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.

Part 1

We wish to display on the 7-segment displays *HEX3* to *HEX0* the values set by the switches SW_{15-0}. Let the values denoted by SW_{15-12}, SW_{11-8}, SW_{7-4} and SW_{3-0} be displayed on *HEX3*, *HEX2*, *HEX1* and *HEX0*, respectively. Your circuit should be able to display the hex digits from 0 to F. You can see that if we had a decoder module that took four bits of input and output the appropriate seven outputs for the 7-segment displays, then we would just have to use this circuit four times to achieve the desired result.

1. Start by designing the decoder module described above. You can use some of the behavioral techniques we discussed in class. You can test this circuit either with a ModelSim testbench or on your DE2 board directly.
2. Create a new project called Part1which will be used to implement the desired circuit on the Altera DE2-115 board.
3. Write a SystemVerilog file that provides the necessary functionality by instantiating your decoder four times, hooking the instances up to the appropriate switches (inputs) and 7-segment displays (outputs). Include this file in your project and assign the pins on the FPGA to connect to the switches and 7-segment displays, as indicated in the User Manual for the DE2 board. The procedure for making pin assignments is the same as in Homework 1. Be sure to treat unused pins as inputs, tri-stated.
4. You should have two modules for this part: your decoder and a top-level module called Part1.
5. Compile the project and download the compiled circuit into the FPGA chip.
6. Test the functionality of your design by toggling the switches and observing the displays.

Part 2

You are to design a circuit that converts a four-bit binary number $V = v_3 v_2 v_1 v_0$ into its two-digit decimal equivalent $D = d_1 d_0$. Table 1 shows the required output values. You are to complete the design of this circuit by creating a SystemVerilog module with four bits of input ('Binary value' from Table 1) and two four-bit outputs (the 'Decimal digits' from Table 1). Your module may include basic logic operations (AND, OR, etc.), comparators, (for instance

```
if ( x>9 ) // do something
```

),
and multiplexers. **Do not** use arithmetic functions such as multiply, or divide, or mod, etc. (you may use + and/or -,however). Test this module with a ModelSim testbench (make a modified runlab.do).

Binary value	Decimal digits	
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.

Perform the following steps:

1. Design and test the module described above.

2. Make a Quartus project for this part called Part2. Make a top-level module called Part2.sv

3. Use switches SW_{3-0} on the DE2-115 board to represent the binary number V, and the displays HEX1 and HEX0 to show the values of decimal digits d1 and d0. You will need one instance of the module from 1), above, and two instances of your decoder from Part I of this homework. Make sure to include in your project the required pin assignments for the DE2-115 board. Be sure to treat unused pins as inputs, tri-stated. Also include red LEDs 3-0 to show the state of your switches.

4. Compile the project, and then download the circuit into the FPGA chip (expect a 'stuck pins' warning).
5. Test your top-level circuit by trying all possible values of V and observing the output displays.

6. The runlab.do you turn in should run a testbench for the module you built for item #1, above. It should contain a line that calls the appropriate *wave.do for this module.

Part 3

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 SystemVerilog code that implements this circuit, as described below.

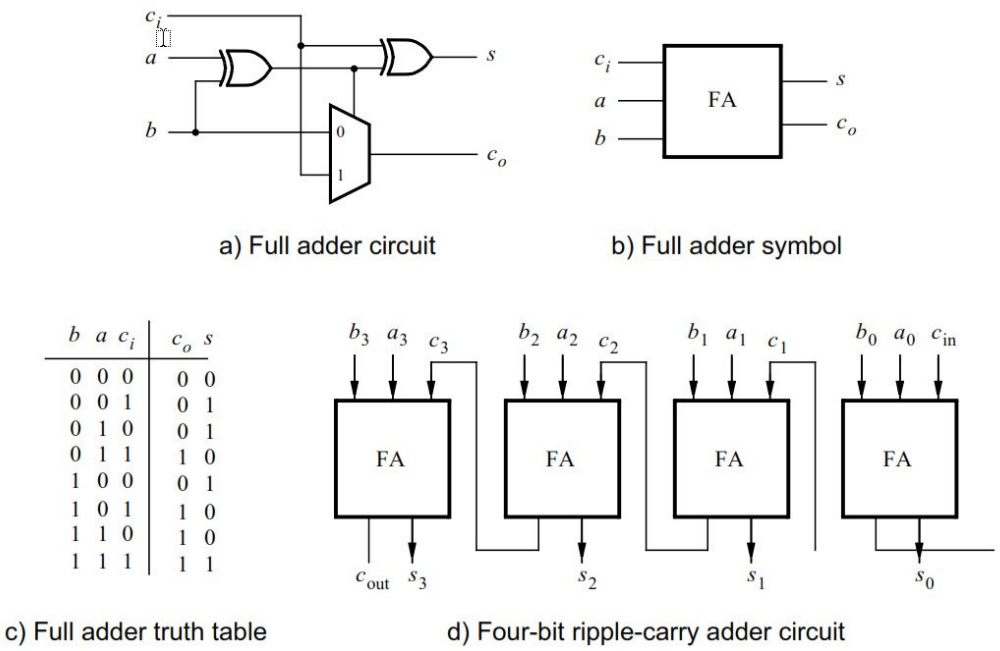

Figure 2. A ripple-carry adder circuit.

1. Create a FullAdder module and a FullAdder_testbench module. Test your adder with all possible input combinations.

2. Create a FourBitAdder module and a FourBitAdder_testbench module for the ripple-carry adder shown in Figure 2d. Test your module with all possible inputs combinations (!)

3. Create a new Quartus project called Part3 and a top-level module called Par3.sv that includes one instance of your 4-bit adder circuit.

4. 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 SW switches to their corresponding red lights LEDR, and display S on HEX0 and c_{out} on HEX1. Include the necessary pin assignments for the DE2 board, compile the circuit, and download it into the FPGA chip.

5. Test your circuit on the DE2-115 board by trying different values for numbers A, B, and c_{in}.

6. The runlab.do you turn in should run a testbench for your 4-bit adder module (item #2, above). It should contain a line that calls the appropriate *wave.do for this module.

Part 4

In part 2 we discussed the conversion of binary numbers into decimal digits. It is sometimes useful to build circuits that use this method of representing decimal numbers, in which each decimal digit is represented using

four bits. This scheme is known as the *binary coded decimal* (BCD) representation. As an example, the decimal value 59 is encoded in BCD form as 0101 1001.

You are to design a circuit that adds two BCD digits. The inputs to the circuit are BCD numbers A and B, plus a carry-in, c_{in}. The output should be a two-digit BCD sum $S_1\,S_0$. Note that the largest sum that needs to be handled by this circuit is $\mathbf{S_1\,S_0 = 9 + 9 + 1 = 19}$. Perform the steps given below.

1. Create a SystemVerilog module called BCD_Add that takes two BCD inputs plus carry-in and produces two BCD numbers, S_1S_0, as outputs. You can use the four-bit adder circuit from Part 3 to produce a four-bit sum and carry-out for the operation $A + B$. A circuit that converts this five-bit result, which has the maximum value 19, into two BCD digits $S_1\,S_0$ can be designed in a very similar way as the binary-to-decimal converter from Part 2. Include a BCD_Add_testbench that tests your module.

2. Create a new Quartus II project called Part4 for your BCD adder and create a top-level module called Part4.sv.

3. Use switches SW_{7-4} and SW_{3-0} for the inputs A and B, respectively, and use SW_8 for the carry-in. Connect the *SW* switches to their corresponding red lights LEDR, and connect the four-bit sum and carry-out produced by the operation $A + B$ to the 7-segment displays. Display the BCD values of A and B on the 7-segment displays *HEX6* and *HEX4*, and display the result $S_1\,S_0$ on *HEX1* and *HEX0*.

4. Since your circuit handles only BCD digits, check for the cases when the input A or B is greater than nine. If this occurs, indicate an error by turning on the green light $LEDG_8$.

5. Include the necessary pin assignments for the DE2 board, compile the circuit, and download it into the FPGA chip.

6. Test your circuit on the DE2-115 by trying different values for numbers A, B, and c_{in}.

7. The runlab.do you turn in should run a testbench for your BCD adder module (item #1, above). It should contain a line that calls the appropriate *wave.do for this module.

Part 5

Design a circuit that can add two 2-digit BCD numbers, $A_1\,A_0$ and $B_1\,B_0$ to produce the three-digit BCD sum $S_2\,S_1\,S_0$. Perform the steps below:

1. Make a module that uses two instances of your BCD_Add module from part 4 to build this two-digit BCD adder.

2. Test this adder using ModelSim.

3. Augment your circuit with a top-level module called Part5 that uses switches SW_{15-8} and SW_{7-0} to represent 2-digit BCD numbers $A_1\,A_0$ and $B_1\,B_0$, respectively. The value of $A_1\,A_0$ should be displayed on the 7-segment displays *HEX7* and *HEX6*, while $B_1\,B_0$ should be on *HEX5* and *HEX4*. Display the BCD sum, $S_2\,S_1\,S_0$, on the 7-segment displays *HEX2*, *HEX1* and *HEX0*. Red LEDs should show the state of the switches.

4. Make the necessary pin assignments and compile the circuit.

5. Download the circuit into the FPGA chip, and test its operation.

6. The runlab.do you turn in should run a testbench for your adder module (item #1, above). It should contain a line that calls the appropriate *wave.do for this module (if you use one).

Part 6

In part 5 you created Verilog code for a two-digit BCD adder by using two instances of the Verilog code for a one-digit BCD adder from part 4. A different approach for describing the two-digit BCD adder in Verilog code is to specify an algorithm like the one represented by the following pseudo-code:

```
 1 T0 = A0 + B0
 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 T1 = A1 + B1 + c1
11 if (T1 > 9) then
12   Z1 = 10;
13   c2 = 1;
14 else
15   Z1 = 0;
16   c2 = 0;
17 end if
18 S1 = T1 - Z1
19 S2 = c2
```

It is reasonably straightforward to see what circuit could be used to implement this pseudo-code. Lines 1, 9, 10, and 18 represent adders, lines 2-8 and 11-17 correspond to multiplexers, and testing for the conditions $T_0 > 9$ and $T_1 > 9$ requires comparators. You are to write SystemVerilog code that corresponds to this pseudo-code. Perform the following steps:

1. Create a SystemVerilog module called BCD_Add2 that implements the algorithm shown above. Include the testbench you wrote in Part V and test your module using ModelSim.
2. Create a new Quartus project called Part6. Copy your top-level module from Part5 here and rename it to Part6. Use the same switches, lights, and displays as in part V. Compile your circuit.
3. Use the Quartus II RTL Viewer tool to examine the circuit produced by compiling your Verilog code. Compare the circuit to the one you designed in Part V.
4. Download your circuit onto the DE2 board and test it by trying different values for numbers $A_1 A_0$ and $B_1 B_0$.
5. The runlab.do you turn in should run a testbench for the module you built (item #1, above). It should contain a line that calls the appropriate *wave.do for this module, if you use one.