout = AB + ACin + BCin

Based on the block design and Boolean algebra of one-bit full adder, the one-bit full adder can be implemented as below.



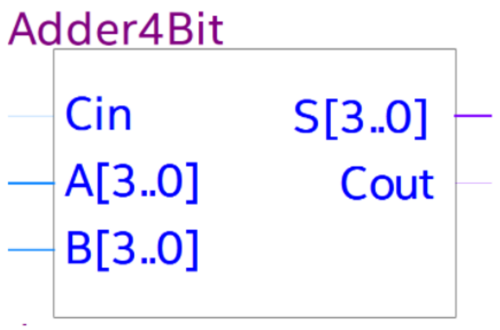
The first part in implementing Sum is using associative law in Boolean algebra.
SUM = A ⊕ B ⊕ Cin = (A ⊕ B) ⊕ Cin

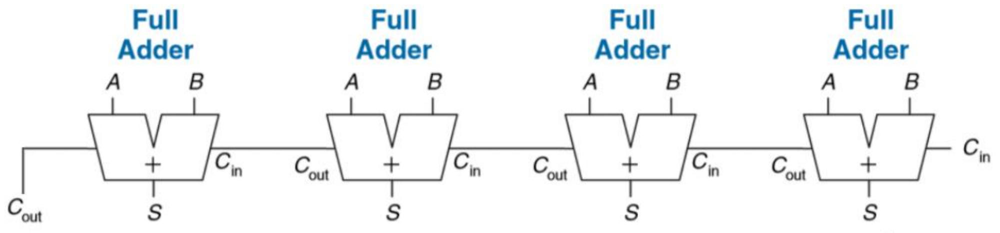
Input A and input B are passing first XOR gate, and the output of XOR gate connected to A and B, and input Cin are connected to second XOR gate. Thus, Sum is the output of the second XOR gate in the schematic design.

The second part is implementing Cout. Implementing Cout can be implemented by using three 2-input AND gates and 3-input OR gate. Because Cout = AB + ACin + BCin. Therefore, Cout is implemented by connecting three AND gate to (A, Cin), (A,B), and (B,Cin) , and then connecting them to 3-input OR gate.

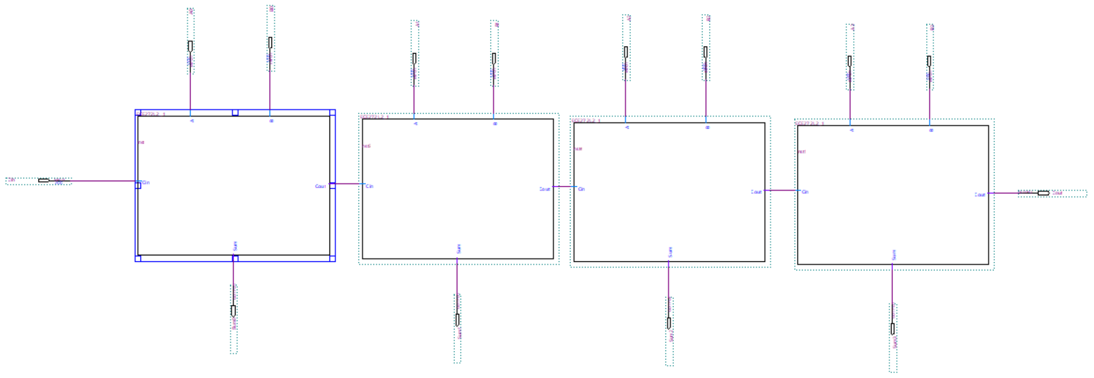
2)Designing 4-bit full adder from one-bit full adder

After implementing one-bit full adder, 4-bit full adder has to be implemented. The block design of 4-bit full adder and complex combinational logic design are provided.

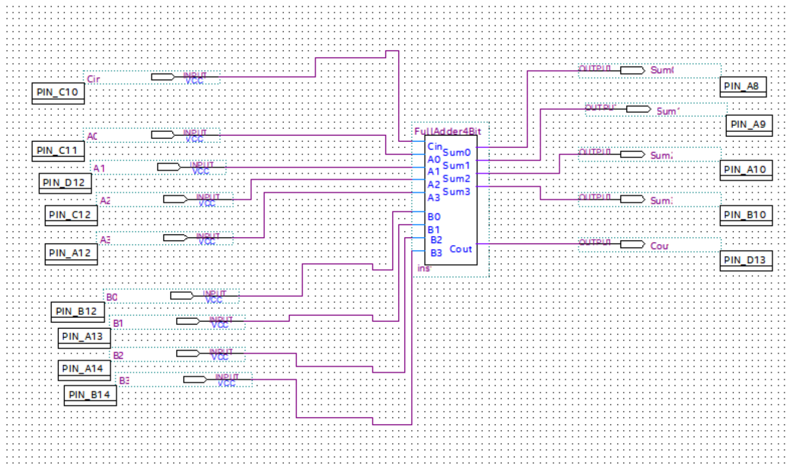




In Quartus software, the design of 4-bit full adder above can be implemented like the design below by using making new symbol function. Cout of a one-bit full adder is connected to Cin of another one-bit full adder successively.



By using making new symbol to the design of 4-bit full adder in Quartus software, the entire 4-bit full adder can be implemented as below.



3) Expecting Results by using truth table

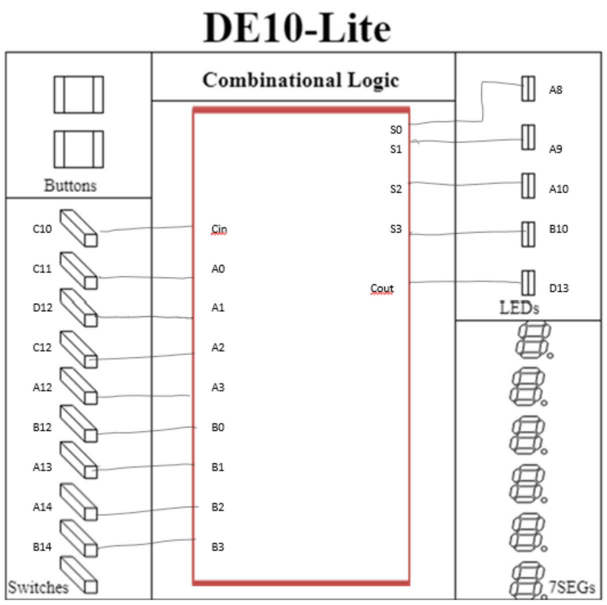
<Expect#1>

Operand1 (A)	Operand2 (B)	Carry In	Value (2-bit binary)	Value (unsigned Decimal)
0b0	0b0	0b0	0	0
0b0	0b0	0b1	1	1
0b0	0b1	0b0	1	1
0b0	0b1	0b0	1	1
0b1	0b0	0b0	1	1
0b1	0b0	0b1	10	2
0b1	0b1	0b0	10	2
0b1	0b1	0b1	11	3

<Expect#2>

Operand1 (A)	Operand2 (B)	5-bit Binary	Unsigned Design	Hexadecimal
0b0000	0b0000	00000	0	0
0b0000	0b0001	00001	1	1
0b0000	0b0010	00010	2	2
0b0000	0b0011	00011	3	3
0b0000	0b0100	00100	4	4
0b0000	0b0101	00101	5	5
0b1000	0b0000	01000	8	8
0b1000	0b0010	01010	10	a
0b1000	0b0011	01011	11	b
0b1111	0b0011	10010	18	12
0b1111	0b1000	10111	23	17
0b1111	0b1010	11001	25	19
0b1111	0b1011	11010	26	1a

4)Programming FPGA



The block design is a 4-bit full adder. Input A0~A3 and input B0~B3 are 4-bit binary input, and input Cin is a one-bit binary input. Output S0~S3 and Cout represents 5-bit binary output. The programmed hardware adds two 4-bit binary inputs by using A0~A3 and B0~B3, and then represents the result in 5-bit binary output by S0~S3 and Cout. For instance, if A3A2A1A0 = 0011, and B3B2B1B0 = 1101, then CoutS3S2S1S0 = 10000. Furthermore, it is possible to do adding binary numbers by using input A0, B0, and Cin. Then the result will be represented in 2-bit binary by S0 and S1.

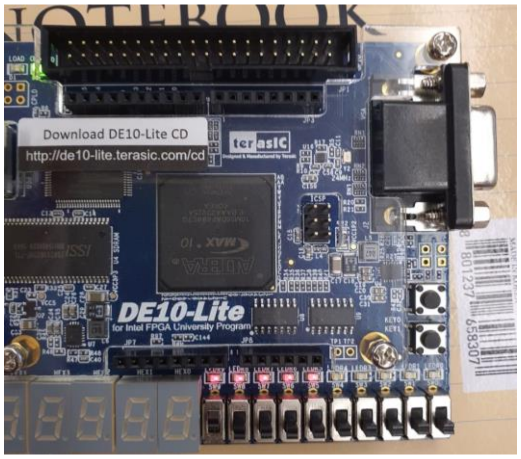
3. Results

<Result of Expect#1>

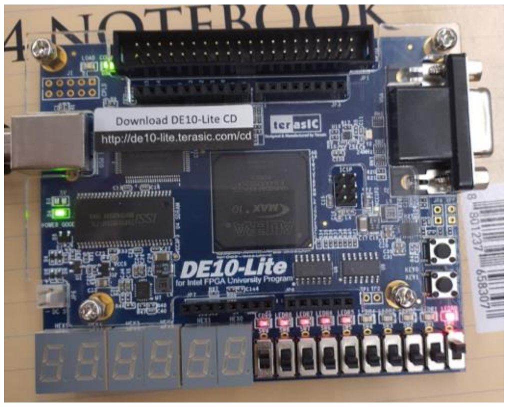
Operand1 (A)	Operand2 (B)	Carry In	Value (2-bit binary)	Value (Actual)
0b0	0b0	0b0	0	0
0b0	0b0	0b1	1	1
0b0	0b1	0b0	1	1
0b1	0b0	0b0	1	1
0b1	0b0	0b1	10	10
0b1	0b1	0b0	10	10
0b1	0b1	0b1	11	11

(Note: The row of the table starts from row 0. The switches are assigned B0B1B2B3A0A1A2A3Cin so that the switch [0] is Cin, and the LEDs are assigned CoutS3S2S1S0 so that LED [0] is S0.)

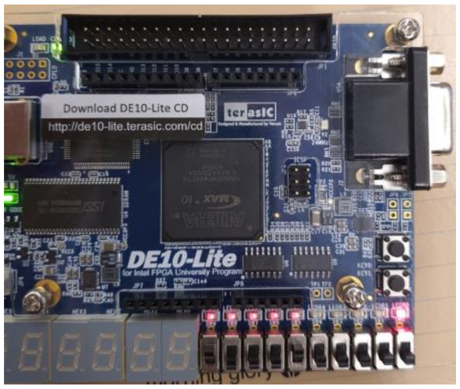
Row.1)



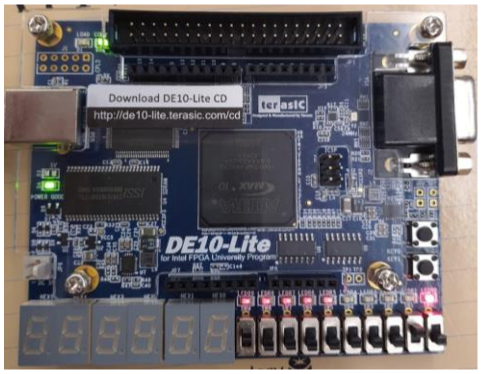
Row 2)



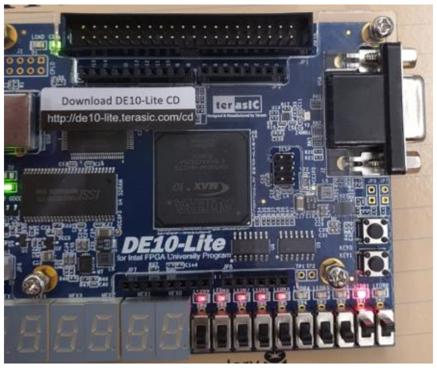
Row 3)


Download DE10-Lite CD
http://de10-lite.terasic.com/cd
terasIC
DE10-Lite
for Intel FPGA University Program

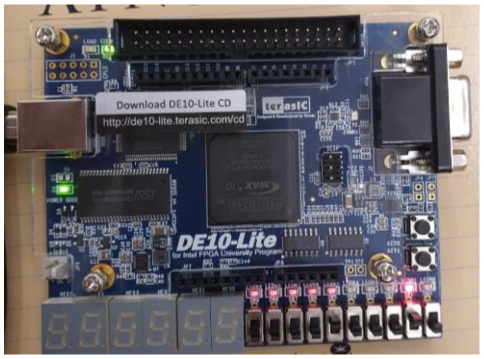
Row 4)


Download DE10-Lite CD
http://de10-lite.terasic.com/cd
terasIC
DE10-Lite
for Intel FPGA University Program

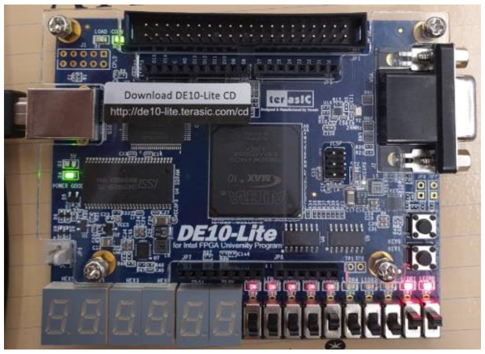
Row 5)


Download DE10-Lite CD
http://de10-lite.terasic.com/cd
terasIC
DE10-Lite
for Intel FPGA University Program

Row 6)


Download DE10-Lite CD
http://de10-lite.terasic.com/cd
terasIC
DE10-Lite
for Intel FPGA University Program

Row 7)

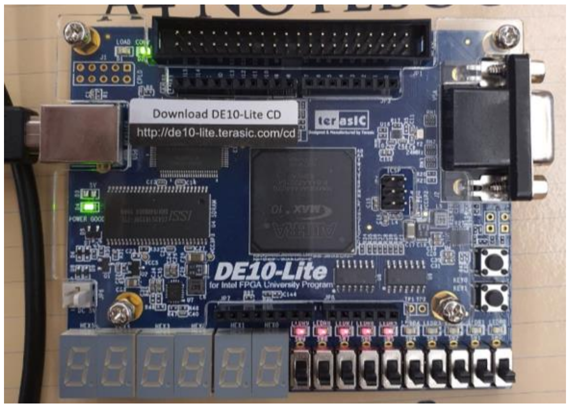

Download DE10-Lite CD
http://de10-lite.terasic.com/cd
terasIC
DE10-Lite
for Intel FPGA University Program

<Result of Expect#2>

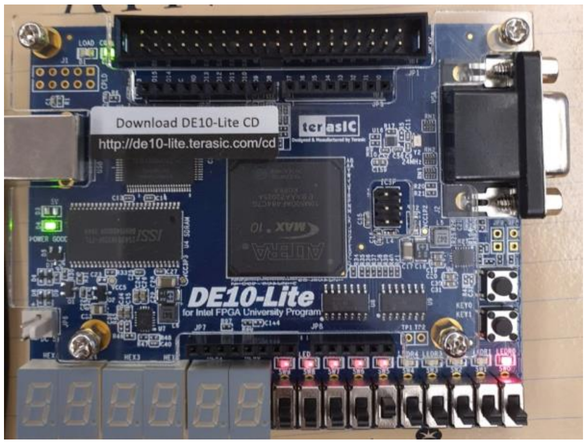
Operand1(A)	Operand2(B)	5-bit Binary	5-bit Binary(actual)
0b0000	0b0000	00000	00000
0b0000	0b0001	00001	00001
0b0000	0b0010	00010	00010
0b0000	0b0011	00011	00011
0b0000	0b0100	00100	00100
0b0000	0b0101	00101	00101
0b1000	0b0000	01000	01000
0b1000	0b0010	01010	01010
0b1000	0b0011	01011	01011
0b1111	0b0011	10010	10010
0b1111	0b1000	10111	10111
0b1111	0b1010	11001	11001
0b1111	0b1011	11010	11010

(Note: The row of the table starts from row 0. The switches are assigned B0B1B2B3A0A1A2A3Cin so that the switch [0] is Cin, and the LEDs are assigned CoutS3S2S1S0 so that LED[0] is S0.)

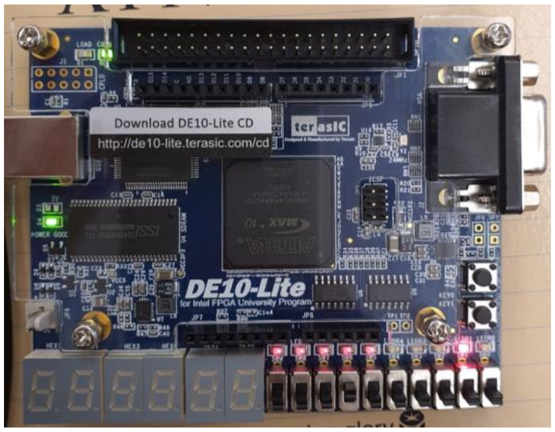
Row 1)



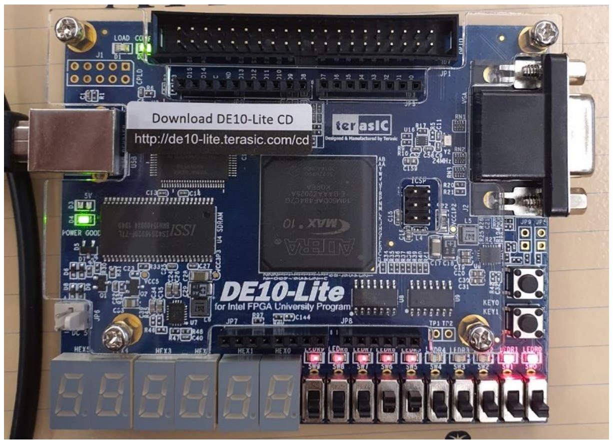
Row 2)



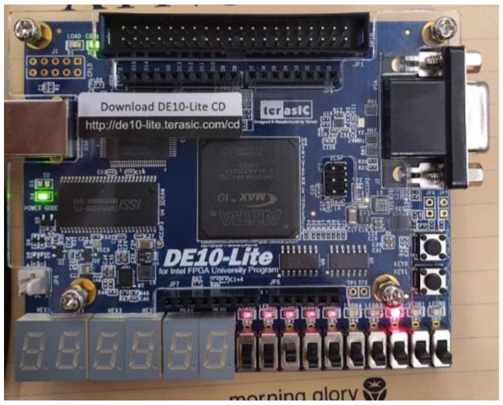
Row 3)



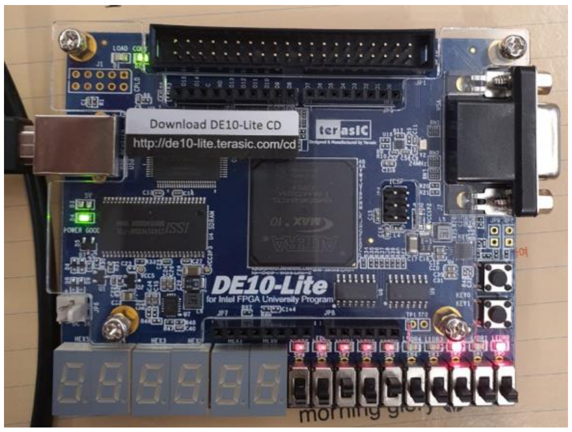
Row 4)



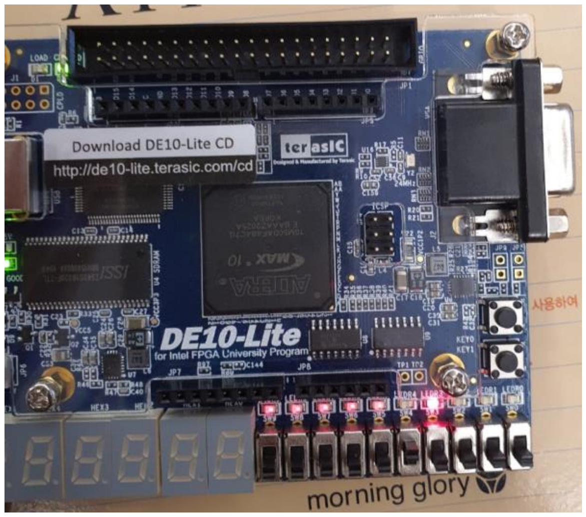
Row 5)



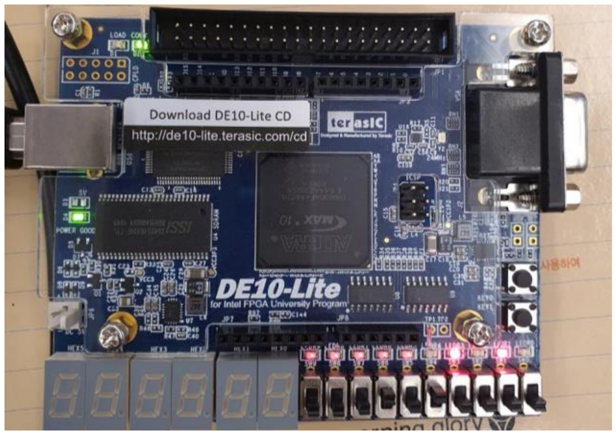
Row 6)



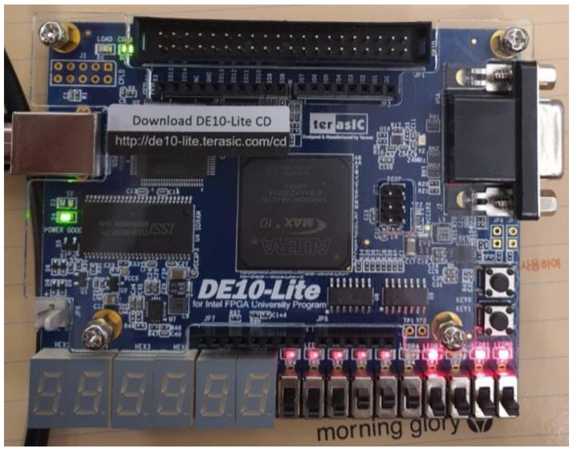
Row 7)



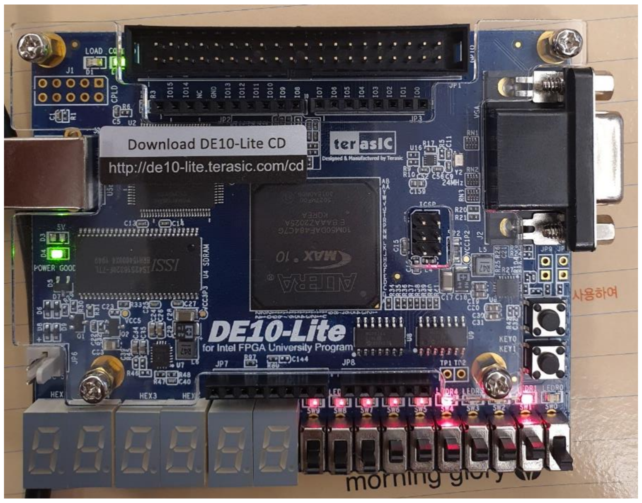
Row 8)



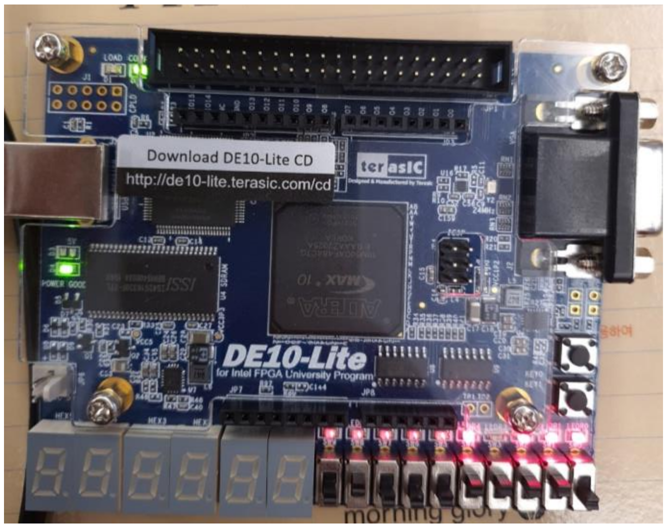
Row 9)



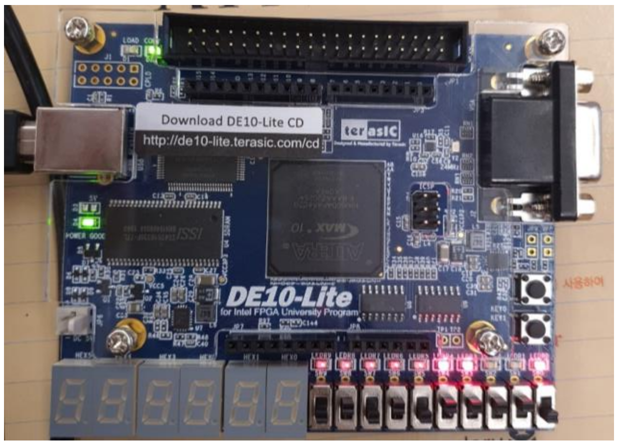
Row 10)



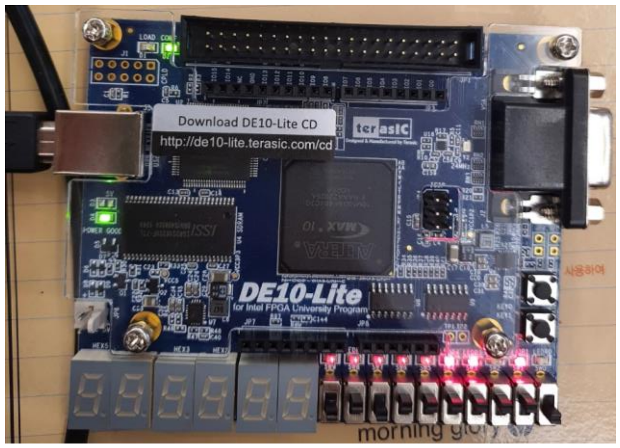
Row 11)



Row 12)



Row 13)



Both predictions before the lab were matched the result of both lab activity. The output of hardware represented sum of 2-bit input and sum of 4-bit input. For instance, the result of 01+01 was 10, and the result of 1111+1011 was 11010. Thus, the experiment was succeeded.

4. Experiment Notes

In this lab activity, there were some hard parts that did not go well. They were making new schematic symbols. The reason was that it was totally new information to me so that I think that it was quite hard to get used to making new schematic symbols. I solved this problem by asking professor in the lab section. However, there were some easy parts that went well. They were designing one-bit full adder and 4-bit full adder because the design and Boolean algebra was provided.

5. Study Questions

1. Explain how you would convert your 4-bit adder to a 4-bit adder/subtractor.

   To convert 4-bit adder to a 4-bit adder/subtractor, I would add NOT gate to input B and make another input connected to Cin which value is 1 [1]. The reason is in Boolean Algebra. First of all, -B = (~B) +1 [2]. Then the subtraction of input A and B, A-B, will be A+(~B) +1 [2]. Therefore, if by controlling another input value connected to Cin and adding NOT gate to input B, the 4-bit adder would be converted to a 4-bit adder/subtractor.

2. Explain your what you did with the least-significant full-adder's carry-in. Why?

In my case, I used the least-significant full-adder's carry-in in adding three operands, input A, input B, and Carry-in. For instance, I tested 1111+1011+1, and the result was 11011 which I expected. The reason is that unlike other inputs, carry-in has only one bit so that it was unable to add three 4-bit binary inputs. Moreover, I wanted to see how carry-in affects the result of the adder so that I predicted that carry-in will just add 1 to the result of the sum of input A and B if the value of carry-in is 1. The result matched my prediction. Thus, I used the carry-in to check the output of the adder if the output changes when the value of carry-in(0/1) changes.

6. Citation

[1] Digital Design and Computer Architecture, Section 5.2.2
[2] Digital Design and Computer Architecture, Section 1.4.6