UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering

EEC 18 **DIGITAL SYSTEMS I** **Fall 2021**

LAB 3: Combinational Network Design Using Muxes and Adders

Objective

This lab is an exercise in designing combinational circuits using multiplexers and adders. It includes creating symbols to make your schematic hierarchical and easier to understand, simulating your design in ModelSim, and verifying your circuit on the DE10-Lite board.

Preparation (Pre-lab)

Before coming to the first lab session, complete the following tasks:

- Read handouts and textbook material covering adders (Unit 4.7) and multiplexers (Unit 9.2)
- Generate a truth table showing inputs vs outputs for the following circuit blocks in Part I: Comparator, Circuit A, and Circuit B.
- Use the truth tables to produce minimized SoP (sum of products) for the Comparator, Circuit A and Circuit B.

Part I – Simple Binary to BCD Conversion

Design Specifications

Design a circuit that converts a four-bit binary number V[3:0] = {V[3], V[2], V[1], V[0]} into its two-digit decimal equivalent D[1..0] = D[1] D[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[3:0] is greater than 9 (z=0 when V≤9, z=1 when V≥10), and uses the output of this comparator in the control of the 7-segment displays. Complete the design of this circuit by creating a schematic which includes the comparator, multiplexers, circuit A, and circuit B. You may use a 7447 component for the 7-segment decoder.

Binary value	Decimal digits
0000	00
0001	01
0010	02
0011	03
…	…
1001	09
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Table 1. Binary-to-decimal conversion values

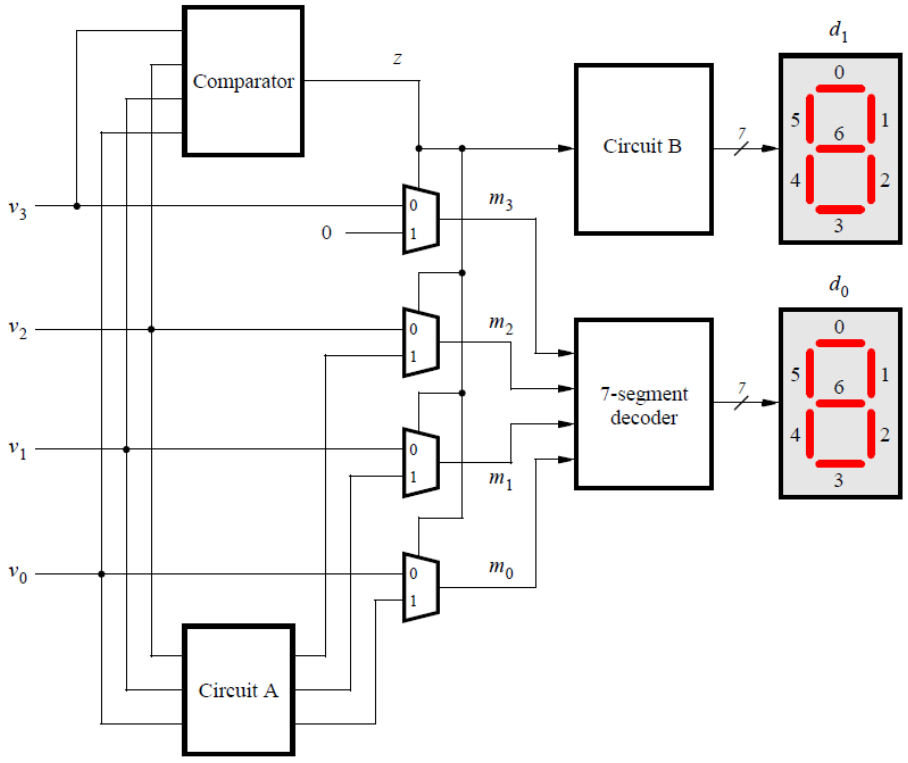


Figure 1. Partial design of the binary-to-decimal conversion circuit where d_1 is the "tens" digit and d_0 is the "ones" digit.

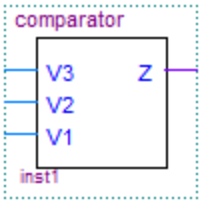
The number displayed in d_0 is represented by {m3, m2, m1, m0}. When the input number is equal or less than 9 (i.e., $z = 0$), the four multiplexers select the 4-bit input number V[3:0], and Circuit A is don't care; when the input number is greater than 9 (i.e., z = 1), Circuit A outputs are selected for m2, m1, and m0; and m3 is equal to 0 when z is equal to 1. For example, when the input number V[3:0] = 10 (i.e., 1010), circuit A output should be 000; when the input number V[3:0] = 15 (i.e., 1111), circuit A output should be 101, since the number '5' should be displayed in d_0. The equations of Circuit A can be derived using a truth table and K-maps. Circuit B is a simplified 7-segment decoder that only decodes 0 and 1. It has 1-bit input (the comparator output z) and 7-bit outputs (each output bit is one segment of HEX display d_1). d_1 should display a '0' when z is equal to 0, and display a '1' when z is equal to 1.

Design Procedure

1. Make a new Quartus project, such as **lab3a**, for your design.

2. Draw a schematic for the complete circuit given in Figure 1, observing the following points:
   - Use **SW[3:0]** as inputs for V[3:0].
   - Use **HEX1** for d_1 and **HEX0** for d_0.
   - Design your own 2-to-1 multiplexer using logic gates.
   - Create a symbol for the 2-to-1 multiplexer, Comparator, and Circuit A sub-circuits as follows:
     i. Create a new Block Diagram/Schematic File (.bdf) and enter the circuitry for one of the blocks, such as the Comparator. The inputs and outputs of the circuit must use INPUT and OUTPUT components, but these should not be names of FPGA pins such as SW[3]…SW[0].
     ii. When your circuit is complete, save it with the name of the block such as comparator.bdf.
     iii. From the Quartus pull-down menus, select **File > Create/Update > Create Symbol Files for Current File**. This will generate a symbol file, comparator.bsf, for your schematic.
     iv. In the top-level design, you can use the Symbol Tool (or double-click on open area) and specify the Comparator component that you just created. It should be located in your project

directory. (Try not to use a name of an existing Altera component for your symbol/circuit.) Your component should look similar to the one shown below.



- Since the circuitry for Circuit B will be very simple, you can keep it on the top-level schematic rather than creating a symbol for it.
- Note that all circuit I/O pins, such as HEX1[6]..HEX1[0], SW[3]..SW[0], HEX0[6]..HEX0[0] should be on the top-level schematic.
- Import the pin assignments for the DE10-Lite board. You can use the lab1.qsf file that you created using System Builder for lab 1, or you can generate a new file following the procedure described in Lab 1.

3. Compile your circuit in Quartus and fix any errors.

4. Do a ModelSim simulation of your design by selecting **Tools > Run Simulation Tool > Gate Level Simulation** in Quartus and then select **Simulate > Start Simulation** in ModelSim. Select the **altera_ver** and **fiftyfivenm_ver** libraries, as in previous labs.

    - Right-click on SW in the Objects window and select **Add Wave**. Do the same for HEX0 and HEX1.

    - Set the Radix for HEX0 and for HEX1 to hexadecimal.

    - Right-click on SW in the Wave window and select **Force…** and force the SW signal to 0000. Simulate for 20 ns and verify the output.

    - Continue to simulate the circuit through the full sequence of SW inputs, 0000 – 1111. Print a simulation waveform showing just one complete cycle of the inputs.

    - Demonstrate your simulation to your TA and have him or her sign your verification sheet.

5. Download your circuit to your DE10-Lite board and verify your circuit using SW3 – SW0. Verify the outputs on HEX1 and HEX0. Demonstrate your circuit to your TA and have him or her sign your verification sheet.

Part II – Ripple Carry Adder Using Full Adder Circuit

Design Specifications

In this part, you will use a Full Adder (FA) circuit to design a three-bit ripple carry adder circuit. Figure 2a shows one possible circuit for a Full Adder, which has the inputs a, b and ci, and produces the outputs s and co.

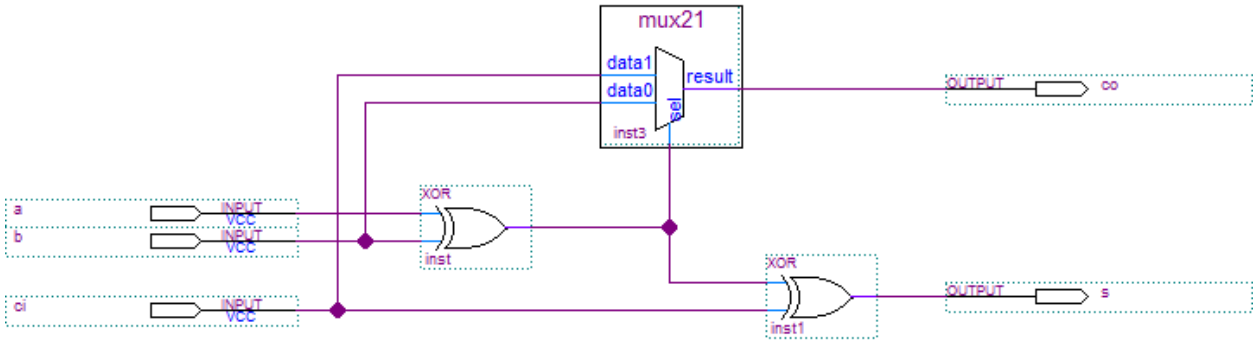


Figure 2a. Full adder circuit

Note that this circuit has been built using the lpm_mux component from the Altera libraries. You could also use the 74157 component, if you prefer. In order to use the lpm_mux component, select Tools > IP Catalog and type mux in the search box. This should locate the component lpm_mux. Double-click on it and specify an IP variation file name in the dialog box. For example, the lpm_mux component in Figure 2a has the IP variation file name mux21. Next specify the number of data inputs (2) and the width on the data input and result output (1). There is no need to pipeline the multiplexer. On the last configuration page, check the box for mux21.bsf to generate a Quartus Prime symbol file so that you can use mux21 as a symbol in your schematic, as shown in Figure 2a. Click Yes to add the Quartus IP file to your project. Now, you can use the mux21 symbol in your schematic.

You will create a symbol for your FA circuit and use three FA modules to build a 3-bit ripple-carry adder, as shown in Figure 2b.

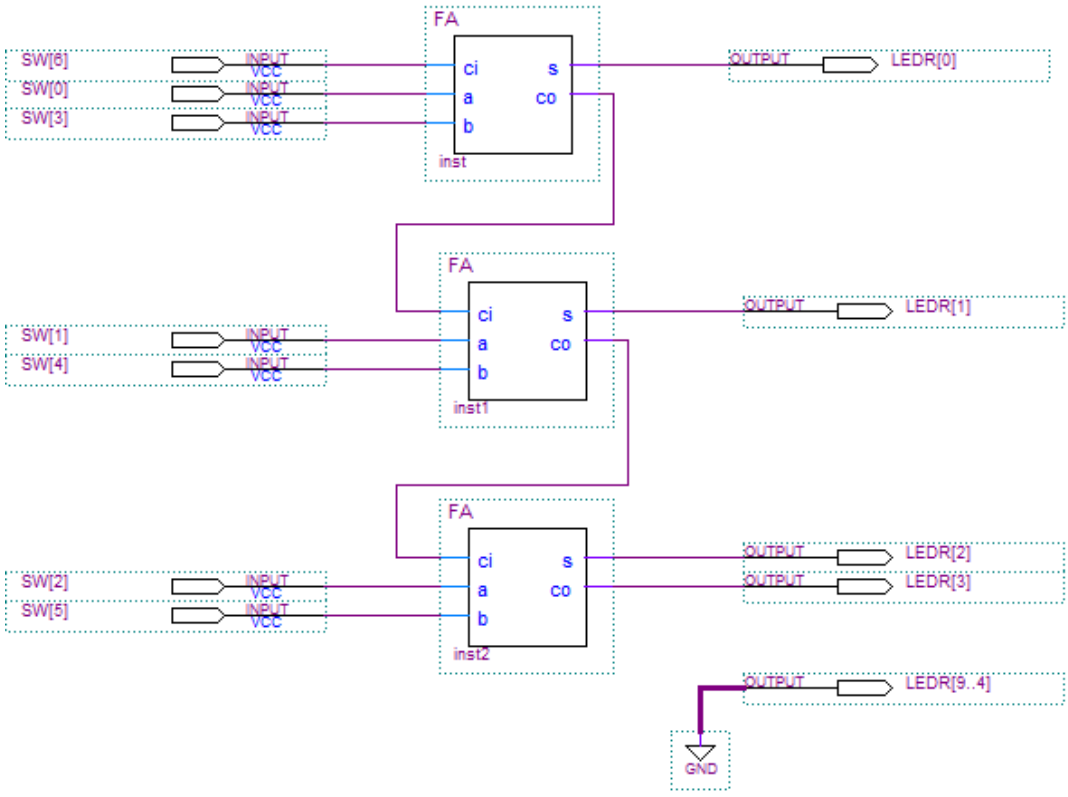


Figure 2b. Three-bit ripple-carry adder

Design Procedure

1. Make a new Quartus project, such as **lab3b**, for your design.

2. Draw a schematic for the complete circuit given in Figure 2a and 2b, observing the following points:

    - Create a symbol for your Full Adder (FA) circuit.
    - Use SW[2..0] for a[2..0], SW[5..3] for b[2..0] and SW[6] for ci.
    - Use LEDR[3:0] for the four-bit output. Turn off LEDR[9..4].

3. Compile your circuit in Quartus and fix any errors.

4. Do a ModelSim simulation of your design. Set the SW radix to octal and the LEDR radix to hexadecimal to make it easy to check your circuit. (Why does octal make sense for the SW inputs?) You do not have to do an exhaustive simulation of the circuit. Instead, choose at least 15 different input combinations to simulate. Simulate each input case with a step size of about 20 ns.

    - Demonstrate your simulation to your TA and have him or her sign your verification sheet.

5. Download your circuit to your DE10-Lite board and verify your circuit using SW6 – SW0. Verify the outputs on LEDR3-LEDR0. Demonstrate your circuit to your TA and have him or her sign your verification sheet.

Part III – Ripple Carry Adder With Decimal Display

Design Specifications

In this part, you will combine your circuits from Parts I and II to create a three-bit ripple-carry output with the output displayed in decimal. Note that the largest sum that needs to be displayed by this circuit is 7 + 7 + 1 = 15.

Design Procedure

1. Make a new Quartus project, such as **lab3c**, for your design.

2. Combine the schematics from Parts I and II into a new schematic. Make sure all of the FPGA I/O pins such as SW[] and HEX1[] and HEX0[] are on the top-level schematic. You do not have to display the output on LEDR, although you can if you want. Turn off all unused LEDR.

    - **Import the pin assignments in the new directory**. When you copy and paste from another schematic, the pin assignments will not be copied over.

3. Compile your circuit in Quartus and fix any errors.

4. Download your circuit to your DE10-Lite board and verify your circuit using SW6 – SW0. Verify the outputs on HEX1 and HEX0. Demonstrate your circuit to your TA and have him or her sign your verification sheet.

Lab Report

Each individual will be required to submit a lab report. Use the format specified in the "Lab Report Requirements" document available on the class web page. Be sure to include the following items in your lab report:

- Lab cover sheet with TA verification for circuit simulation and circuit performance
- Graded pre-lab
- Quartus schematics for Parts I, II and III
- Simulation waveforms for Parts I and II
- Based on the design in Figure 2a, draw a truth table for your full adder circuit. Use K-maps to generate minimized sum of products equations for co and s and compare them to the circuit in Figure 2a. Is there an advantage to Figure 2a compared to the SoP equations?
- What is the 'octal' radix and why is it convenient to represent the SW[] input in octal in the Part II simulation? Why isn't octal convenient for the LEDR[] output?

Acknowledgement: This lab is based on a Laboratory Exercise developed by Altera Corp. Used by permission.

2021/09/30 Posted
2021/10/14 Added background reading to pre-lab