B EE 427 PA #1
Due: 01/23/2023 3:30pm

Note: There are three parts in this PA. Create a Quartus project file for each part. You should submit your Verilog code to Canvas. In addition, you should show me your demo using the DE1-SoC board. There is no report for this PA.

Part 1

You are to design a circuit that converts a four-bit binary number *V* = $v_3v_1v_1v_0$ into its two-digit decimal equivalent D = d_1d_0. Table 1 shows the conversion values.

Input (binary)				Output (decimal)	
v_3	v_2	v_1	v_0	d_1	d_0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	2
…				…	
1	0	0	1	0	9
1	0	1	0	1	0
1	0	1	1	1	1
1	1	0	0	1	2
…				…	
1	1	1	1	1	5

Table 1: Binary-to-decimal conversion 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.

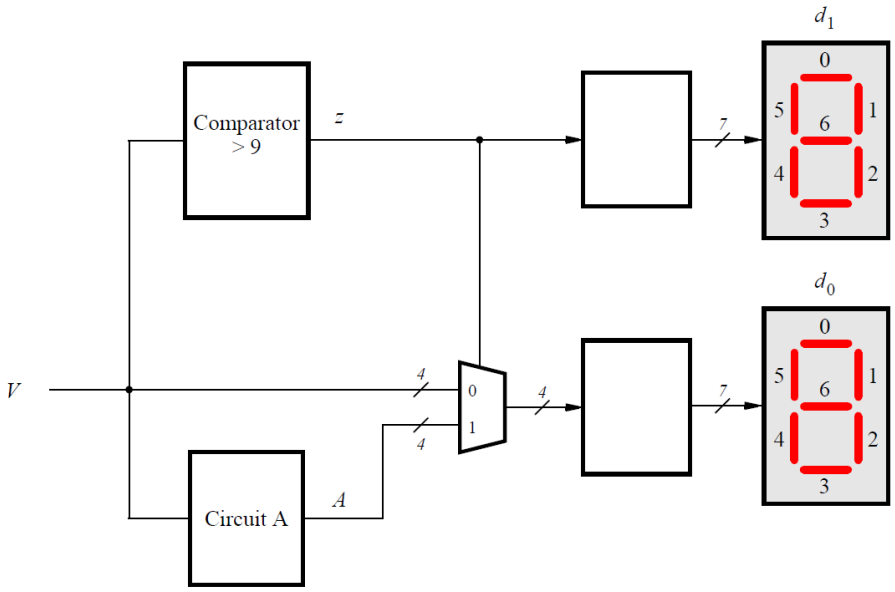


Figure 1: Partial design of the binary-to-decimal conversion circuit.

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 shows the valuations of the inputs V_{3-0} for which *z* has to be 1.

Notice that the circuit in Figure 1 includes a 4-bit wide 2-to-1 multiplexer. 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:

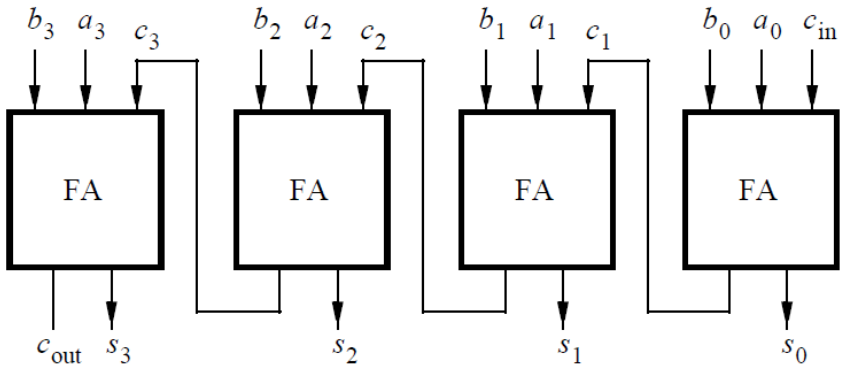
1. Create a new Quartus project using the system builder program. You'll use switches and 7-segment displays for Part 1.
2. Write Verilog code to implement your design. The code should have the 4-bit input SW_{3-0}, which should be used to provide the binary number *V*, and the two 7-bit outputs HEX1 and HEX0, to show the values of decimal digits d_1 and d_0. The intent of Part 1 is to use simple Verilog assign statements to specify the required logic functions using Boolean expressions, so your Verilog code **should not include any if-else, case, or similar statements**.
3. Compile your code and download it onto the DE1-SoC board. Test the circuit by trying all possible values of *V* and observing the output displays.

Part 2

Figure 2(a) shows a truth table for the *full adder*, which produces the two-bit binary sum $c_o s = a + b + c_i$. Figure 2(b) 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 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.

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

(a)



(b)

Figure 2: (a) A truth table of the full adder. (b) A 4-bit ripple-carry adder circuit.

Perform the following steps:

1. Create a new Quartus project using the system builder program. You'll use switches and LEDs for Part 2.
2. Write a Verilog module for the full adder sub-circuit and write a top-level Verilog module that includes four instances of this full adder.
3. 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 $LEDR_{4-0}$.
4. Compile the circuit and download it onto the DE1-SoC board.
5. Test your circuit by trying different values for numbers a, b, and c_{in}.

Part 3

In Part 1, we discussed the conversion of binary numbers into decimal digits. For this part, you are to design a circuit that adds the two binary coded decimal (BCD) digits. The inputs to your circuit are the 4-bit numbers *A* and *B*, plus a 1-bit 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 *A* and *B* 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.

One 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 additions in your Verilog code instead of the subtraction shown in line 9. Use the ripple-carry adder module designed in Part 2 (you may need to increase the bit-width from 4 to 5).

Perform the following steps:

1. Create a new Quartus project for your Verilog code. You'll use switches and 7-segment displays for Part 3.
2. 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.
3. Download your design onto your DE1-SoC board and test it by trying different values for numbers *A* and *B*.

Reference

[1] Intel, "Teaching Materials for the Intel FPGA Academic Program"

Supplement: Truth table for Part 1

Input				Output								
				Decimal		HEX0 for d_0						
v_3	v_2	v_1	v_0	d_1	d_0	[6]	[5]	[4]	[3]	[2]	[1]	[0]
0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	1	0	1	1	1	1	1	0	0	1
0	0	1	0	0	2							
0	0	1	1	0	3							
0	1	0	0	0	4							
0	1	0	1	0	5							
0	1	1	0	0	6							
0	1	1	1	0	7							
1	0	0	0	0	8							
1	0	0	1	0	9							
1	0	1	0	1	0	1	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	0	0	1
1	1	0	0	1	2							
1	1	0	1	1	3							
1	1	1	0	1	4							
1	1	1	1	1	5							

Comparator result z	HEX1 for d_1						
	[6]	[5]	[4]	[3]	[2]	[1]	[0]
0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	1

HEX1[6] = 1

HEX1[5] =

HEX1[4] =

HEX1[3] =

HEX1[2] = 0

HEX1[1] = 0

HEX1[0] =

HEX0[6] =

HEX0[5] =

HEX0[4] =

HEX0[3] =

HEX0[2] =

HEX0[1] =

HEX0[0] =