Laboratory Exercise 2

Numbers and Displays

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

Preparation

In advance of the lab exercise you are required to write the Verilog code for Parts I to IV. You should use simulation with ModelSim to verify the correct functionality of your code for each part of the exercise. You are required to show to your TA ModelSim simuations for Parts II and IV (either by having a printout or by runnning ModelSim on the lab computer). Using simulations will help to ensure that you are able to successfully demonstrate correct circuits in the lab for each part of the exercise. Also, for Part IV you are required to draw a circuit diagram (like the one that is given in Figure 1) and explain to your TA how this circuit works.

In-Lab

You are required to implement and test Parts I to IV of the exercise. You will only need to demonstrate Parts III and IV to your teaching assistant. Part V of the exercise is optional and is not graded. Doing Part V will provide you with more design experience and will be beneficial for later lab exercises.

Part I

You are to display on the LEDR lights the values of the switches *SW*. Also, you are to display the values of the switches SW_{7-0} on the 7-segment displays *HEX1* and *HEX0*, as digits from 0 to 9. Display the values of SW_{7-4} and SW_{3-0} on *HEX1* and *HEX0*, respectively. Your circuit should be able to display only the digits from 0 to 9, and should treat the valuations 1010 to 1111 as don't-cares.

1. Write a Verilog file that provides the necessary functionality. The intent of this exercise is to manually derive the logic functions needed for the 7-segment displays. Therefore, you should use only simple Verilog **assign** statements in your code and specify each logic function as a Boolean expression.

2. A *top.v* file is provided as part of the *design files* for this exercise, for use with the *DESim* tool. Compiling and simulating your code with *DESim* is a good way to see how the 7-segment displays will look when your circuit is implemented in a DE1-SoC board. To use the *top.v* file directly, declare your Verilog module as:

```
module part1 (SW, LEDR, HEX1, HEX0);
    input [7:0] SW;
    output [7:0] LEDR;
    output [6:0] HEX1, HEX0;
    ...
endmodule
```

3. After you have finished testing your Verilog code using simulation, create a Quartus project which will be used to implement your code on the DE1-SoC board. Make sure to include the required pin assignments on the FPGA to connect to the *SW* switches, *LEDR* lights, and 7-segment displays.

4. Compile the project and download the resulting circuit into the FPGA chip.

5. Test the functionality of your design by toggling the switches and observing the displays.

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 equivalent $D = d_1d_0$. Table 1 shows the required output values. A partial design of this circuit, which you need to complete, is given in Figure 1. It includes a module, called 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. The output z for the comparator circuit can be specified using a single Boolean expression, with the four inputs V_{3-0}. Design this Boolean expression by making a truth table that specifies the valuations of the inputs V_{3-0} for which z has to be 1.

$v_3v_2v_1v_0$	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.

Notice that the circuit 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 from V 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. Use *ModelSim* to simulate your circuit and check for correct operation of your comparator, multiplexers, and circuit *A*. An example of ModelSim setup files, including a *testbench*, are provided as part of the *design files* for this exercise.

3. A *top.v* file is provided in the design files for this exercise, for use with the *DESim* tool. As mentioned for Part I, using DESim is a good way to see what the 7-segment displays will show on a DE1-SoC board. To use the *top.v* file directly, declare your Verilog module as:

```
module part2 (SW, HEX1, HEX0);
    input [3:0] SW;
    output [0:6] HEX1, HEX0;
    ...
endmodule
```

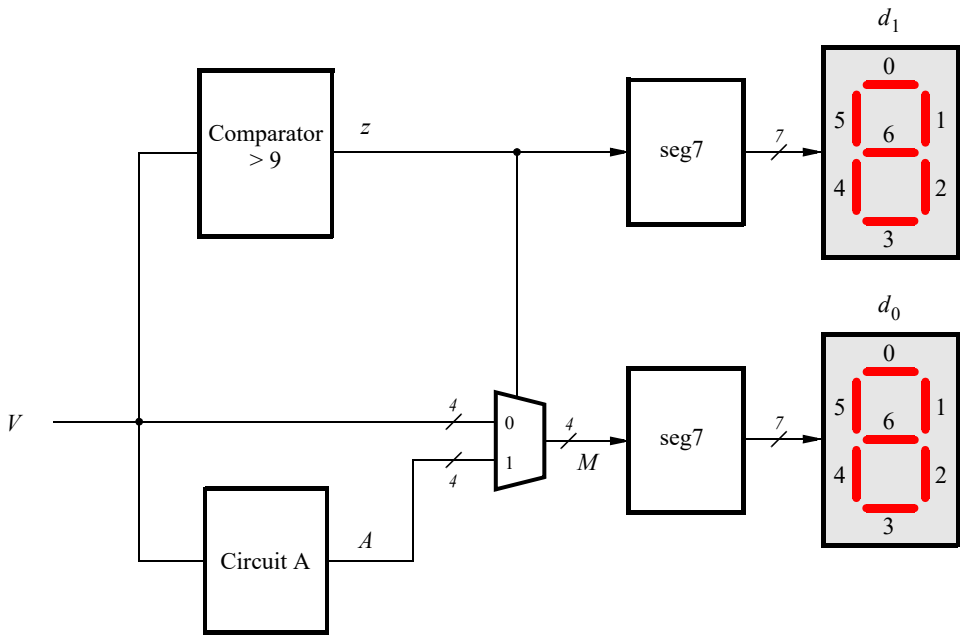


Figure 1: Partial design of the binary-to-decimal conversion circuit.

4. After you have finished testing your Verilog code using simulation, create a Quartus project for your Verilog module. Make sure to include the required pin assignments.

5. Compile your Verilog code in Quartus and download the circuit into an FPGA board. Test the circuit by trying all possible values of V and observing the output displays.

Part III

Figure 2a 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 2d 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.

Perform the steps given below.

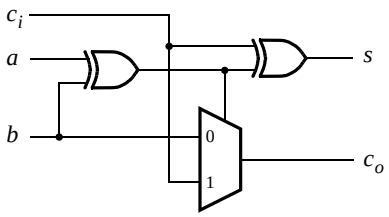
1. Write a Verilog module for the full adder subcircuit and write a top-level Verilog module that instantiates four instances of this full adder. Use switches SW_{7-4} and SW_{3-0} to provide the inputs A and B, respectively. Use SW_8 for the carry-in c_{in} of the adder. Connect the outputs of the adder, c_{out} and S, to the red lights LEDR.

2. You should use *ModelSim* to simulate your circuit and check for correct operation of the adder. An example of ModelSim setup files, including a *testbench*, are provided as part of the *design files* for this exercise.

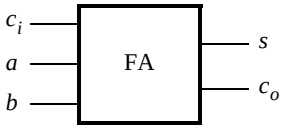
3. After completing simulations of your Verilog code, create a new Quartus project for this part of the exercise.

4. Include the necessary pin assignments for the DE1-SoC board, compile the circuit, and download it into the FPGA chip.

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



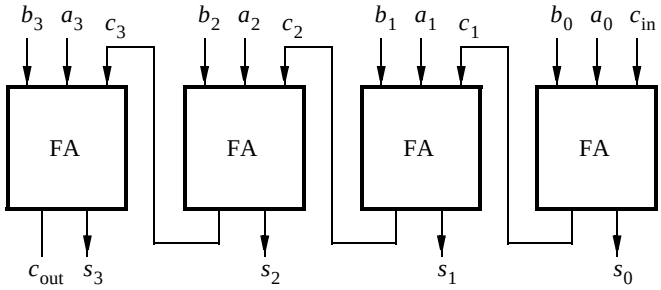
a) Full adder circuit



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.

Part IV

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 five-bit binary number. 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. Write Verilog code for the BCD adder. You must use only 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. One reasonable approach in designing this circuit is to use a modified version of the solution for Part II. In that part you had to display a four-bit number on the 7-segment displays. In this part you have to make a five-digit number by adding X + Y, and display that on the 7-segment displays. The five-bit sums 0 to 15 would already "just work" using the approach from Part II. But for sums 16, 17, 18, and 19 you would not directly display z (see Figure 1) on HEX1 and M on HEX0. Instead, for these sums (which are the sums for which the carry-out of X + Y is 1) you would have to provide different signals for the 7-segment decoders. Alternatively, there is a circuit in the recommended textbook that you could study, understand (so that you can explain it to your TA!), and use.

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 five-bit result produced by your circuit 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. Use ModelSim to simulate and debug your Verilog code. Make sure that you produce the correct output values for all possible results of the BCD addition.

5. A *top.v* file is provided in the design files for this exercise, for use with the *DESim* tool. To use the *top.v* file directly, declare your Verilog module as:

```
module part4 (SW, LEDR, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0);
    input [8:0] SW;
    output [9:0] LEDR;
    output [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
    ...
endmodule
```

6. After you have finished testing your Verilog code using simulation, create a new Quartus project for your BCD adder.

7. Include the necessary pin assignments for the DE1-SoC board, compile the circuit, and download it into the FPGA chip.

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

Part V

In Part IV you created Verilog code for a BCD adder by using only **assign** statements and Boolean logic expressions. 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   T0 = A + B + c0
2   if (T0 > 9) then
3       Z0 = 10;
4       c1 = 1;
5   else
6       Z0 = 0;
7       c1 = 0;
8   end if
9   S0 = T0 − Z0
10  S1 = c1
```

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. Write the required Verilog code following the style described above. Use the switches, lights, and 7-segment displays in the same way as you did for Part IV.

2. Create a Quartus project for your Verilog code. 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 the DE1-SoC board and test it.