Laboratory Exercise 2

Numbers and Displays

Môn: Hệ thống số (TN)
Lớp: L01
Nhóm: 7
GVHD: Kiều Đỗ Nguyên Bình

Thành viên:	Lê Đức Huy	1810166
	Nguyễn Gia Huy	1810173
	Huỳnh Thiên Trình	1810615
	Lê Bá Thông	1810555
	Lê Duy Bình	51204735

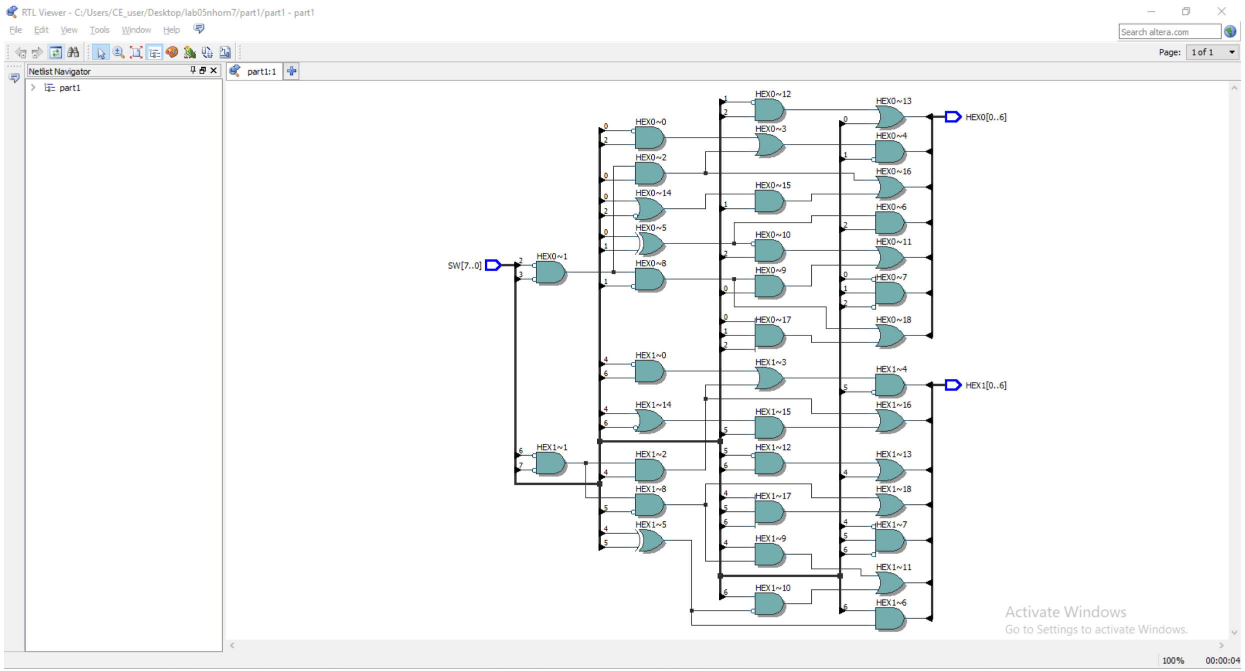
This is an exercise in designing combinational circuits that can perform binary-to-decimal number conversion and binary-coded-decimal (BCD) addition.

Part I

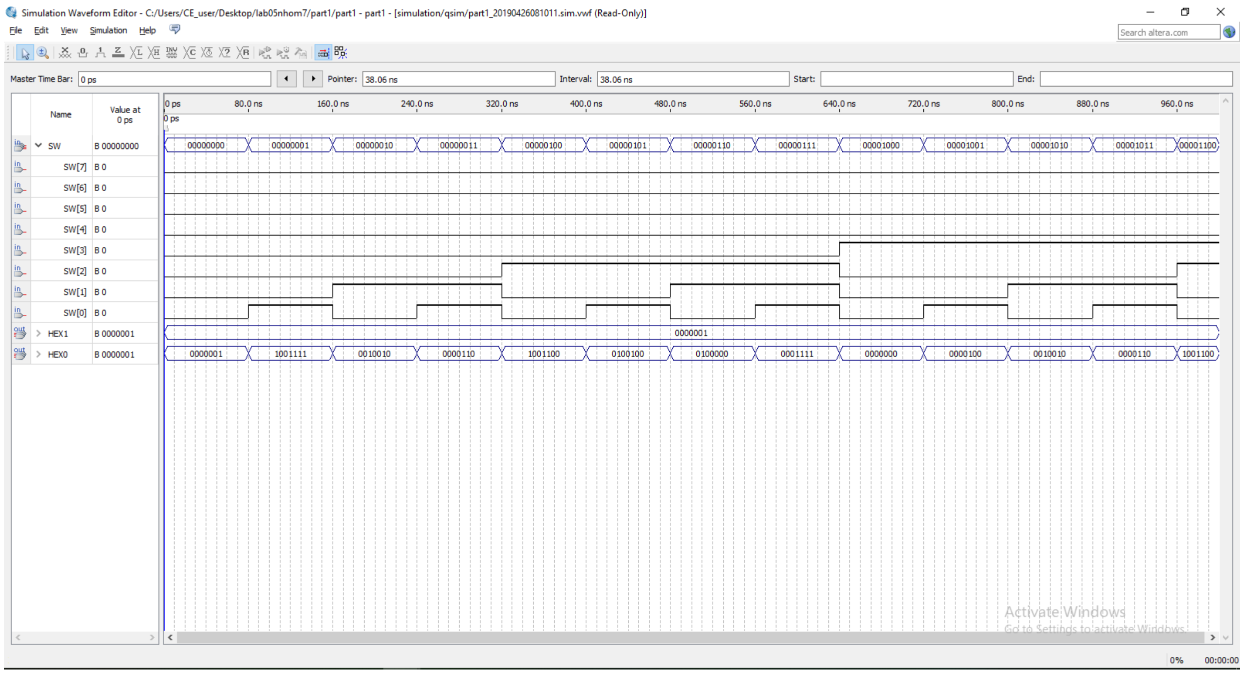
We wish to display on the 7-segment displays *HEX1* and *HEX0* the values set by the switches SW_{7-0}. Let the values denoted by SW_{7-4} and SW_{3-0} be displayed on *HEX1* and *HEX0*, respectively. Your circuit should be able to display the digits from 0 to 9, and should treat the valuations 1010 to 1111 as don't-cares.

1. Create a new project which will be used to implement the desired circuit on your Intel FPGA DE-series board. The intent of this exercise is to manually derive the logic functions needed for the 7-segment dis- plays. Therefore, you should use only simple Verilog **assign** statements in your code and specify each logic function as a Boolean expression.
2. Write a Verilog file that provides the necessary functionality. Include this file in your project and assign the pins on the FPGA to connect to the switches and 7-segment displays. Make sure to include the necessary pin assignments.
3. Compile the project and download the compiled circuit into the FPGA chip.
4. Test the functionality of your design by toggling the switches and observing the displays.

Generated RTL Viewer：


RTL Viewer - C:/Users/CE_user/Desktop/lab05nhom7/part1/part1 - part1

Simulation picture:

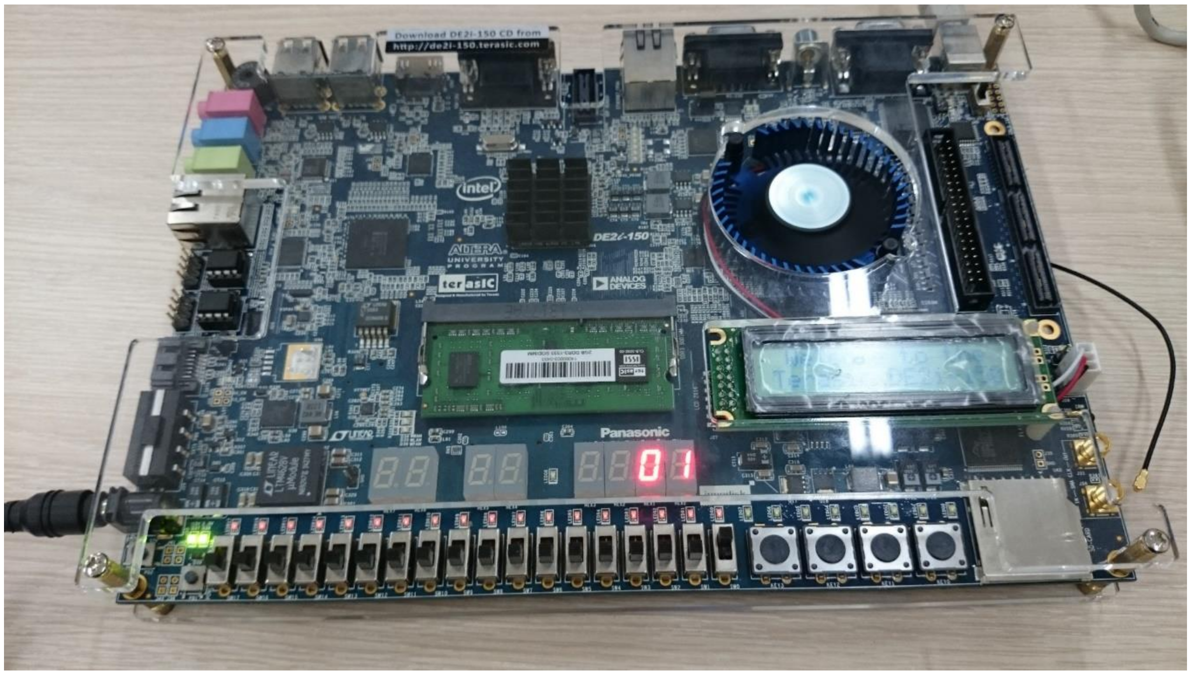

Simulation Waveform Editor - C:/Users/CE_user/Desktop/lab05nhom7/part1/part1 - part1 - [simulation/qsim/part1_20190426081011.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: 38.06 ns Interval: 38.06 ns Start: End:

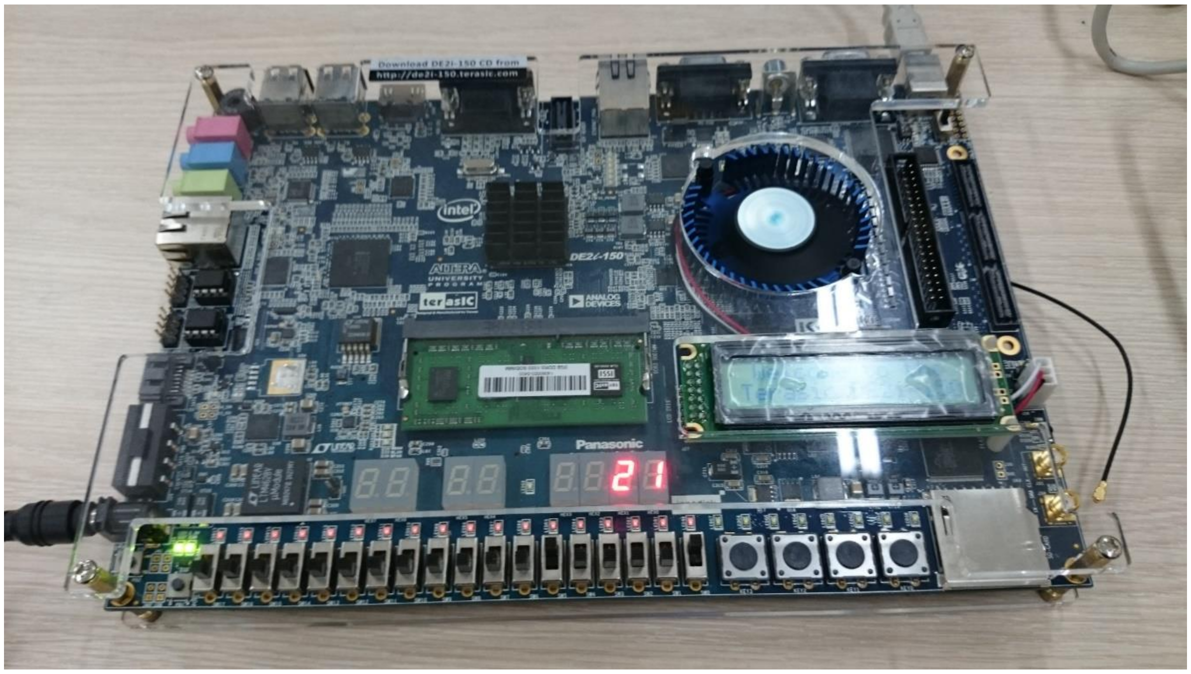
Name	Value at 0 ps
SW	B 00000000
SW[7]	B 0
SW[6]	B 0
SW[5]	B 0
SW[4]	B 0
SW[3]	B 0
SW[2]	B 0
SW[1]	B 0
SW[0]	B 0
HEX1	B 0000001
HEX0	B 0000001

```
SW:   00000000 00000001 00000010 00000011 00000100 00000101 00000110 00000111 00001000 00001001 00001010 00001011 00001100
HEX1: 0000001
HEX0: 0000001  1001111  0010010  0000110  1001100  0100100  0100000  0001111  0000000  0000100  0010010  0000110  1001100
```

Testcase 1: SW7-4: 0000 (0), SW3-0: 0001 (1). Result on DE2i-150 board: 01.



Testcase 2: SW7-4: 0010 (2), SW3-0: 0001 (1). Result on DE2i-150 board: 21.



Testcase 3: SW7-4: 1001 (9), SW3-0: 1001 (9). Result on DE2i-150 board: 99.

Part II

You are to design a circuit that converts a four-bit binary number $V = v_3v_2v_1v_0$ into its two-digit decimal equiv- alent $D = d_1d_0$. Table 1 shows the required output values. A partial design of this circuit is given in Figure 1. It includes a comparator that checks when the value of V is greater than 9, and uses the output of this comparator in the control of the 7-segment displays. You are to complete the design of this circuit.

$v_3v_2v_1v_0$	d_1	d_0
0000	0	0
0001	0	1
0010	0	2
. . .	. . .	. . .
1001	0	9
1010	1	0
1011	1	1
1100	1	2
1101	1	3
1110	1	4
1111	1	5

Table 1: Binary-to-decimal conversion values.

The output z for the comparator circuit can be specified using a single Boolean expression, with the four inputs $V_{3\text{-}0}$. Design this Boolean expression by making a truth table that shows the valuations of the inputs $V_{3\text{-}0}$ for which z has to be 1.

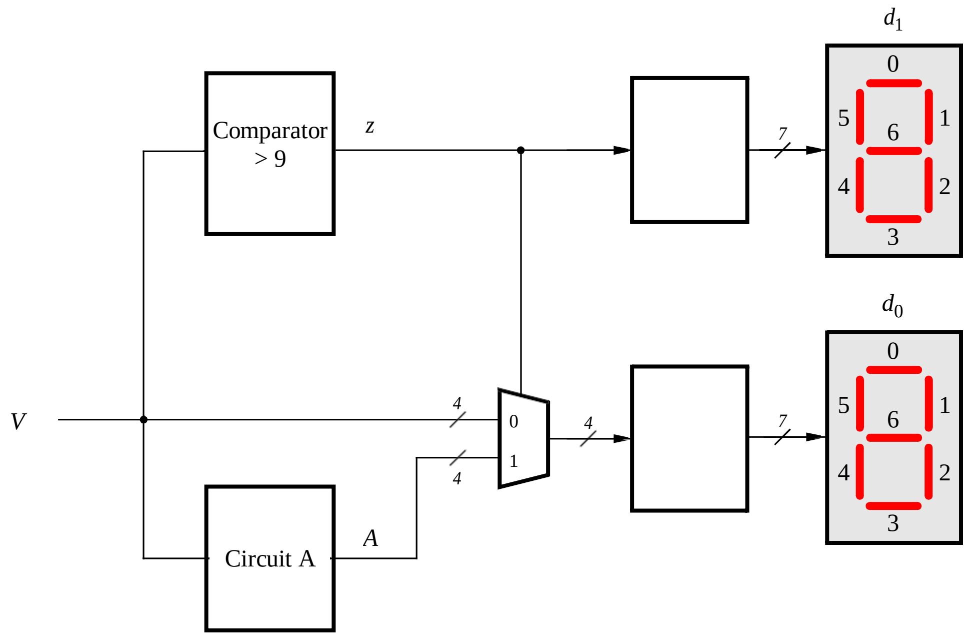


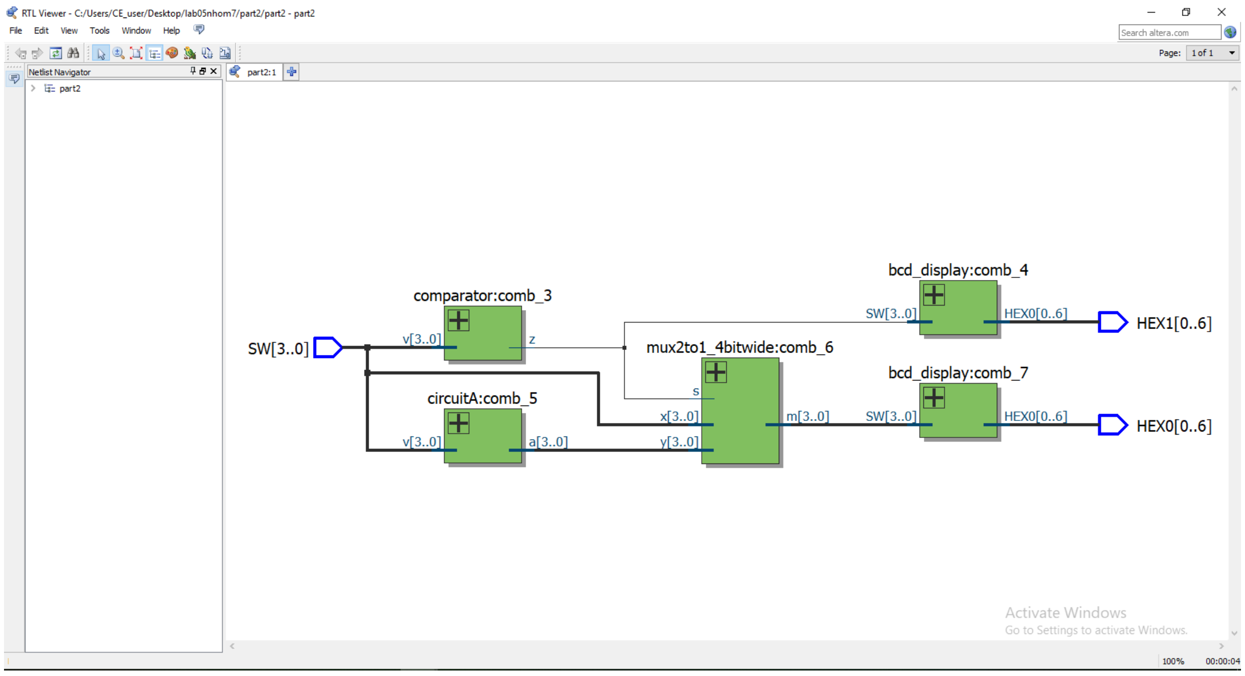
Figure 1: Partial design of the binary-to-decimal conversion circuit.

Notice that the circuit in Figure 1 includes a 4-bit wide 2-to-1 multiplexer (a similar multiplexer was described as part of Laboratory Exercise 1). The purpose of this multiplexer is to drive digit d_0 with the value of V when $z = 0$, and the value of A when $z = 1$. To design circuit A consider the following. For the input values $V \leq 9$, the circuit A does not matter, because the multiplexer in Figure 1 just selects V in these cases. But for the input values $V > 9$, the multiplexer will select A. Thus, A has to provide output values that properly implement Table 1 when $V > 9$. You need to design circuit A so that the input $V = 1010$ gives an output $A = 0000$, the input $V = 1011$ gives the output $A = 0001$, . . ., and the input $V = 1111$ gives the output $A = 0101$. Design circuit A by making a truth table with the inputs V_{3-0} and the outputs A_{3-0}.

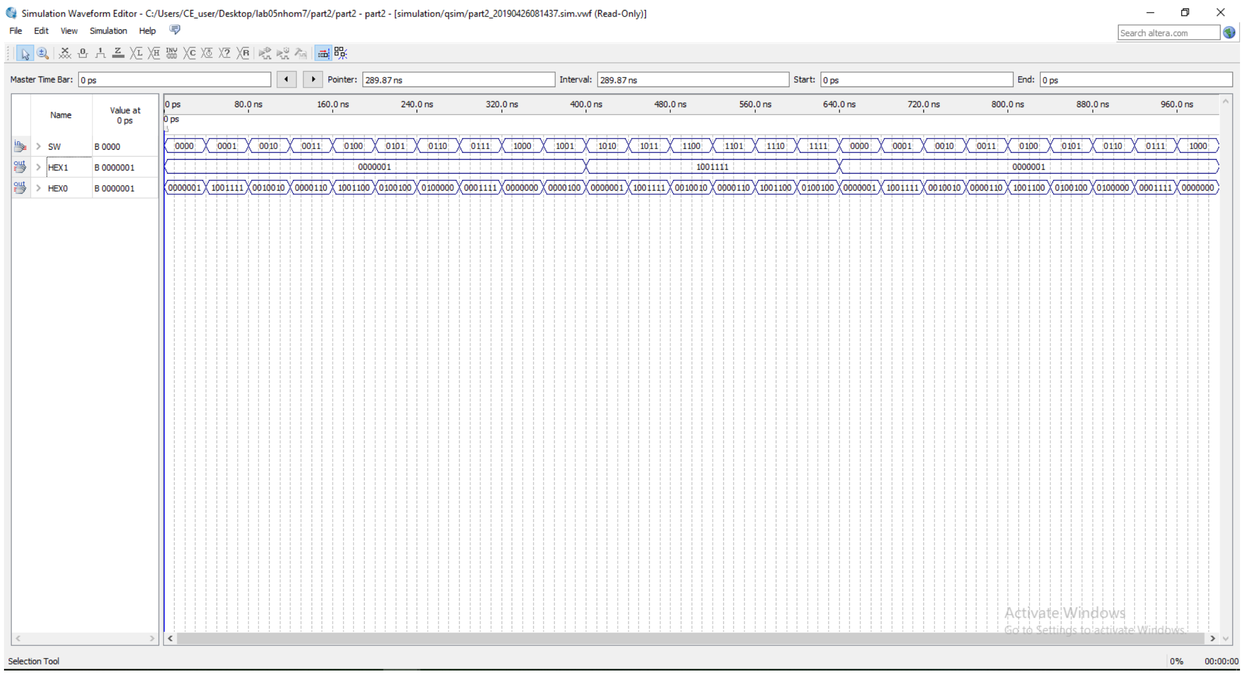
Perform the following steps:

1. Write Verilog code to implement your design. The code should have the 4-bit input SW_{3-0}, which should be used to provide the binary number V, and the two 7-bit outputs *HEX1* and *HEX0*, to show the values of decimal digits d_1 and d_0. The intent of this exercise is to use simple Verilog **assign** statements to specify the required logic functions using Boolean expressions. Your Verilog code should not include any **if-else**, **case**, or similar statements.
2. Make a Quartus project for your Verilog module.
3. Compile the circuit and use functional simulation to verify the correct operation of your comparator, multi- plexers, and circuit A.
4. Download the circuit into an FPGA board. Test the circuit by trying all possible values of V and observing the output displays.

Generated RTL Viewer:

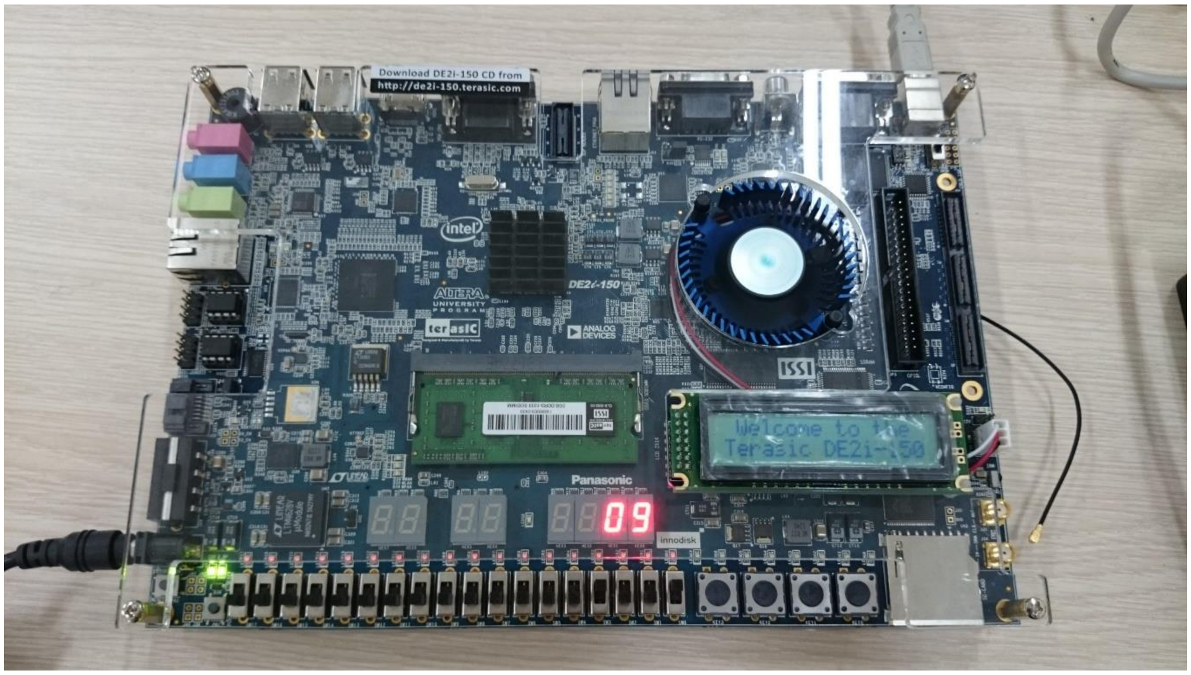


Simulation picture:

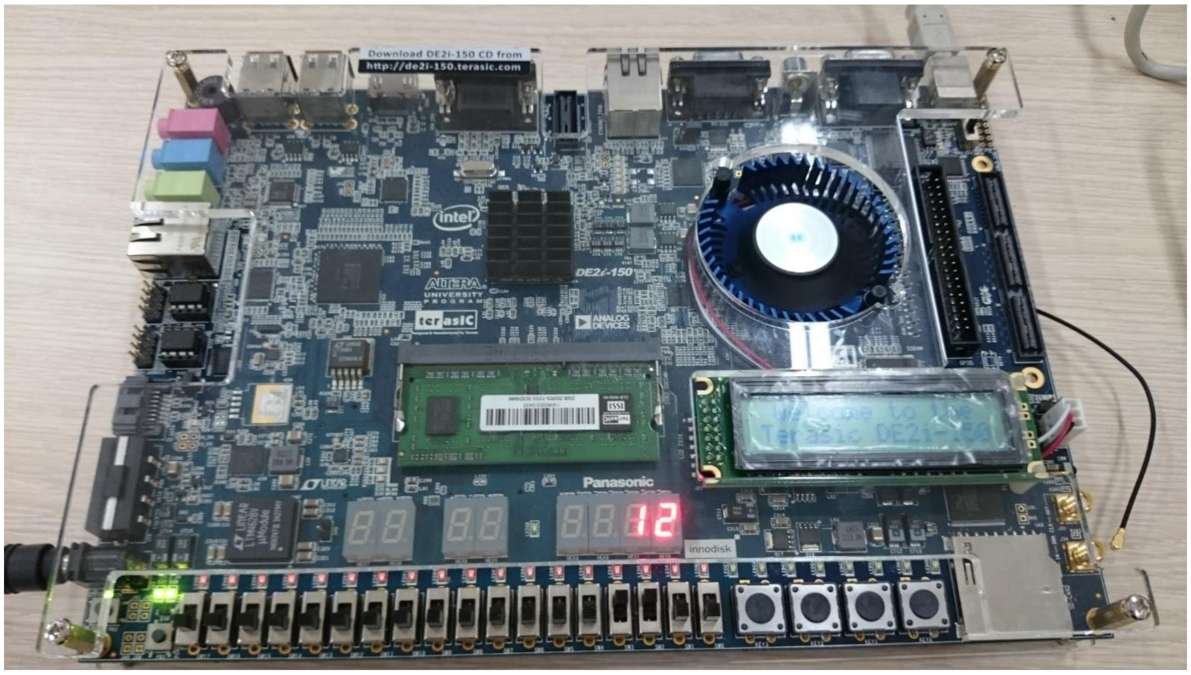

Simulation Waveform Editor - C:/Users/CE_user/Desktop/lab05nhom7/part2/part2 - part2 - [simulation/qsim/part2_20190426081437.sim.vwf (Read-Only)]
File
Edit
View
Simulation
Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 289.87 ns
Interval: 289.87 ns
Start: 0 ps
End: 0 ps
Activate Windows
Go to Settings to activate Windows.
Selection Tool
0%
00:00:00

```
Name   Value at 0 ps   0 ps  80.0 ns  160.0 ns  240.0 ns  320.0 ns  400.0 ns  480.0 ns  560.0 ns  640.0 ns  720.0 ns  800.0 ns  880.0 ns  960.0 ns
SW     B 0000      0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111 0000 0001 0010 0011 0100 0101 0110 0111 1000
HEX1   B 0000001   0000001 | 1001111 | 0000001
HEX0   B 0000001   0000001 1001111 0010010 0000110 1001100 0100100 0100000 0001111 0000000 0000100 0000001 1001111 0010010 0000110 1001100 0100100 0000001 1001111 0010010 0000110 1001100 0100100 0100000 0001111 0000000
```

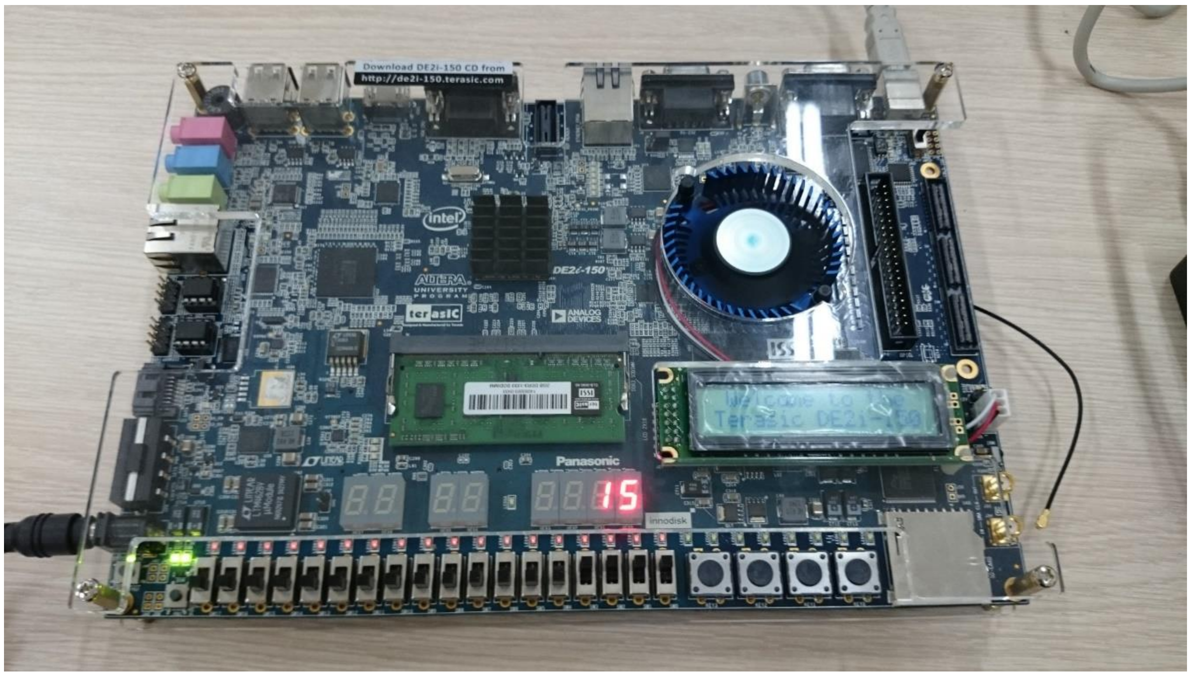
Testcase 1: SW3-0: 1001 (9). Result on DE2i-150 board: 09.



Testcase 2: SW3-0: 1100 (12). Result on DE2i-150 board: 12.

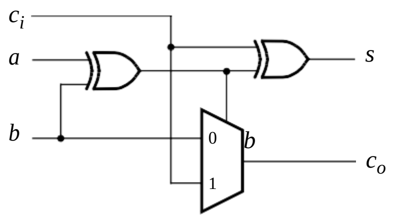


Testcase 3: SW3-0: 1111 (15). Result on DE2i-150 board: 15.

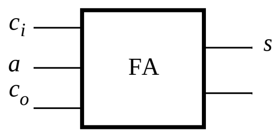


Part III

Figure 2*a* shows a circuit for a *full adder*, which has the inputs a, b, and c_i, and produces the outputs s and c_o. Parts b and c of the figure show a circuit symbol and truth table for the full adder, which produces the two-bit binary sum $c_o s = a + b + c_i$. Figure 2*d* shows how four instances of this full adder module can be used to design a circuit that adds two four-bit numbers. This type of circuit is usually called a *ripple-carry* adder, because of the way that the carry signals are passed from one full adder to the next. Write Verilog code that implements this circuit, as described below.



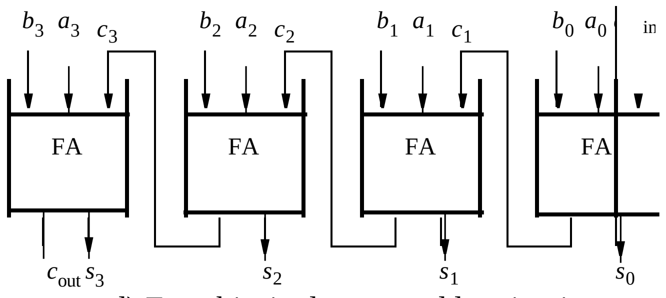
a) Full adder circuit



b) Full adder symbol

b a c_i	c_o s
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1

c) Full adder truth table

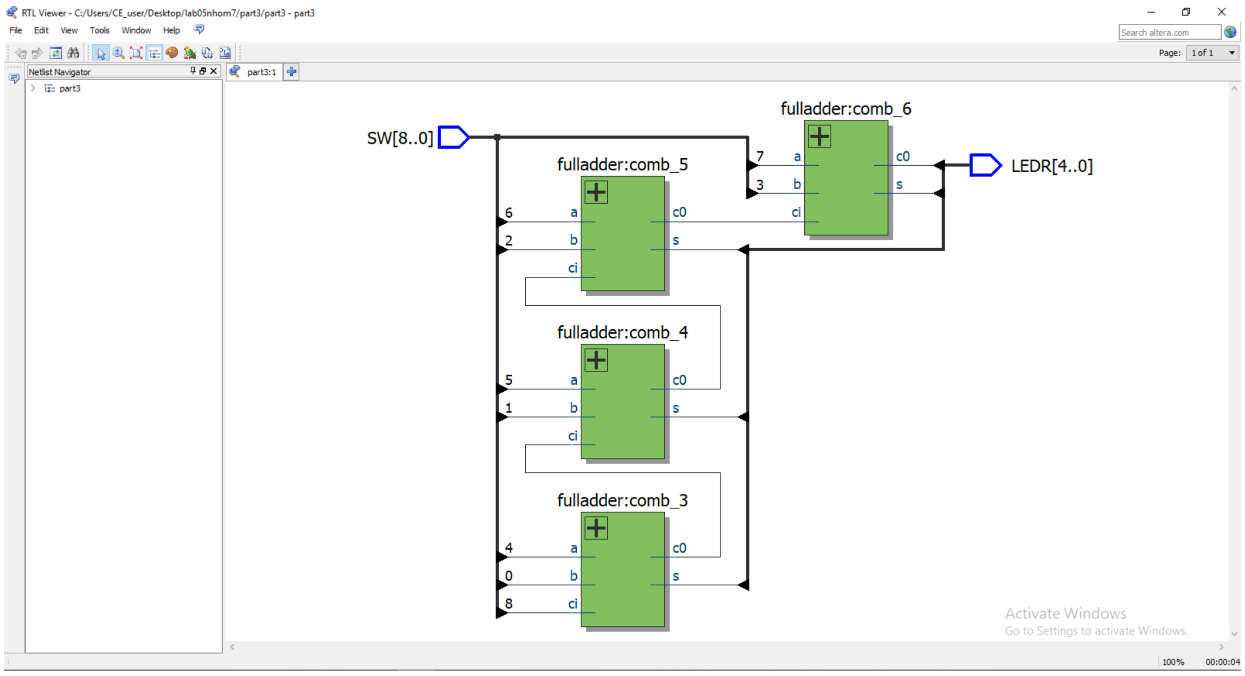


d) Four-bit ripple-carry adder circuit

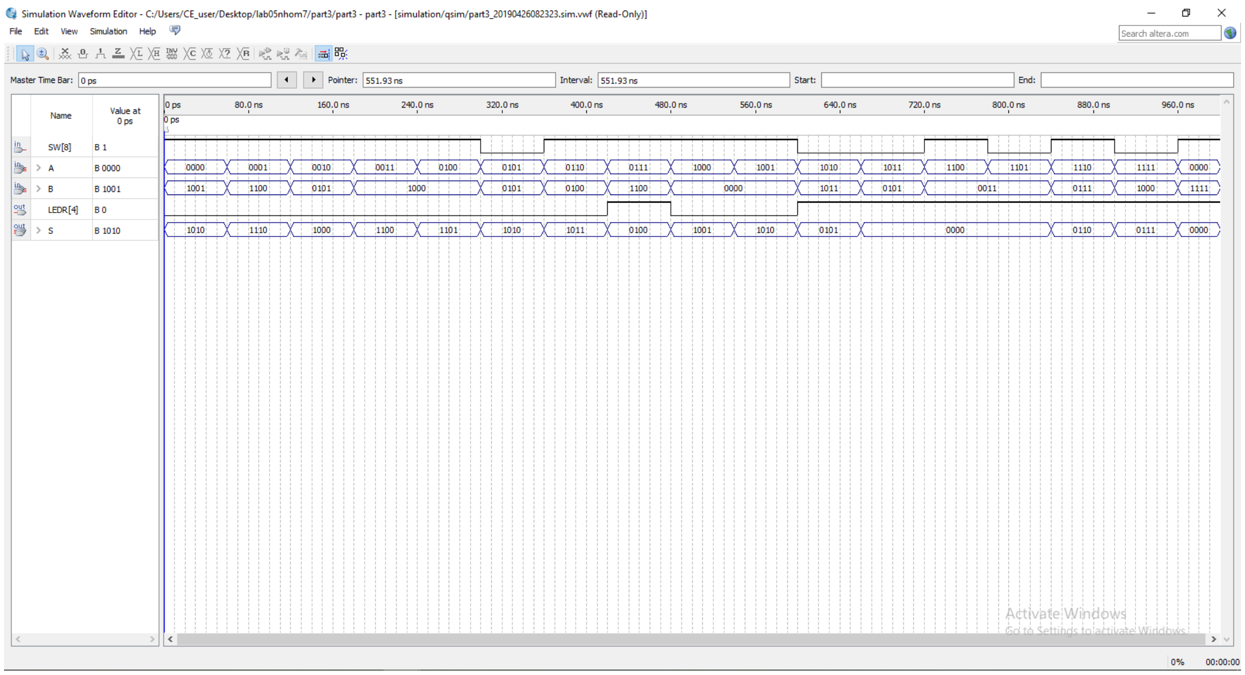
Figure 2: A ripple-carry adder circuit

1. Create a new Quartus project for the adder circuit. Write a Verilog module for the full adder subcircuit and write a top-level Verilog module that instantiates four instances of this full adder.
2. Use switches SW_{7-4} and SW_{3-0} to represent the inputs A and B, respectively. Use SW_8 for the carry-in c_{in} of the adder. Connect the outputs of the adder, c_{out} and S, to the red lights LEDR.
3. Include the necessary pin assignments for your DE-series board, compile the circuit, and download it into the FPGA chip.
4. Test your circuit by trying different values for numbers A, B, and c_{in}.

Generated RTL Viewer:



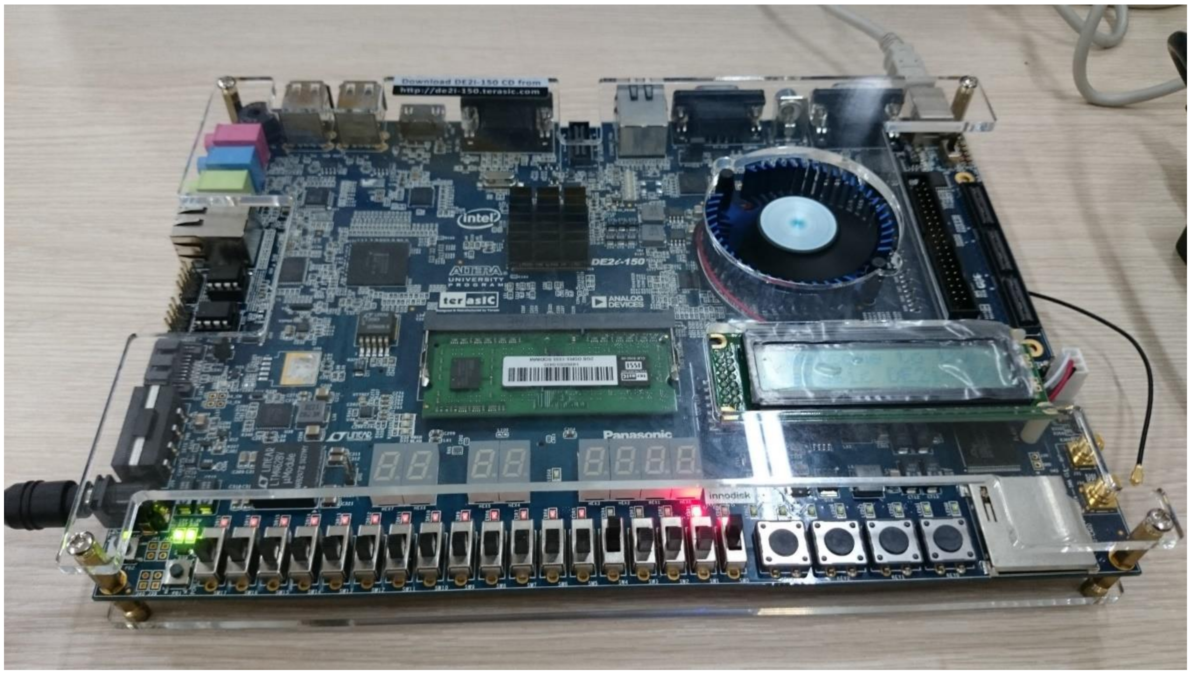
Simulation picture:



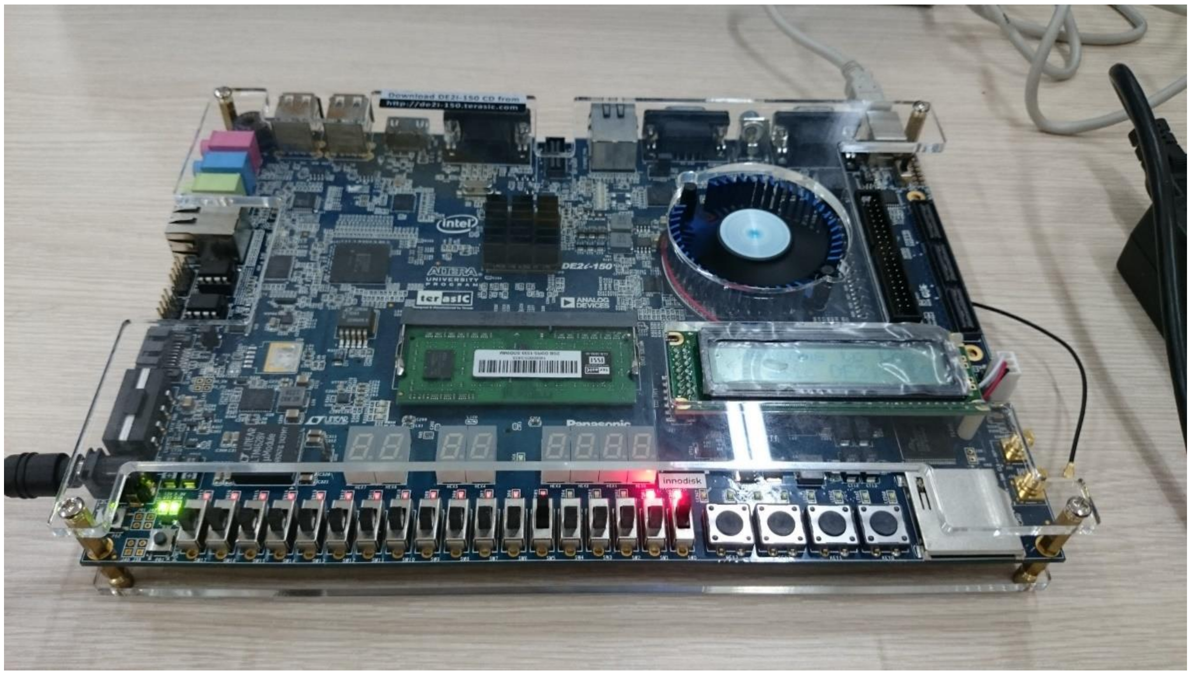
```
A:  0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111 0000
B:  1001 1100 0101 1000 0101 0100 1100 0000 1011 0101 0011 0111 1000 1111
S:  1010 1110 1000 1100 1101 1010 1011 0100 1001 1010 0101 0000 0110 0111 0000
```

Testcase 1: SW8 (carry-in): 0, SW7−4 (A): 0001 (1), SW3-0 (B): 0001 (1).

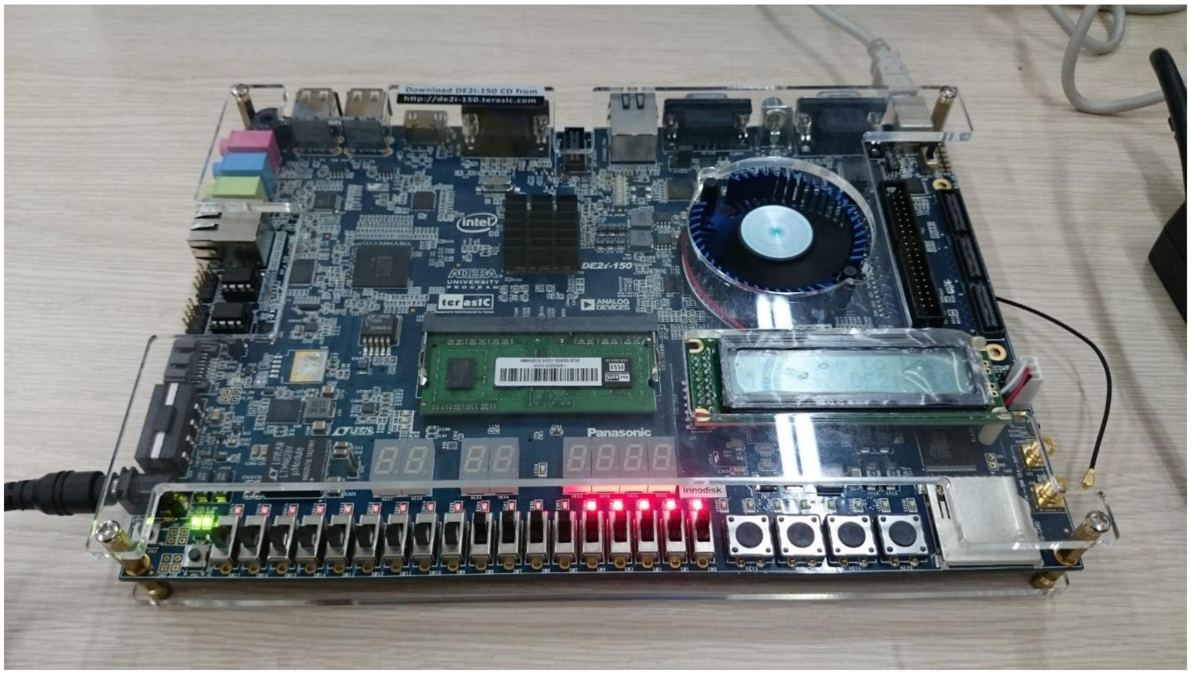
Result on DE2i-150 board: (0+1+1=2) LEDR4-0 (carry-out and S): 00010 (2).



Testcase 2: SW8 (carry-in): 0, SW7–4 (A): 0010 (2), SW3-0 (B): 0001 (1).
Result on DE2i-150 board: (0+2+1=3) LEDR4-0 (carry-out and S): 00011 (3).



Testcase 3: SW8 (carry-in): 1, SW7–4 (A): 1111 (15), SW3-0 (B): 1111 (15).
Result on DE2i-150 board: (1+15+15=31) LEDR4-0 (carry-out and S): 11111 (31).



Part IV

In part II we discussed the conversion of binary numbers into decimal digits. For this part you are to design a circuit that has two decimal digits, X and Y, as inputs. Each decimal digit is represented as a 4-bit number. In technical literature this is referred to as the *binary coded decimal* (BCD) representation.
You are to design a circuit that adds the two BCD digits. The inputs to your circuit are the numbers X and Y, plus a carry-in, c_{in}. When these inputs are added, the result will be a 5-bit binary number. But this result is to be displayed on 7-segment displays as a two-digit BCD sum S_1S_0. For a sum equal to zero you would display S_1S_0 = 00, for a sum of one S_1S_0 = 01, for nine S_1S_0 = 09, for ten S_1S_0 = 10, and so on. Note that the inputs X and Y are assumed to be decimal digits, which means that the largest sum that needs to be handled by this circuit is $S_1S_0 = 9 + 9 + 1 = 19$.
Perform the steps given below.

1. Create a new Quartus project for your BCD adder. You should use the four-bit adder circuit from part III to produce a four-bit sum and carry-out for the operation $X + Y$.

   A good way to work out the design of your circuit is to first make it handle only sums $(X + Y) \leq 15$. With these values, your circuit from Part II can be used to convert the 4-bit sum into the two decimal digits S_1S_0. Then, once this is working, modify your design to handle values of $15 < (X + Y) \leq 19$. One way to do this is to still use your circuit from Part II, but to modify its outputs before attaching them to the 7-segment display to make the necessary adjustments when the sum from the adder exceeds 15.

   Write your Verilog code using simple **assign** statements to specify the required logic functions–do not use other types of Verilog statements such as **if-else** or **case** statements for this part of the exercise.

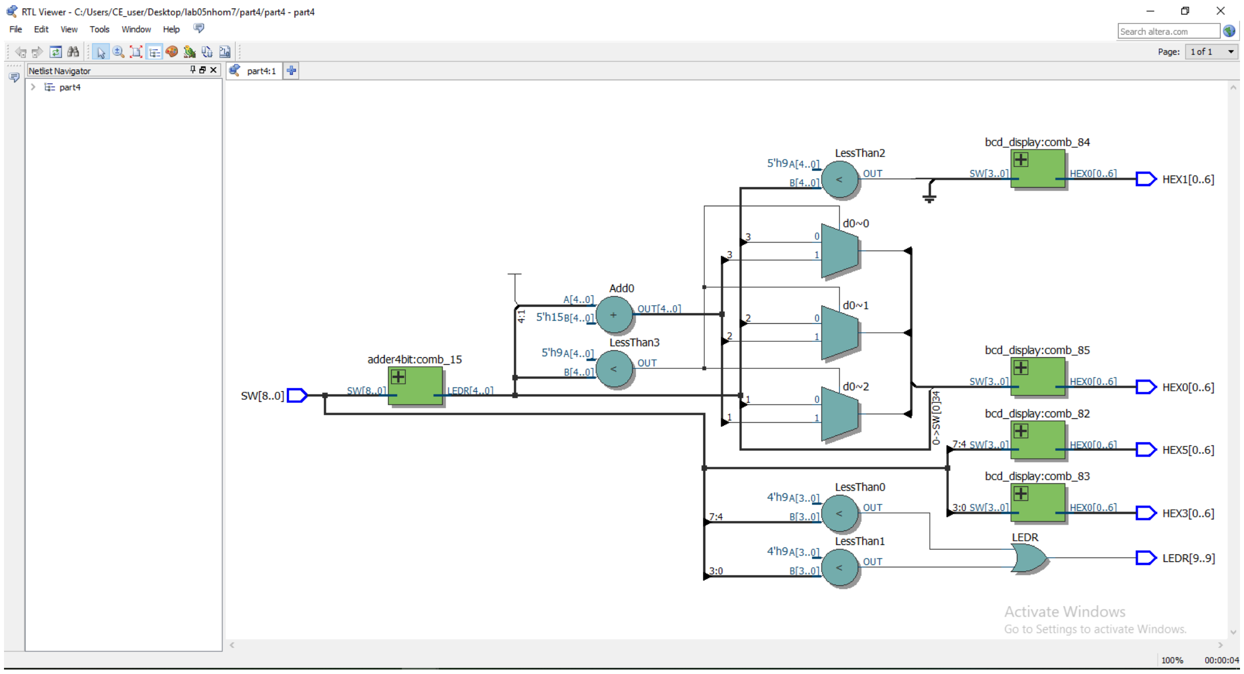
2. Use switches SW_{7-4} and SW_{3-0} for the inputs X and Y, respectively, and use SW_8 for the carry-in. Connect the four-bit sum and carry-out produced by the operation $X + Y$ to the red lights LEDR. Display the BCD values of X and Y on the 7-segment displays *HEX5* and *HEX3*, and display the result S_1S_0 on *HEX1* and *HEX0*.

3. Since your circuit handles only BCD digits, check for the cases when the input X or Y is greater than nine.
   If this occurs, indicate an error by turning on the red light $LEDR_9$.

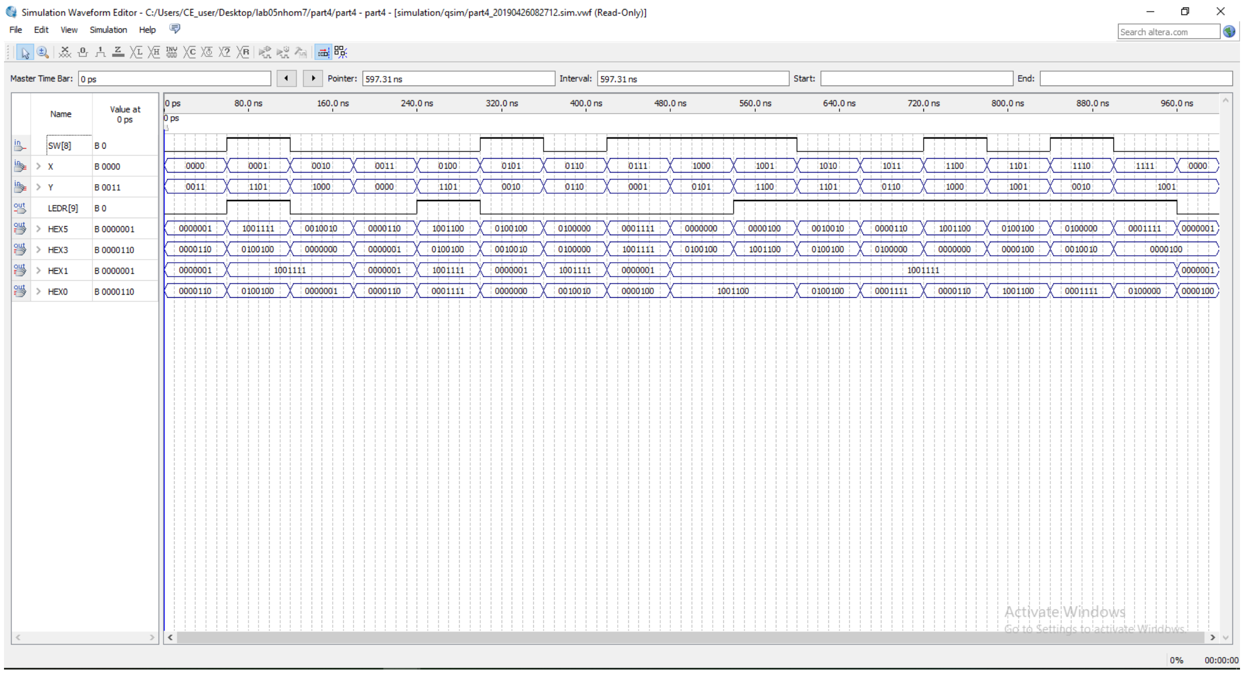
4. Include the necessary pin assignments for your DE-series board, compile the circuit, and download it into the FPGA chip.

5. Test your circuit by trying different values for numbers X, Y, and c_{in}.

Generated RTL Viewer:


RTL Viewer - C:/Users/CE_user/Desktop/lab05nhom7/part4/part4 - part4

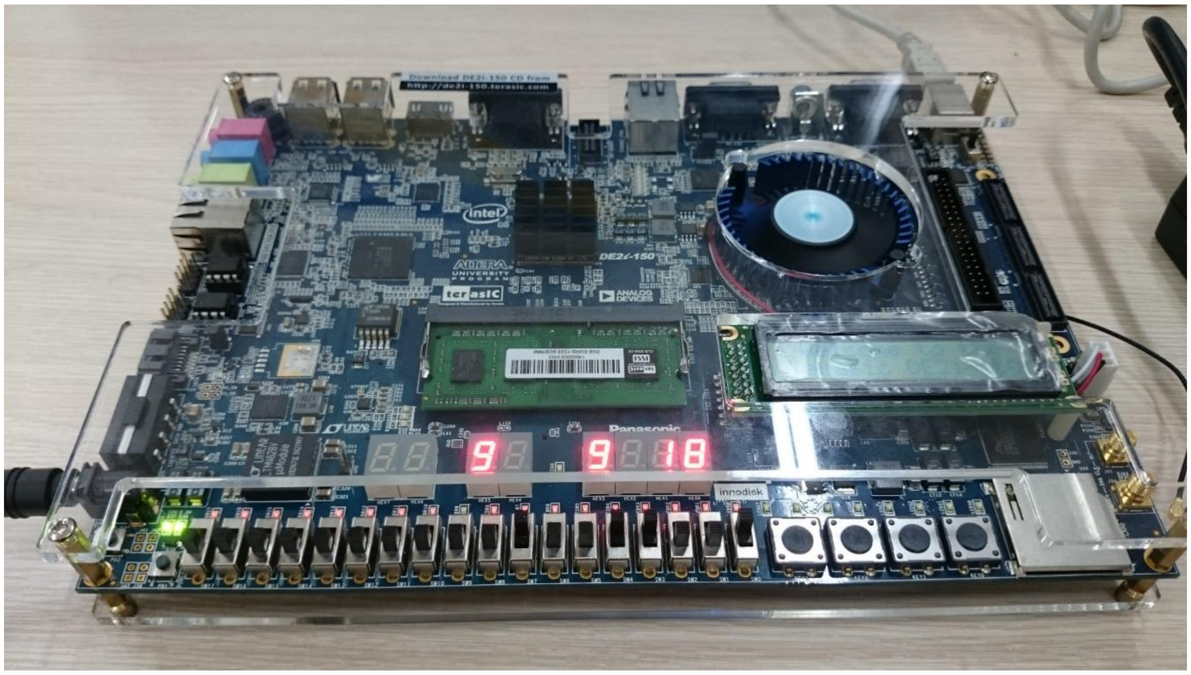
Simulation picture:


Simulation Waveform Editor - C:/Users/CE_user/Desktop/lab05nhom7/part4/part4 - part4 - [simulation/qsim/part4_20190426082712.sim.vwf (Read-Only)]

Master Time Bar: 0 ps
Pointer: 597.31 ns
Interval: 597.31 ns
Start:
End:

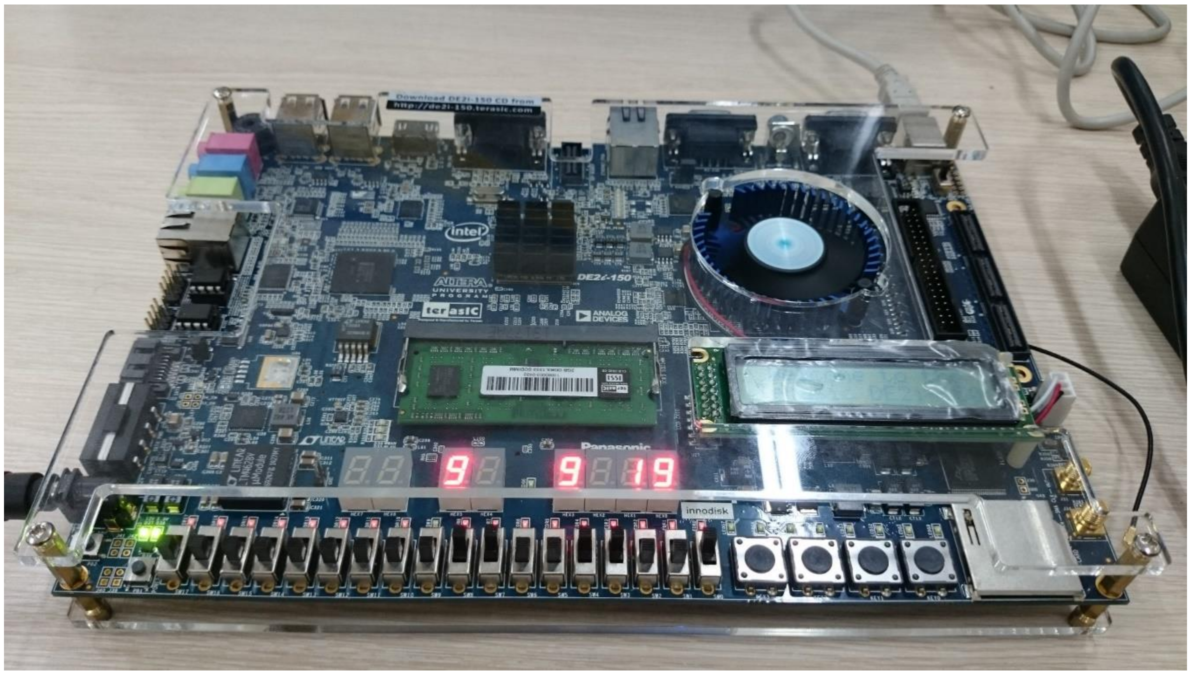
Testcase 1: SW8: 0 (0), SW7-4: 1001 (9), SW3-0: 1001 (9).

Result on DE2i-150 board: (0+9+9) 18.



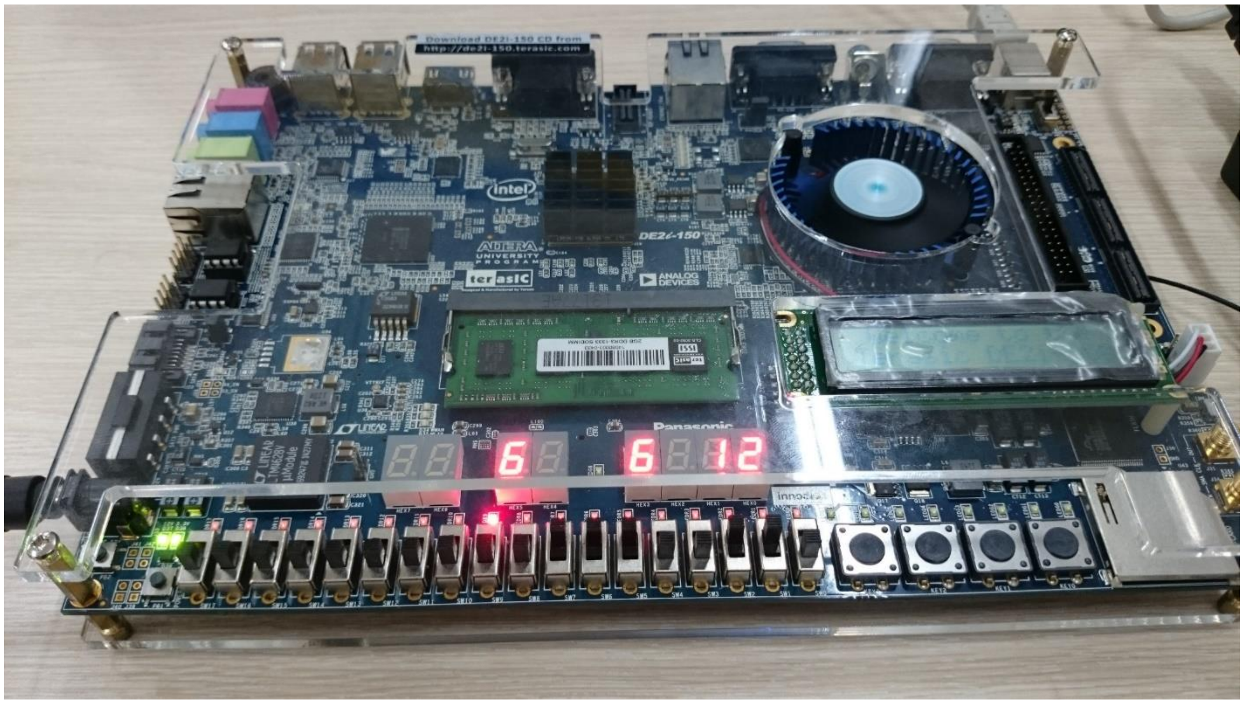
Testcase 2: SW8: 1 (1), SW7-4: 1001 (9), SW3-0: 1001 (9).

Result on DE2i-150 board: (1+9+9) 19.



Testcase 3: SW8 (Cin): 0, SW7-4 (X): 1110 (14 >9), SW3-0 (Y): 0110 (6).

Result on DE2i-150 board: LEDR9 (error indicator): 1.



Part V

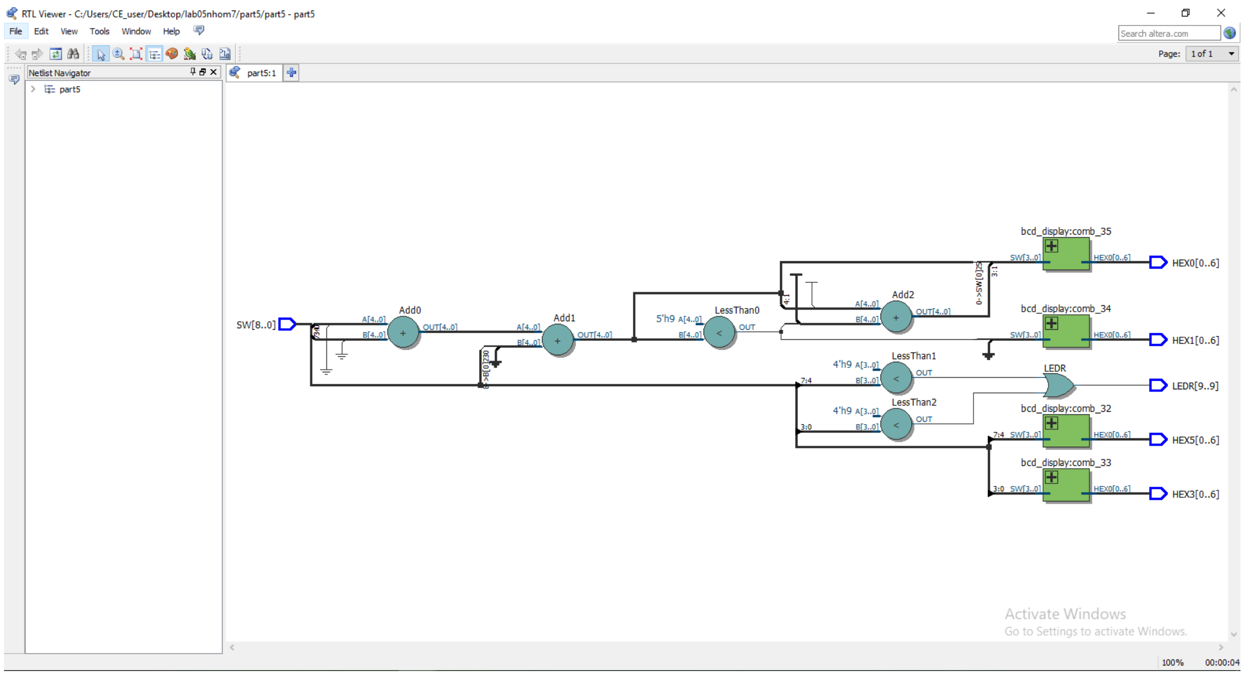
In part IV you created Verilog code for a BCD adder. A different approach for describing the adder in Verilog code is to specify an algorithm like the one represented by the following pseudo-code:

```
1   $T_0 = A + B + c_0$
2   if ($T_0 > 9$) then
3       $Z_0 = 10$;
4       $c_1 = 1$;
5   else
6       $Z_0 = 0$;
7       $c_1 = 0$;
8   end if
9   $S_0 = T_0 - Z_0$
10  $S_1 = c_1$
```

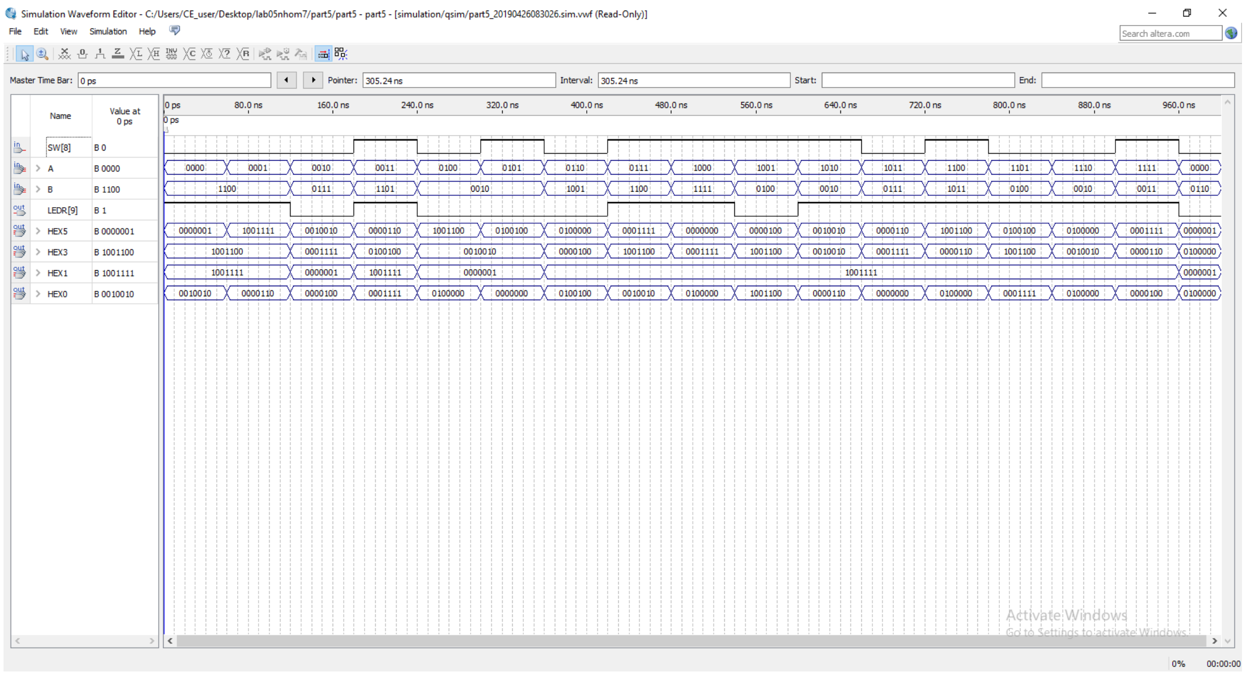
It is reasonably straightforward to see what circuit could be used to implement this pseudo-code. Lines 1 and 9 represent adders, lines 2-8 correspond to multiplexers, and testing for the condition $T_0 > 9$ requires comparators. You are to write Verilog code that corresponds to this pseudo-code. Note that you can perform addition operations in your Verilog code instead of the subtraction shown in line 9. The intent of this part of the exercise is to examine the effects of relying more on the Verilog compiler to design the circuit by using **if-else** statements along with the Verilog > and + operators. Perform the following steps:

1. Create a new Quartus project for your Verilog code. Use switches *SW_{7-4}* and *SW_{3-0}* for the inputs *A* and *B*, respectively, and use *SW_8* for the carry-in. The value of *A* should be displayed on the 7-segment display *HEX5*, while *B* should be on *HEX3*. Display the BCD sum, *S_1S_0*, on *HEX1* and *HEX0*.
2. Use the Quartus RTL Viewer tool to examine the circuit produced by compiling your Verilog code. Compare the circuit to the one you designed in Part IV.
3. Download your circuit onto your DE-series board and test it by trying different values for numbers *A* and *B*.

Generated RTL Viewer:



Simulation picture:



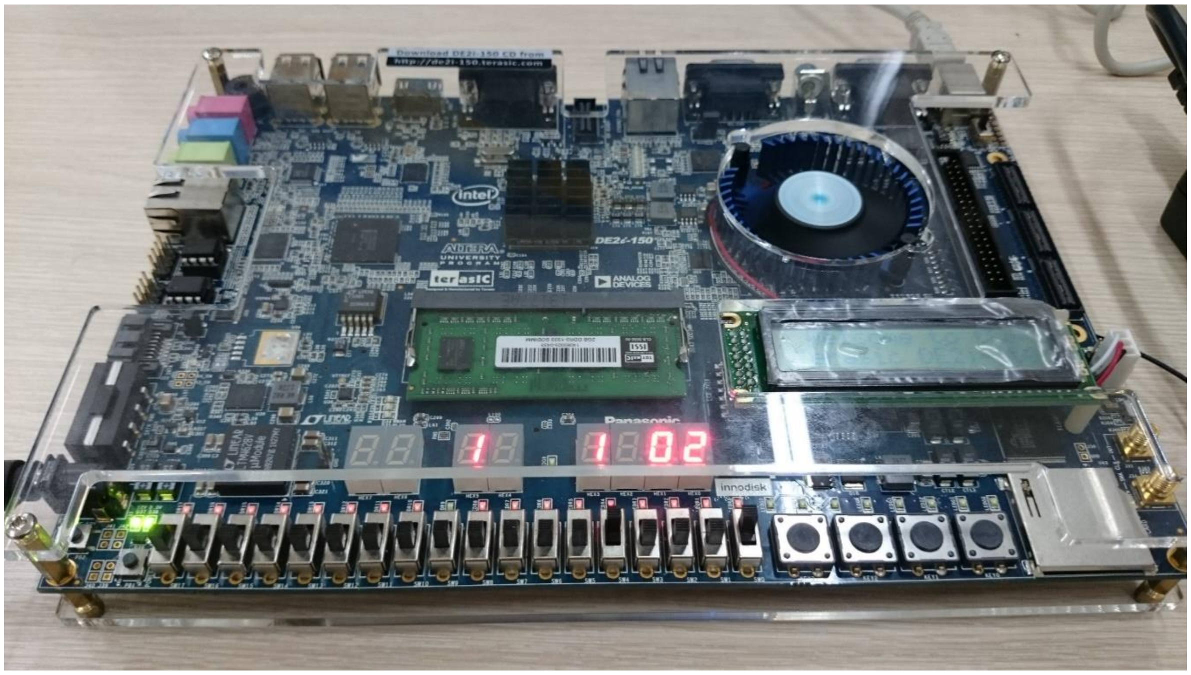
Simulation Waveform Editor - C:/Users/CE_user/Desktop/lab05nhom7/part5/part5 - part5 - [simulation/qsim/part5_20190426083026.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: 305.24 ns Interval: 305.24 ns Start: End:

```
Name     Value at 0 ps   0 ps … 960.0 ns
SW[8]    B 0
A        B 0000      0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 | 0000
B        B 1100      1100 | 0111 | 1101 | 0010 | 1001 | 1100 | 1111 | 0100 | 0010 | 0111 | 1011 | 0100 | 0010 | 0011 | 0110
LEDR[9]  B 1
HEX5     B 0000001   0000001 | 1001111 | 0010010 | 0000110 | 1001100 | 0100100 | 0100000 | 0001111 | 0000000 | 0000100 | 0010010 | 0000110 | 1001100 | 0100100 | 0100000 | 0001111 | 0000001
HEX3     B 1001100   1001100 | 0001111 | 0100100 | 0010010 | 0000100 | 1001100 | 0001111 | 1001100 | 0010010 | 0001111 | 0000110 | 1001100 | 0010010 | 0000110 | 0100000
HEX1     B 1001111   1001111 | 0000001 | 1001111 | 0000001 | 1001111 | 0000001
HEX0     B 0010010   0010010 | 0000110 | 0000100 | 0001111 | 0100000 | 0000000 | 0100100 | 0010010 | 0100000 | 1001100 | 0000110 | 0000000 | 0100000 | 0001111 | 0100000 | 0000100 | 0100000
```

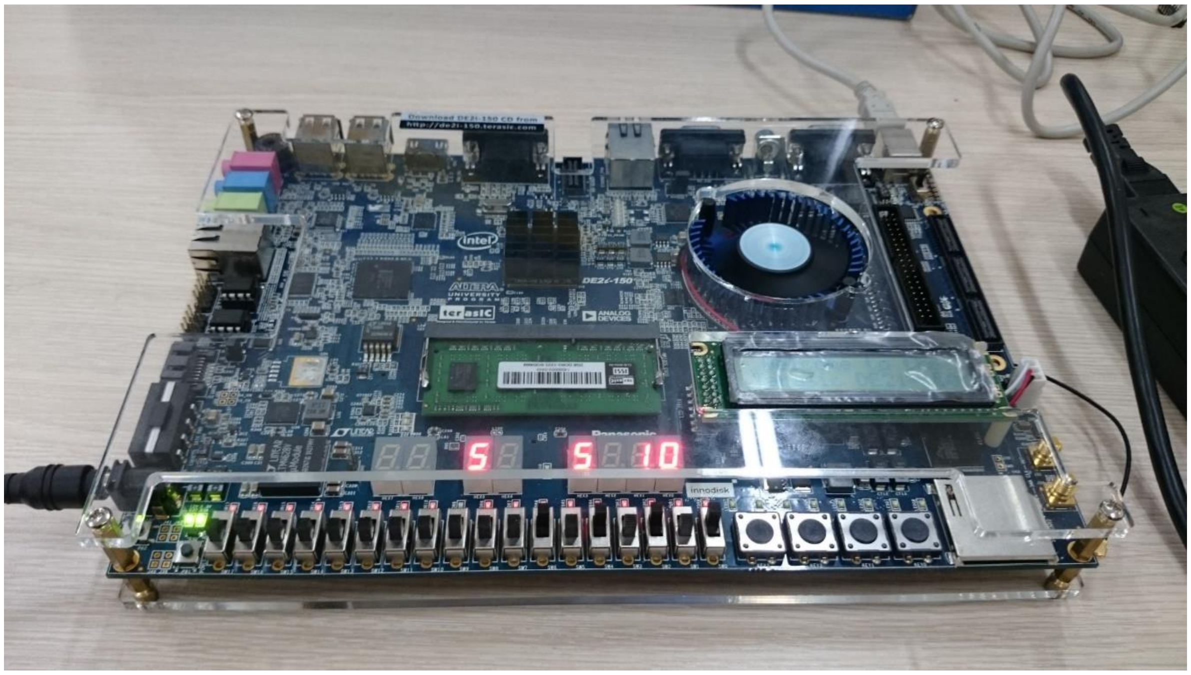
Testcase 1: SW8: 0 (0), SW7-4: 0001 (1), SW3-0: 0001 (1).

Result on DE2i-150 board: (0+1+1) 02.



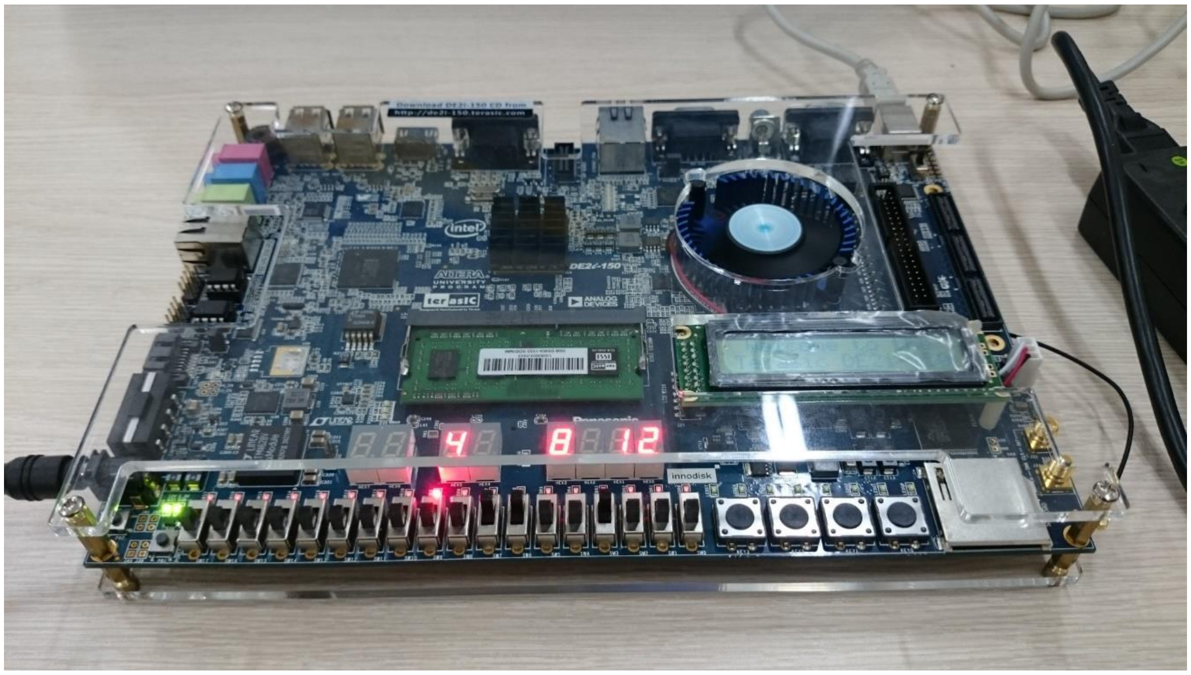
Testcase 2: SW8: 0 (0), SW7-4: 0101 (5), SW3-0: 0101 (5).

Result on DE2i-150 board: (0+5+5) 10.



Testcase 3: SW8 (Cin): 0, SW7-4 (A): 1100 (12 >9), SW3-0 (B): 1000 (8).

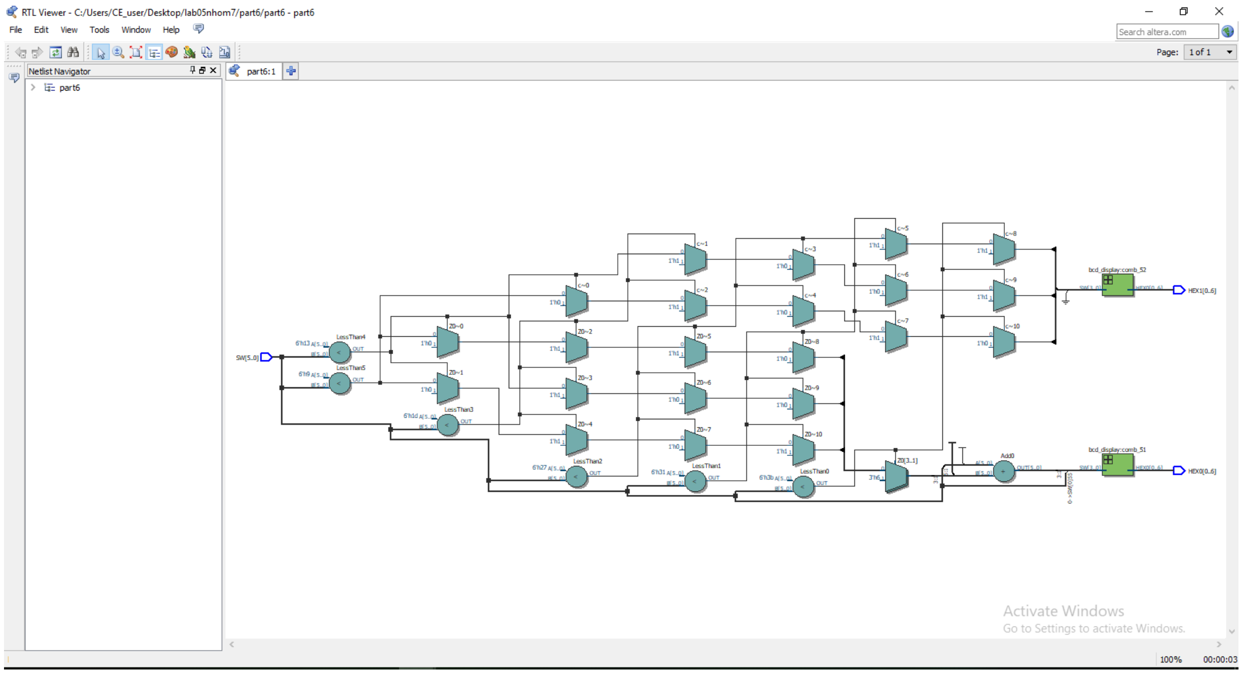
Result on DE2i-150 board: LEDR9 (error indicator): 1.



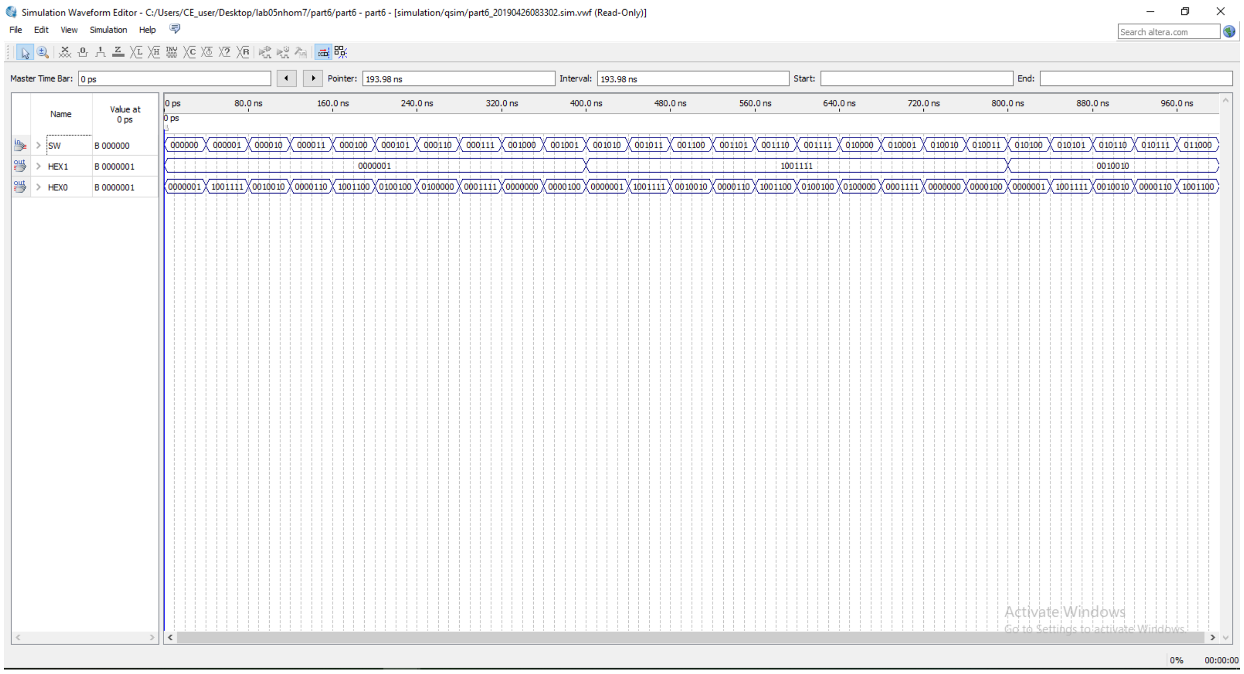
Part VI

Design a combinational circuit that converts a 6-bit binary number into a 2-digit decimal number represented in the BCD form. Use switches SW_{5-0} to input the binary number and 7-segment displays *HEX1* and *HEX0* to display the decimal number. Implement your circuit on a DE-series board and demonstrate its functionality.

Generated RTL Viewer:



Simulation picture:

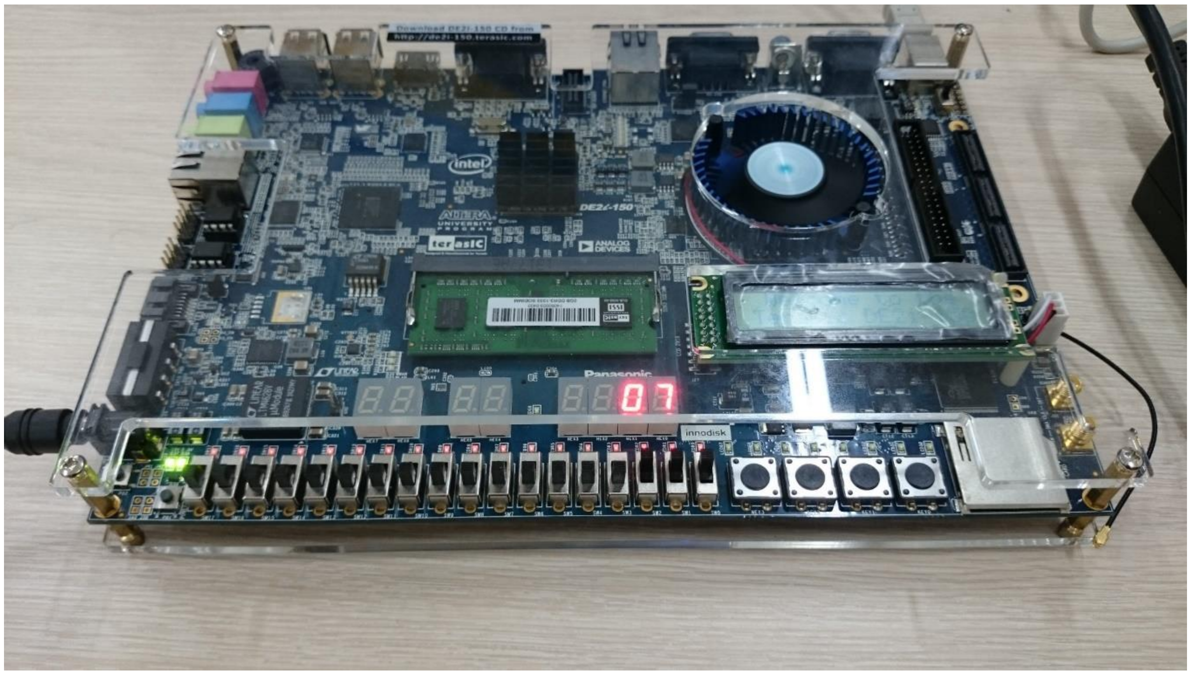


Simulation Waveform Editor - C:/Users/CE_user/Desktop/lab05nhom7/part6/part6 - part6 - [simulation/qsim/part6_20190426083302.sim.vwf (Read-Only)]

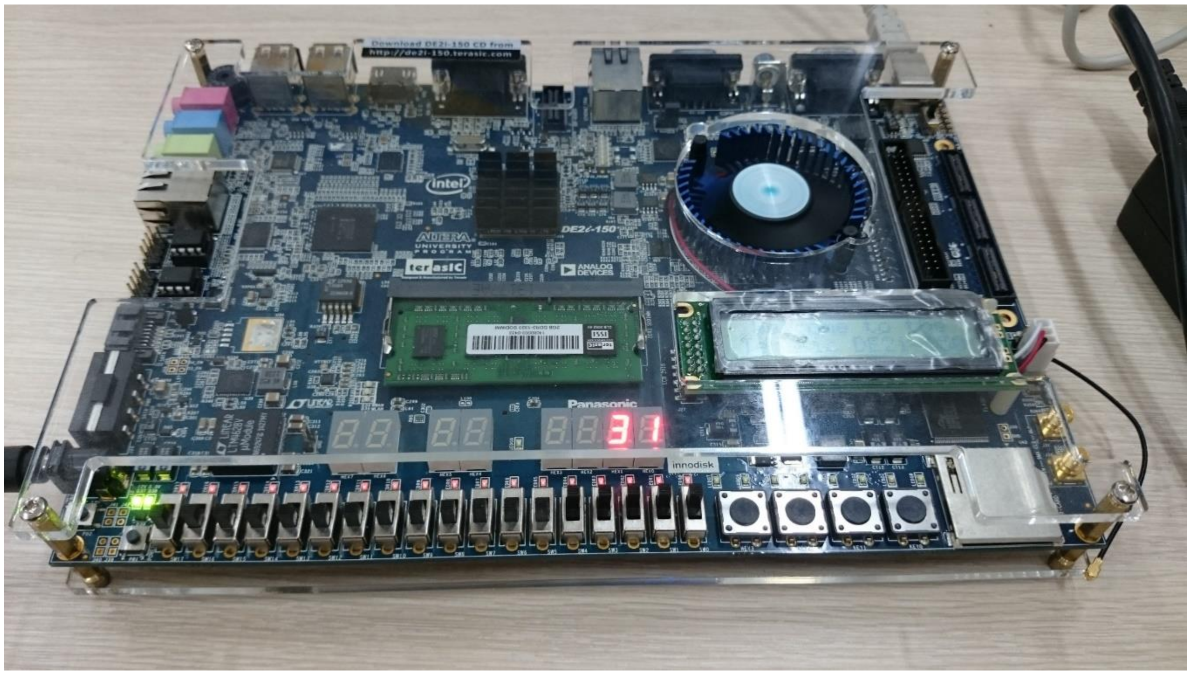
Master Time Bar: 0 ps Pointer: 193.98 ns Interval: 193.98 ns Start: End:

```
Name   Value at 0 ps   0 ps     80.0 ns     160.0 ns     240.0 ns     320.0 ns     400.0 ns     480.0 ns     560.0 ns     640.0 ns     720.0 ns     800.0 ns     880.0 ns     960.0 ns
SW     B 000000        000000 000001 000010 000011 000100 000101 000110 000111 001000 001001 001010 001011 001100 001101 001110 001111 010000 010001 010010 010011 010100 010101 010110 010111 011000
HEX1   B 0000001       0000001 1001111 0010010
HEX0   B 0000001       0000001 1001111 0010010 0000110 1001100 0100100 0100000 0001111 0000000 0000100 0000001 1001111 0010010 0000110 1001100 0100100 0100000 0001111 0000000 0000100 0000001 1001111 0010010 0000110 1001100
```

Testcase 1: SW5-0: 000111 (7). Result on DE2i-150 board: 07.



Testcase 2: SW5-0: 011111 (31). Result on DE2i-150 board: 31.



Testcase 3: SW5-0: 111111 (63). Result on DE2i-150 board: 63.