

# 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 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 displays. 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.

### 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 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  $V_{3-0}$ . 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.

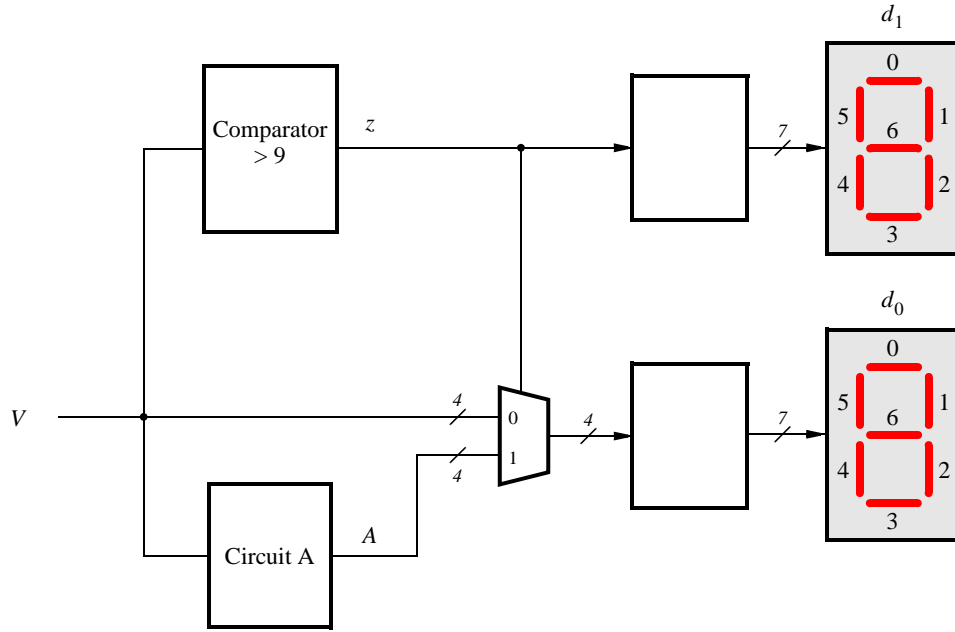


Figure 1: Partial design of the binary-to-decimal conversion circuit.

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, multiplexers, 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.

### 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. Write Verilog code that implements this circuit, as described below.

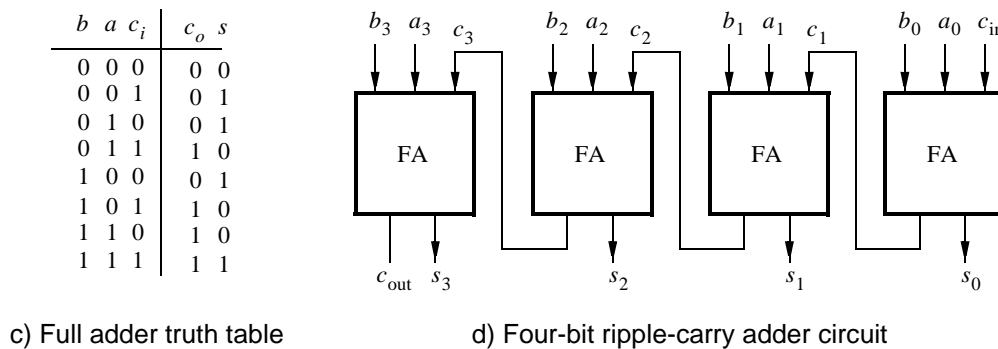
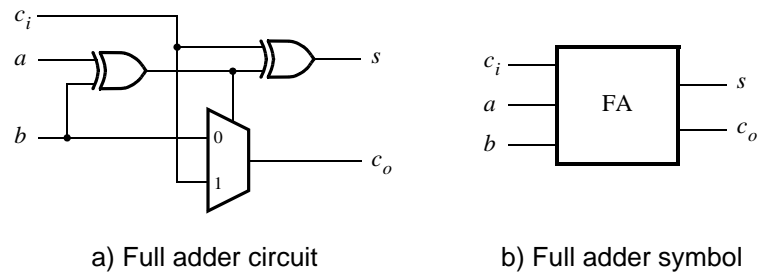


Figure 2: A ripple-carry adder circuit.

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  $c_{in}$ .

## 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  $c_{in}$ .

## 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  $B$ .

## 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.

Copyright © 1991-2016 Intel Corporation. All rights reserved. Intel, The Programmable Solutions Company, the stylized Intel logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Intel Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Intel products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Intel warrants performance of its semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel Corporation. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

This document is being provided on an "as-is" basis and as an accommodation and therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.