ECE241 - Digital Systems
University of Toronto

# Lab #5 - Fall 2008
# Adders, Registers and Counters

## 1. Introduction

The purpose of this exercise is to explore how adders, registers and counters can be used. We will begin with the design of a ripple-carry adder (like that shown in class) and then use it in conjunction with registers to perform simple arithmetic computations. We will also explore counters as an optional (bonus) part of the exercise.

## 2. Preparation

To prepare for this laboratory you will need to create the circuits described below. Note that Part IV is optional, so you need not include it in the preparation.

Your preparation for this lab must consist of the following:

1. **PRINT** your Verilog code for parts I through III and place them in your lab book.
2. Place a **PRINTED and COMMENTED** simulation of each circuit in your lab book. Provide a sufficient number of test cases to show the correct operation of the circuit. Each input vector and the output associated with it should be accompanied by a comment explaining why the output is correct.
3. Provide an explanation of the operation of the circuit in Figure 2. The explanation must consist of the following items:
   (a) Assuming that all of the registers are initially cleared, and the number 3 (00000011) is placed on the input $I$, write down the values of the signals $A$, $B$, $C$, and $S$ for the first three clock cycles (you need to show the values of all four signals in each clock cycle).
   (b) Describe the function of the circuit in one sentence.
   (c) Assume that signal $A = 253\ (11111101)$ and $B = 1\ (00000001)$. Now assume that you set $I = 2\ (00000010)$. What will appear on wires $C$ and $S$ after 1 clock cycle? Assuming $I$ is kept the same, what will be the value on wires $C$ and $S$ after 2 clock cycles? You should be able to explain to the TA what you expect to observe.

## 3. In the lab

In the lab you will have to implement and test circuits described in the sections below. To simplify some of the steps a starter kit has been provided on the website:

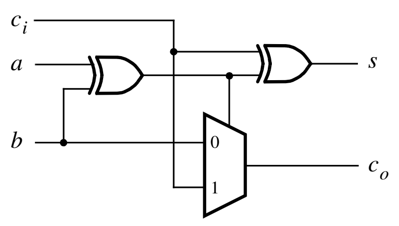
`http://www.eecg.utoronto.ca/˜jayar/ece241_08F/Lab5_starterkit.zip`

The starter kit is a ZIP archive containing a Quartus II projects for each part of the lab. Unzip the archive into a working directory called *lab5*.

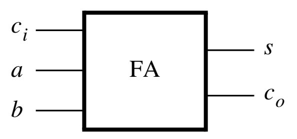
The in-lab component of your lab grade will be based on the circuit you create in **parts II and III** below. Part IV is optional and can be completed for a bonus mark. Please inform your TA when you have completed **all parts** and are ready to be graded.

**Part I**

Figure 1$a$ 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 $c_o s = a + b + c_i$ (Note: in this expression, the + symbol means *addition*, as opposed to logical OR). Figure 1$d$ 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.



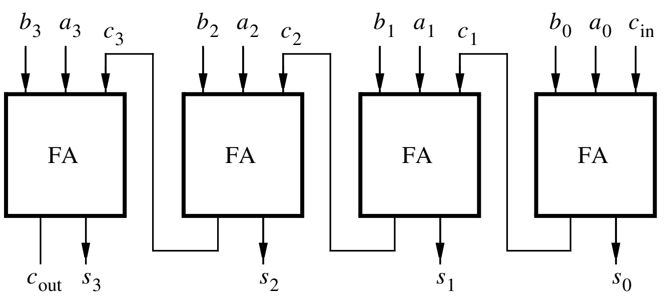
a) Full adder circuit



b) Full adder symbol

| $b$ | $a$ | $c_i$ | $c_o$ | $s$ |
|---|---|---|---|---|
| 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 |

c) Full adder truth table



d) Four-bit ripple-carry adder circuit

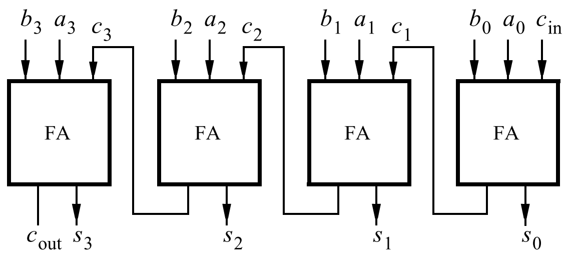
Figure 1. A ripple-carry adder circuit.

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

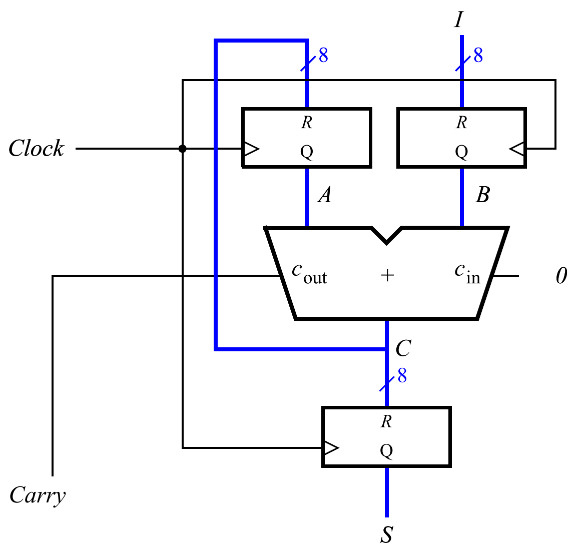
1. The project for this part is provided in the starter kit. Open the project named *part1* in the *part1* subdirectory to begin your work.
2. Write a Verilog module for the full adder subcircuit and write a top-level Verilog module that instantiates four instances of this full adder.
3. Use switches $SW_{7-4}$ and $SW_{3-0}$ to represent the inputs $A$ (which is made up of $a_3a_2a_1a_0$) and $B$ (which is made up of of $b_3b_2b_1b_0$) respectively. Use $SW_8$ for the carry-in $c_{in}$ of the adder. Connect the *SW* switches to their corresponding red lights LEDR, and connect the outputs of the adder, $c_{out}$ and $S$ (which comprises $s_3s_2s_1s_0$), to the green lights LEDG.
4. Simulate your circuit by trying different values for numbers $A$, $B$, and $c_{in}$.
5. If you have not already done so, include the necessary pin assignments for the DE2 board, compile the circuit, and download it into the FPGA chip.
6. Download the circuit onto the DE2 board and test its functionality.

**Part II**

Consider again the four-bit ripple-carry adder circuit from Part I; a diagram of this circuit is reproduced in Figure 2$a$. You are to create an 8-bit version of the adder with additional registers, and include it in the circuit shown in Figure 2$b$. Your circuit should be designed to support signed numbers in 2's-complement form. Note that the boxes labeled "R" in the figure are 8-bit registers - that is 8 D-type flip-flops with a common clock.



a) Four-bit ripple-carry adder circuit



b) Eight-bit registered adder circuit

Figure 2. An 8-bit signed adder with registered inputs and outputs.

Perform the steps shown below.

1. The project for this part is provided in the starter kit. Open the project named *part2* in the *part2* subdirectory to begin your work.
2. Write Verilog code that describes the circuit in Figure 2$b$. Use the circuit structure in Figure 2$a$ to describe your adder.
3. Include the required input and output ports in your project to implement the adder circuit on the DE2 board. Connect the input $B$ to switches $SW_{7-0}$. Use $KEY_0$ as an active-low asynchronous reset input for the registers, and use $KEY_1$ as a manual clock input. Display the sum outputs of the adder on the red $LEDR_{7-0}$ lights and display the carry output on the green $LEDG_8$ light. The hexadecimal values of $A$ and $B$ should be shown on the displays *HEX7-6* and *HEX5-4*, (using the hex decoder from Lab 3) and the hexadecimal value of $S$ should appear on *HEX1-0*.

4. Compile your code and use timing simulation to verify the correct operation of the circuit.

5. Open the Quartus II Compilation Report and examine the results reported by the Timing Analyzer. What is the maximum operating frequency, *fmax*, of your circuit? What is the longest path in the circuit in terms of delay?

6. Download the circuit onto the DE2 board and test its functionality by using different values of $B$. Use some test cases where the circuit produces arithmetic overflow and ensure that you understand why your circuit produces incorrect results for these cases.

**Part III**

Modify your circuit from Part II so that it can perform both addition and subtraction of eight-bit numbers. Use switch $SW_{16}$ to specify whether addition or subtraction should be performed. Connect the other switches, lights, and displays as described for Part II.

1. Simulate your adder/subtractor circuit to show that it functions properly, and then download it onto the DE2 board and test it by using different switch settings.

2. Open the Quartus II Compilation Report and examine the results reported by the Timing Analyzer. What is the *fmax* of your circuit? What is the longest path in the circuit in terms of delay?

**Part IV (Optional, for a bonus mark)**

Implement a 3-digit BCD counter. Display the contents of the counter on the 7-segment displays *HEX2−0*. Derive a control signal, from the 50-MHz clock signal provided on the Altera DE2 board, to increment the contents of the counter at one-second intervals. Use the pushbutton switch $KEY_0$ to reset the counter to 0. A diagram of the circuit is shown in Figure 3.

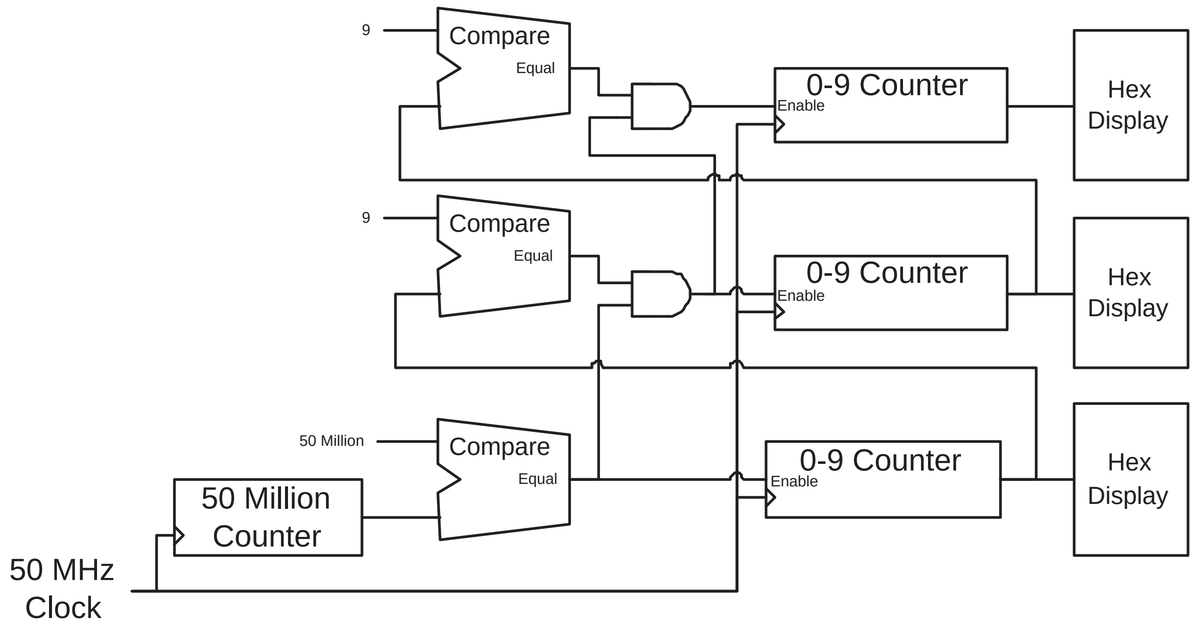


Figure 3. Block Diagram of a 3-digit BCD Counter.

**IMPORTANT:** In your design ensure that you do NOT *gate* the clock input. That is, the clock input to your circuit MUST be connected directly to the *clock* input of each flip-flop in your design. When a need arises to disable a flip-flop for a number of clock cycles, use an *enable* signal to keep the flip-flop's stored value from changing. An example of Verilog code for a flip-flop with an enable signal is shown in Figure A.33 on page 810 of the course text book.

Perform the following steps to complete this part:

1. The project for this part is provided in the starter kit. Open the project named *part4* in the *part4* subdirectory to begin your work.
2. Write a Verilog file that specifies the desired circuit.
3. Include the Verilog file in your project and compile the circuit.
4. Simulate the designed circuit to verify its functionality.
5. Assign the pins on the FPGA to connect to the 7-segment displays and the pushbutton switch.
6. Recompile the circuit and download it into the FPGA chip.
7. Verify that your circuit works correctly by observing the display.