

# ELEC 326: Class project

Kartik Mohanram

## 1 Introduction

For this project you will design and test a three-digit binary-coded-decimal (BCD) adder capable of adding positive and negative BCD numbers. In the process, you will

1. gain experience with modern CAD techniques used to design digital systems,
2. gain experience with designs that require ad-hoc design techniques,
3. learn to test projects through simulation,
4. learn to download projects to an FPGA board and perform physical testing, and
5. learn about BCD addition and subtraction.

### 1.1 Project Behavioral Specification

The BCD adder you design must be capable of adding two three-digit BCD operands to produce a three-digit sum. Each of the two operands can be either positive or negative, so the adder must, in effect, be capable of both addition and subtraction. The operands and sum are represented by 13-bit binary vectors where 12 of the bits are the three 4-bit BCD digits and the 13th bit is a sign bit (0 if the number is positive and 1 if it is negative). This is essentially a sign-plus-magnitude representation where the magnitude is represented with the BCD code. The adder should also have a one-bit overflow output signal that is 1 if the addition results in an overflow (a magnitude that is too big to be represented with 3 BCD digits) and 0 otherwise. BCD codes and arithmetic are discussed in Example 2.10 and Section 5.2 of the textbook. Recall, however, that Verilog allows you the great '+' operator that abstracts the details and makes design easy!

### 1.2 Project Organization

The adder is to be organized as three identical digit-adders connected in a ripple-carry configuration. Each of the digit-adders will be capable of adding two 4-bit BCD digits and a 1-bit carry/borrow input to produce a 4-bit BCD sum and 1-bit carry/borrow output. The adder should implement the following general algorithm:

1. The adder checks the signs and relative magnitude of the two operands.
2. If signs are the same, the adder adds the operands and makes the sign of the result equal the signs of the operands.
3. If the signs differ, then the adder compares the magnitudes of the operands, subtracts the smaller from the larger, and sets the sign of the result equal to the sign of the larger. Note that if the result is 0, the sign of the result must be positive (0 sign bit).

A high-level block diagram for the project is given in Figure 1. It shows the main data flow paths that connect the modules, but does not show the signals that connect to the control unit. That module will need inputs from the other modules and will generate control signals for all the modules. Part of the project assignment is to figure out what these control signals should be. The block diagram also shows how the switches and displays on the FPGA board are used. The remainder of this section describes the behavior of the modules in this diagram.

**DIGITADD:** This is a digit adder that takes two 4-bit BCD numbers and a carry/borrow input and produces a 4-bit BCD sum or difference and a carry/borrow output.

**SSCTLR:** This is the seven-segment display controller. It takes three 4-bit BCD numbers and displays them in the right three digits of the display. It also displays the sign in the leftmost digit position. The sign digit should turn on only the middle segment (minus sign) when the result is negative. When the result is positive, all segments on the fourth digit should be off.

**REGUNIT:** This module contains two 4-bit registers that hold the input operands for the DIGITADD module. There are not enough switches on the FPGA board to bring each input digit from a switch, so BCD input digits must be entered sequentially, one at a time, into the REGUNIT registers. This unit is also responsible for swapping the operands when necessary because the available adders can only subtract the B input from the A input, not the other way around. It is also convenient to add logic to the REGUNIT to determine which of the input operands is larger (needed to determine if the operands must be switched). The logic for this should be designed so that when several REGUNIT modules are connected in series, they will determine which of the 3-digit BCD numbers is larger (or whether they are equal).

**CTLUNIT:** This module takes information from the other modules and generates the control signals for those modules. It must check the signs and magnitudes of the operands and decide whether to do an addition or subtraction and whether to swap the operands. It is also responsible for computing the sign of the result.

**SWITCHES and DISPLAYS:** Operands are entered through five toggle switches (4 for the magnitude and 1 for the sign). The 4-digit, seven segment display shows both the sign and magnitude of the result. One LED is used to indicate an overflow. One toggle switch (ABSEL) indicates which of the two operands (A or B) is being loaded. Another toggle switch (SHOWREG) indicates whether the output display shows the value stored in one of the operand registers or the output of the adders. If a register operand is displayed, the ABSEL switch determines which one. One push-button switch clears all the registers to 0. Another push-button causes the next digit to be loaded in the registers.

## 2 Project Implementation

This section gives some more detailed information about how to implement the modules in the block diagram above. Each of the modules should be implemented as a Verilog module.

### 2.1 DIGITADD

Probably the best way to implement the digit adder is by first doing a binary addition/subtraction and then correcting the sum, if it is outside the range of valid BCD digits (i.e., greater than 9). The correction can be implemented by adding a correction value to the intermediate result produced by the first addition/subtraction. You can implement this with a 4-bit binary adder/subtractor to do the first addition/subtraction and another binary adder to perform the correction. You must determine when to do the

corrections and what correction values to add. You will need logic to generate the correction value.

## 2.2 REGUNIT

You should implement the registers in this module so that when the three REGUNIT modules are connected together, the three A operand registers form a 4-bit wide shift register and the three B operand registers form another shift register. These shift registers should shift from the least significant digit to the most significant digit. Connect four of the data entry toggle switches to the least significant position of both of these shift registers and use the ABSEL switch to determine which shift register is actually loaded (and shifted) when the LOAD pushbutton is pushed.

You can use multiplexers on the outputs of the registers to swap the two operands. It is also possible to implement the feature that switches the output display from showing the result of the addition to showing the values stored in the registers by observing that adding 0 to a number does not change it.

The logic that checks the relative magnitudes of the operands should take information from the comparison logic in the previous REGUNIT module and produce output signals that represent the result of comparing all digits through this position. These output signals are then passed to the next REGUNIT module. The output signals of the comparison logic in the last REGUNIT module indicate which of the full 3-digit operands is largest (or whether or not they are equal).

## 2.3 SSCTRL

Each digit on the seven-segment display is controlled by a signal connected to the anode of all the diodes in that digit. The signal must be low for any of the diodes to light. Individual diodes in a digit light when their anode is low and their cathode is also low. Note that this is different from what the Digilent manual for the Nexys-2 board says and what you would expect from the way diodes work. Indeed, a diode's anode must be high relative to its cathode in order for it to light. It appears that there is logic on the board that effectively inverts the anode signals.

The four digits share the eight cathode input signals. For example cathode input signal CA is connected to the cathode of segment A on all four digits. Therefore, to light a segment on a digit, you must set the cathode signal corresponding to that segment low and also set the anode for that digit low. To display a BCD value on a digit of the display, you must set all the cathode inputs low for the segments that are lit for that value when the anode input for that digit is low. The SSCTRL module must have logic that will cycle through the digits while making sure that the cathode inputs are in sync with the anode signals. For additional information of how these displays work, refer to the Digilent Nexys2 manual downloadable from their web site.

The leftmost digit must display the sign, not a BCD digit. To implement this, you will need to use one of the 4-bit patterns not used to represent BCD digits. That is, use patterns for binary numbers greater than 9 and less than 16.

It is a good idea to use the following submodule SSSCONV inside the SSCTRL module. You can then add the necessary logic to cycle through the digits to get the complete module.

### 2.3.1 SSSCONV

This is submodule of SSCTRL that does the conversion from a 4-bit input representing a BCD digit (or sign) to the seven outputs that control the cathodes of the seven segment display.

To complete the implementation of the SSCTLR module you need to add logic that takes three 4-bit BCD input digits and a sign bit and selects them one at a time in a fixed order. At the same time it must activate the digits on the seven segment display (set their anodes low) in such a way that they are in sync with the selection of input digits.

## 2.4 CTLUNIT

This module must contain two flip-flops to hold the signs of the two operands. One of these should be loaded with the value of the sign switch every time the load button is pushed. The ABSEL switch determines which one to load. The module must also contain logic to use the signs of the two operands and the result of the magnitude comparison of the operands to compute the sign of the result and to tell the DIGITADD modules whether to add or subtract.

## 3 Assignments

The project consists of three assignments, each with its own deadline and requirements. This section gives the details for the first of the three assignments. Detailed instructions for the next two stages will be emailed and posted on the class web-site as necessary.

### 3.1 Design of the SSCONV and DIGITADD modules

For this assignment you are to design and test the following modules:

1. SSCONV used to convert BCD numbers to the seven segment code. This module comes in handy as you debug all the stages of the project. It will eventually be integrated into the SSCTLR module.

Derive expressions for each of the cathode signals in terms of the four-bit BCD digits. Then write a behavioral Verilog module to realize these expressions. Verify the correctness of your module by simulation.

2. DIGITADD for BCD addition and subtraction. Develop an algorithm for BCD addition and subtraction and write a behavioral Verilog module to implement this algorithm. Test the algorithm for correctness by simulating the Verilog module. You should use high-level Verilog operations rather than binary logical operations or gate instantiations. For example, use the + operations for addition; don't do addition by simulating a hardware adder using the logical operations (AND, OR, NOT, XOR, etc.). Use always statements, conditional operations (IF, CASE etc.), and comments to make it easy for the reader to understand your algorithm.

**Demonstration:** Go to the lab and download your design to an FPGA board. In your configuration file, connect the outputs of your circuit to one of the digits of the seven segment display. Run through all the tests that the lab assistants provide – you are not considered done unless you have successfully passed all the tests.

**Inputs:** Here are the inputs that we will expect to see on all projects when we show up to grade them.

1. The 4-bit operands A and B will be mapped to the 8 on-board toggle-switches.

2. We will ignore the sign bits for A and B in this project. This is because the subtract/add control signal is determined in the control unit CTLUNIT in the final stage of the project by looking at the sign bits of the two full numbers. This helps abstract the determination of subtracting and adding out of the DIGITADD unit.
3. The carry/borrow (input) bit is mapped to push-button 1. In the final project, this bit is useless since there is no carry or borrow into the units place. It is useful here since it will demonstrate that your design, when used in the tens and hundreds places, will operate flawlessly.
4. An ADD/SUBTRACT control signal will be mapped to push-button 2. This is again useless as an external signal for later stages, but will help you (and us) test the unit to make sure that you can do both operations correctly.

### **Outputs:**

1. Overflow (carry-out/borrow) should be mapped to LED 1. Eventually, you will use the overflow bit in conjunction with the sign bit to display whether the result is positive or negative, or if overflow occurred, in the left-most seven-segment display available to you.  

You need not implement a “negative” sign in this stage of your project. This requires that you multiplex inputs to the 7-segment displays using a ring counter, something that we have not looked at in class and something that you’d rather not have to deal with right now. The borrow\_out signal is the equivalent of the negative sign in some sense, and that is all that we will look at in grading your project.
2. The results of addition/subtraction will be directly displayed on one of the four 7-segment display units using the SSCONV module.

**Grading:** When we show up to do the grading, here are some of the things to expect.

1. We will set the A and B switches with some values, and perform addition/subtraction with and w/o carry/borrow to see if the results and overflow are set correctly.
2. Along the way, we may just change A or B without changing the other and verify that you can handle this.
3. We will quiz all team members to see if each of you understands this stage of the project as a whole. We will also expect to see you answer general questions about the project. For example, we may want you to describe how the DIGITADDs can be integrated together. This will demonstrate that you have worked with an eye on the big picture and not in an assignment-driven manner.
4. Sample input-output combinations follow on the next page.

1. Addition (cin = carry\_in, cout = carry\_out)

a b cin sum cout

3 6 0 9 0

9 2 0 1 1

3 6 1 0 1

9 2 1 2 1

2. Subtraction (bin = borrow in, dif = difference, b\_out = borrow\_out)

a b bin dif b\_out

3 6 0 7 1

9 2 0 7 0

3 6 1 6 1

9 2 1 6 0

