Lab 7

Binary Coded Decimal

Updated: February 25, 2020
Version: SystemVerilog 2017

7.1 Introduction

In the last lab, we implemented a 7-segment display. We input numbers in binary, and it outputs in hex. When you get used to it, hex is fine to read, but it's not as easy as decimal, particularly for multi-digit numbers. In this lab, you will make a cheat button that will turn the display to decimal and then back to hex.

7.2 Objectives

After completing this lab, you should be able to:

- Use the double-dabble algorithm to convert hex values into BCD
- Explain how this algorithm can be implemented in hardware
- Use structural style to

7.3 Background

We are going to implement the binary-to-decimal conversion using binary coded decimal (BCD). This method represents each decimal digit with a fixed number of bits (4 in this case). For more, see Wikipedia: BCD.

BCD is not good for internal representation of numbers, as it is very inefficient in for storage and tends to make algorithms more complicated than necessary (compared to binary). However, it is a much better format for displaying a number for a human user. (It's often the case that what is easy for a human is hard for a computer, and vice versa.)

For simplicity, we will assume numbers are positive, or *unsigned decimal* (so 8 bits = 0 to 255, not -128 to 127). We will deal with the negative sign another day. We can use the **double-dabble** algorithm, also known as **shift and add 3**. For more, see Wikipedia: Double dabble.

Recall that 8 bits is a *byte*, and 4 bits is a *nibble*.

The basic algorithm is as follows:

1. Set up one nibble for each decade the answer might have, and append the binary number to the right of them (not in them).
2. Shift the number one bit to the left.
3. If you have shifted the number once for each bit in the starting number then you are done and the answer is in the decade registers.
4. If not then any decade that has a number greater than 4 (i.e. 5+) in it should have 3 added to that decade (there will never be a carry between digits at this stage. Why?)
5. Go to 2.

This is easiest to understand with a couple examples. See Tables 7.1, 7.2, and 7.3.

7.4 Pre-lab

Apply the double-dabble algorithm to the hex values 0B, 3E, A7 using the same format as Tables 7.1, 7.2, and 7.3. Write them on your own paper and submit as directed by your instructor.

7.5 Procedure

Now we need to implement this. The algorithm can be cleanly represented as a circuit, shown for various numbers of bits in Figures 7.1-7.3. Using these

Table 7.1: Convert 4-bit hex D to BCD

Tens	Ones	Binary	Comment
		1101	setup
	1	101	Shift 1
	11	01	Shift 2
	110	1	Shift 3
	1001	1	Add 3 to Ones
1	0011		Shift 4
1	3		Answer

Table 7.2: Convert 6-bit decimal 50 to BCD

Tens	Ones	Binary	Comment
		110010	setup
	1	10010	Shift 1
	11	0010	Shift 2
	110	010	Shift 3
	1001	010	Add 3 to Ones
1	0010	10	Shift 4
10	0101	0	Shift 5
10	1000	0	Add 3 to Ones
101	0000		Shift 6
5	0		Answer

Table 7.3: Convert 8-bit hex 92 to BCD

Hundreds	Tens	Ones	Binary	Comment
			10010010	setup
		1	0010010	Shift 1
		10	010010	Shift 2
		100	10010	Shift 3
		1001	0010	Shift 4
		1100	0010	Add 3 to Ones
	1	1000	010	Shift 5
	1	1011	010	Add 3 to Ones
	11	0110	10	Shift 6
	11	1001	10	Add 3 to Ones
	111	0011	0	Shift 7
	1010	0011	0	Add 3 to Tens
1	0100	0110		Shift 8
1	4	6		Answer

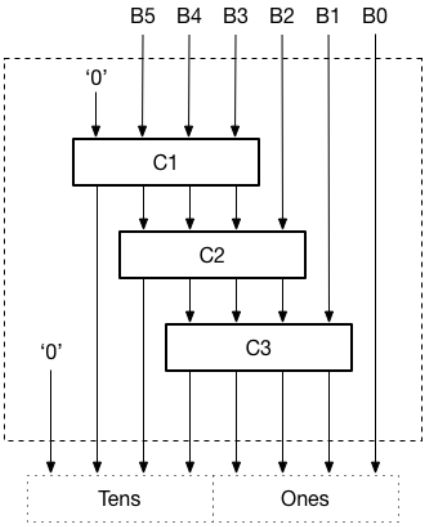

Figure 7.1: 6-bit double-dabble circuit

figures, we see we need to make two modules as described below.

7.5.1 Add 3

First, you will need an `add3` module, which adds 3 to the number if it is 5 or greater. As you can see from the diagrams/schematics, it needs a 4-bit input and output. Build this module, generate an ERT for it, and then test (simulate).

You can use a `for` loop to exhaustively test it without having to write lots of code (like the 7-seg decoder test bench from the previous lab). For your report, include a screenshot showing all 16 values and your ERT. (Make this two screenshots if needed for the values to be readable.)

7.5.2 BCD Modules

For the simulations of the next two modules, take a screenshot of the beginning and end of your waveform, and include ERTs of two or three values with each one. Make sure the values on the waveform are readable. Your test bench should be exhaustive, but you do *not* need to show every value in your report.

The second module is a 6-bit BCD converter module, which will use 3 instances of `add3` and a bunch of wires to implement Figure 7.1.

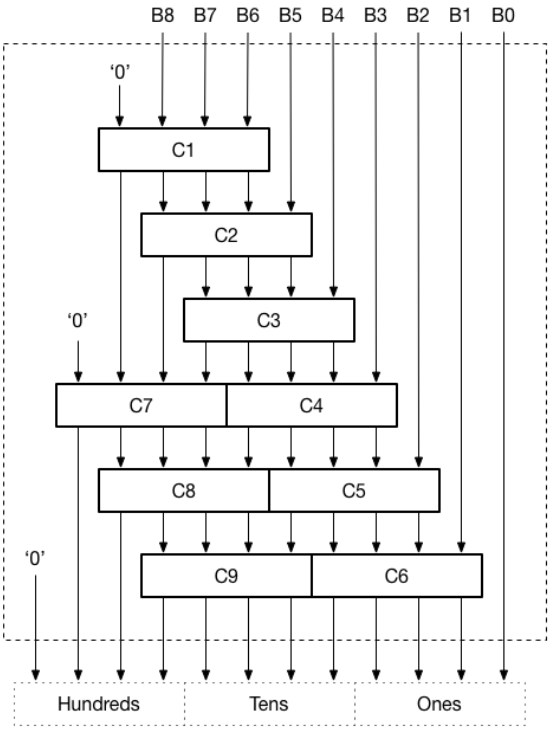

Figure 7.2: 9-bit double-dabble circuit

Third, draw an 11-bit BCD *circuit diagram* similar to Figure 7.2 or Figure 7.3. Then implement and test it.

7.5.3 Top-level Module

Finally, create a new top-level module named `sseg1_BCD` to implement this on your board. You can copy the code from your `sseg1` module, paste it into your new module, and modify from there. Don't forget to update the module name within the file!

Add your `bcd11` module to this file. Instead of routing the switches to the MUX, use the `ones` and `tens` outputs from the BCD module, and let the switches be the input to the BCD.

Optional: If you want the ability to switch back and forth between hex display and decimal (BCD) display, then add two more MUXs between the BCD module and the existing MUX. Use `sw[14]` as the select input for these.

Use the Add Sources dialog and check the box to copy your `mux2_4b` and `sseg_decoder` into your new

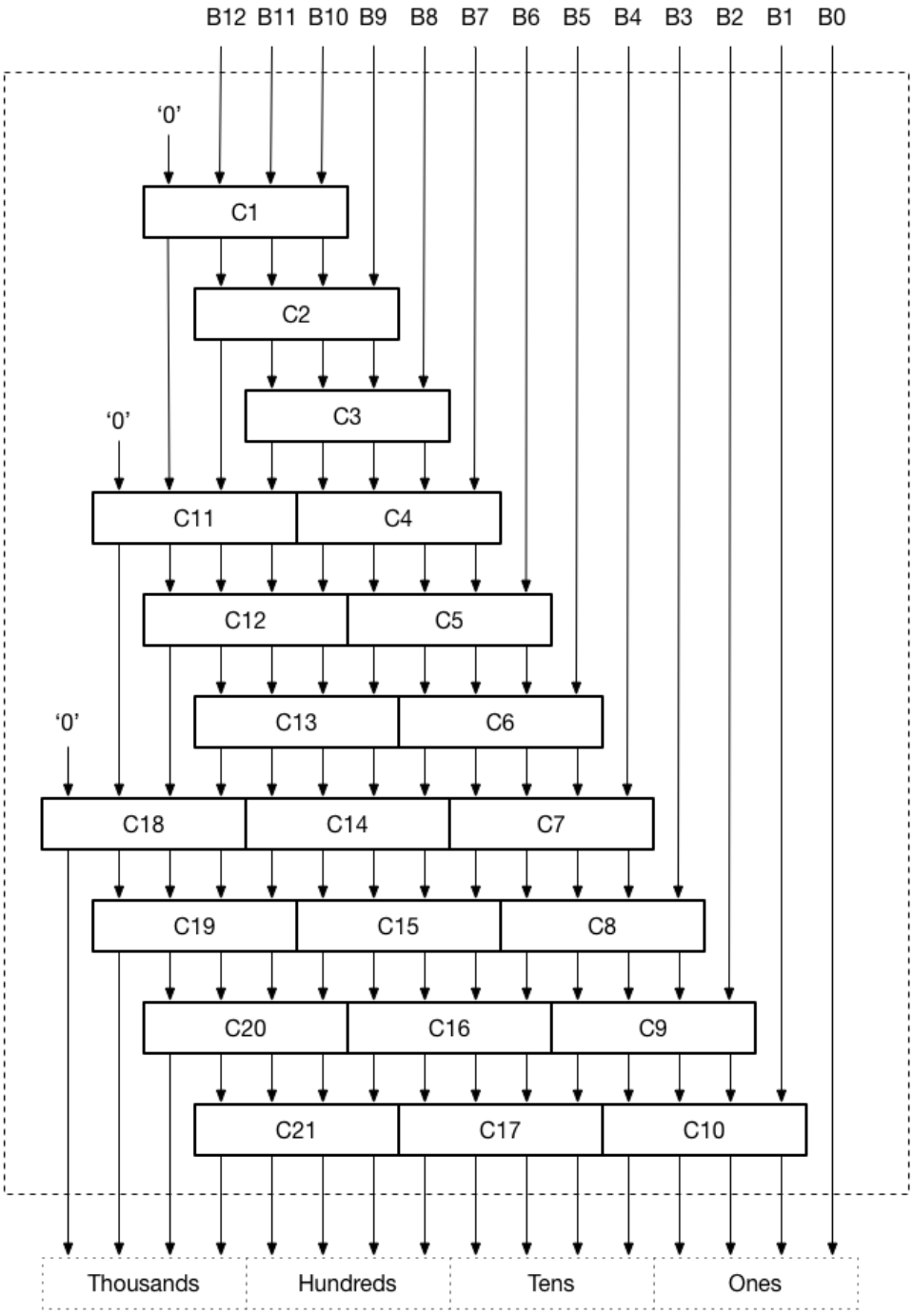

Figure 7.3: 13-bit double-dabble circuit

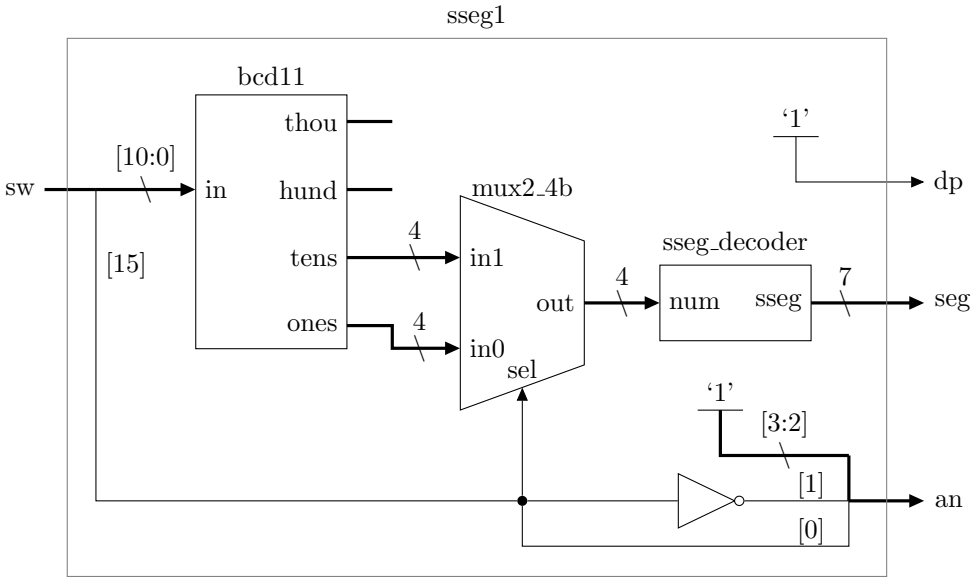

Figure 7.4: Block diagram for `sseg1_bcd`

project. Do the same with the necessary constraints files.

Program your board and check for correct output (decimal equivalent of switches 10:0). If you are not sure, ask your instructor. Set the output to decimal 98 and take a picture with each digit lit. Include all of the switches in your pictures, showing which ones are on and which are off.

7.6 Deliverables

Submit a report containing the following:

1. Four Verilog source files you wrote and three test benches
2. Three simulation waveforms with ERTs
3. Schematic/circuit diagram for the 11-bit converter
4. Two pictures of your board, one for each digit. Make sure all of the switches are visible as well.