# SE 2DA4 Lab 3

First lab Week of: Oct. 21, 2019
Prep Due week of: (8:40/14:40), Oct. 28, 2019
Demo Due Week of: (11:20/17:20), Oct. 28, 2019
Assigment due in class: 17:40, Nov. 7, 2019

## Announcements:

**Note 1:** all lab due dates/times are for your scheduled lab in the indicated week.

**Note 2:** The 3rd edition of the text no longer contains the Altera tutorial appendices (B, C, and D) that were in the 2nd edition. They can be downloaded from:
http://www.cas.mcmaster.ca/~leduc/slides2d04/labs/2ndEdAppdx/

**Note 3:** Lab 3 will be a lot more complicated and longer than lab 2. It's important that you allocate enough time for this lab, and not leave it to the last minute. In the past several groups thought the lab would be easy like lab 2, left it till later, and never finished the lab.

**For this and every lab:** after you have demonstrated your lab to a TA , you must e-mail (as per instructions in lab 1) the final versions of your Verilog files (the ones demoed to the TA) as attachements to the following e-mail address: `rlta1@cas.mcmaster.ca`

## Part 1: Assignment

**Note:** You must put your **lab section** on the top of your assignment as they will be handed back during your lab.

The following are relevant textbook questions to be handed in 17:40, Nov. 7, 2019:

1) Do Ch. 3 # 1,3,4 and check them against the solns in the text. You do not need to hand this in.

2) Apply the instructions for exercise 3.5 in text to the operations below. Assume that the second row of additions contain only 6 bit numbers.

| | | |
|---|---|---|
| 00000110<br>+00001101 | 11111010<br>+00001101 | 11111010<br>+11110011 |
| 110001<br>−010010 | 101110<br>−110111 | |

3) Ch. 3 # 22 (hint: build upon soln for ex 3.21).

4) Verify that Figure 1 below implements a full adder. Use Boolean algebra.

Extra practice (not to hand in!): Convert the following to decimal (assume unsigned) a) $(1001110001)_2$ b) $(127.4)_8$ c) $(B65F)_{16}$ d) convert to binary: $(2C6B)_{16}$ e) convert to hexadecimal: $(10110001101011)_2$

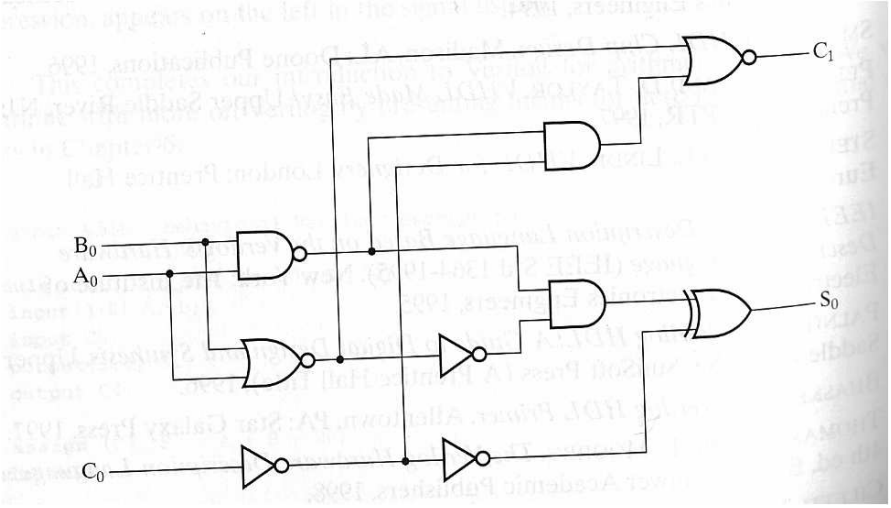


Figure 1: Circuit for Q 4.

# Part 2: Hierarchical Design, Adders & Registers

1. Purpose

   The purpose of this lab is to introduce sequential circuits such as flip-flops and registers, create an adder, and use it in combination with registers to implement a simple ALU. The lab also provides practice in coding Verilog. As a side effect of building bigger things from smaller things, we illustrate the concept and use of hierarchy in design.

2. Background

   To learn about D type flip flops, see introduction to Chapter 5, and Section 5.4. See also Appendix A, Sections A.14.2, A.14.3, and A.14.4, plus Figures A.27 and A.30. For ripple-carry adders see Sections 3.2-3.2.2, and Sections 3.5.2-3.5.3 for Verilog implementations. You may also find Section C.2 of Appendix C (2nd edition) useful.

3. Preparation

   As preparation, hand in schematics (if any), Verilog code, and simulations from steps 4 and 5 (excluding steps 4b and 4.c as you did this already in lab 2). Also, the results of step 5.a. You should arrive at the lab with your preparation ready to hand in. Do not try to print it out at the start of the lab period.

4. Tool Use

Design and simulate (using timing simulation) the following circuits. Remember that for large circuits, it's often not realistic to simulate all possible input combinations. Instead, come up with a set of conditions to check that will test the circuit, making you confident that it works correctly.

**a)** Use Verilog with the CASE statements and logic vectors as described in the Verilog reference manual (Appendix A: A.11.1, A.11.3, A.11.4, and Figure A.19) and also, in particular, Ch. 4, Sec 4.6.3, to implement a hexadecimal 7-segment display driver. Essentially, for the decimal numbers 0 to 15 (corresponding to 4 bit binary numbers 0000 to 1111 that you will actually get as input), you will display the corresponding hexadecimal digits 0 to F. See table 3.1 of text to determine the correct correspondance between the two.

Use a file called `hex7seg.v` for this circuit. (NOTE: the code in Figure 4.34 for the 7-segment display assumes an active high signal for the LEDs but the display on the Altera boards is active low).

Compile your Verilog code for Cyclone V FPGA (5CSEMA5F31C6) device found on the DE1-SoC1 board and perform a simulation.

**b)** Build and simulate an edge triggered D type flip flop as described in the Verilog Reference guide, Section A.14.2 (see Figure A.27) of Appendix A of the text. You should have done this already in lab 2 (ie. if so, you don't need to redo this).

**c)** In the same fashion, build and simulate a 3-bit D register with an asynchronous reset signal as described in Section A.14.3 (see Figure A.30) and A.14.4. You should have done this already in lab 2.

5. Design Problem: A Simple ALU

Using the Verilog for a seven segment decoder and the 3-bit register (with reset) that you built in 4, you will build an adder unit, as shown in Figure 2, whose inputs are: a single three bit binary number ($X_2X_1X_0$), the Clock, and the Reset signal. The four output bits of the adder are connected to the 7-segment display so that it can be viewed. In the preparation below, you are required to figure out how to use this circuit to add two three bit numbers. This is actually a simple version of the arithmetic and logic unit that can be found inside all computers.

**a)** Determine the sequence of control signals and data inputs that you would need to be able to add two 3-bit numbers using the circuit of Figure 2. You will need to know this in order to properly simulate the circuit that you will be designing. The sequence is essentially a timing diagram (giving the sequence of the input values $X_2X_1X_0$, Reset and Clock) such as you would create using the waveform editor of Quartus to test the circuit. In other words, submit the sequence in the form of a timing diagram.

**b)** Design, using Verilog, a 3-bit ripple carry adder. You should design the adder as a basic logic function - do not use the built in addition capability of Verilog. Simulate the adder to be sure that it works. Note that a 3-bit adder has two 3-bit inputs and one four bit output ($C_{out}$ plus 3 sum bits).

**c)** Using the adder, two 3-bit D type registers with reset, and the 7-segment decoder, build the circuit of Figure 2 as a hierachical circuit using Verilog. Design and simulate the circuit using Verilog only in a file called `alu3.v` (top level file).

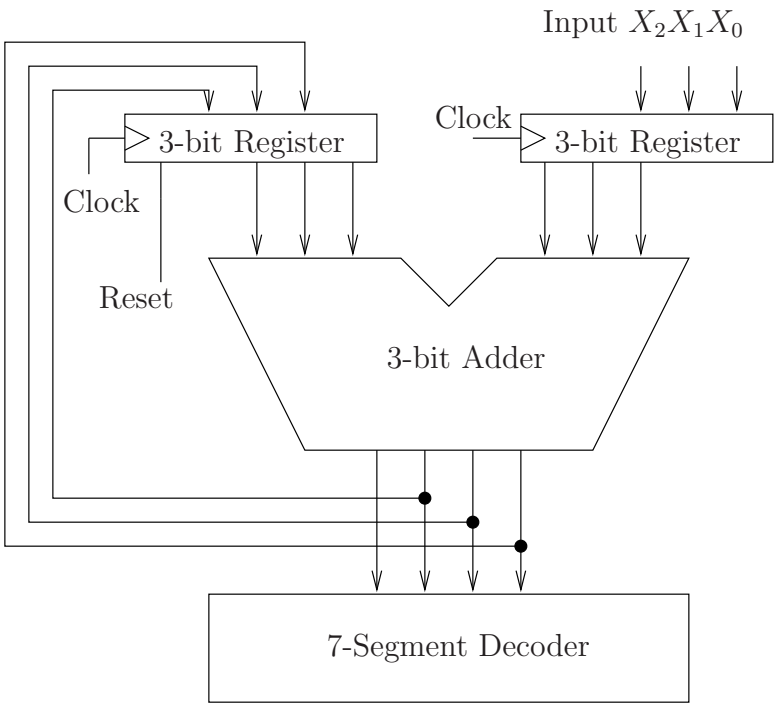


Figure 2: Adder Unit Schematic

**HINT:** when testing a hierarchical circuit like this, first do a detailed simulation for each component design to verify that they work correctly. Ideally, you would test each component on the alterra board as well. When you combine the components into one circuit, you can take for granted that the components work properly; what you have to test now is that they work correctly together. Again, for more complicated circuits, this does not have to be an exhaustive test.

Demonstrate this circuit to the TAs by compiling it and downloading it to the DE1-SoC board. Use toggle switches SW[9], SW[8], and SW[7] for inputs $X_2, X_1, X_0$ respectively. Use one HEX led as output to display the alu result. Connect the reset signal to pushbutton KEY[1], and the clock to pushbutton KEY[0] (active low pushbutton) to generate reset or clock pulses.

**NOTE:** The switches are going to be noisy, but noise on the X inputs are probably not going to be a problem as long as they have settled before you give a clock pulse.

However, if the clock is noisy, you may latch data too many times and get the wrong answer. Fortunately, the push buttons on DE1-SoC boards are debounced in hardware. See page 23 of the DE1-SoC user manual for more details.