

Lab 7: Karnaugh Maps and Subcircuits

CS/CoE 0447(B) Fall 2015 — Dr Kosiyaatrakul (Tan)

Released: Wednesday, October 28

Due: Monday, November 2, 11:59pm

Submission timestamps will be checked and enforced strictly by CourseWeb; **late submissions will not be accepted**. Remember that, per the course syllabus, if you are not marked by your recitation instructor as having attended a recitation, your score will be cut in half. **Follow all instructions.**

In this lab, you will develop a circuit that displays a short binary-coded decimal (BCD) number with seven-segment displays.

A. Computing minimized Boolean expressions

For this assignment, we will use the “7-Segment Display” component found under the “Input/Output” library. The seven-segment display unit takes eight inputs, each of which maps directly to a specific segment in the display, as shown in Figure 1. Take care **not** to use the similar-looking “Hex Digit Display”, which only takes two inputs.

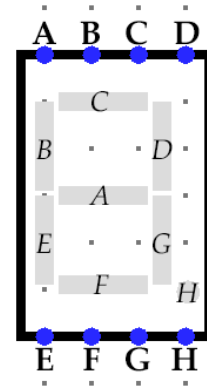


Figure 1. The segments of a seven-segment display.








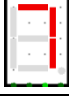
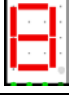


Observe, using the truth table provided in Figure 2 (on the next page), how a four-bit BCD input can be used to map to the eight output signals necessary to display a digit with the appropriate appearance. For example, the digit 6, encoded as 0110, should result in segments *A*, *B*, *E*, *F*, and *G* being on, while segments *C*, *D*, and *H* remain off.

Your task is to create a Karnaugh map (or “K-map”) for each of the outputs *A* through *H* **from the values presented in the truth table in Figure 2.** A K-map is an alternative representation of a truth table which allows us to minimize expressions in Boolean algebra, which will in turn allow us to simplify the circuits we build to represent those Boolean expressions.

To help you get started, let’s walk through building the K-map for segment *A*. We start by arranging our inputs — the four bits *b3*, *b2*, *b1*, and *b0* — along the sides of the grid, ordered in Gray code. We then fill in the desired output values for segment *A* in the appropriate squares of the K-map grid.

Next, we find groups of adjacent 1s in the K-map to obtain the minterms. Remember that valid minterm groupings must be rectangular, must have an area that is a power of two, and should be as large as possible without containing 0s. Keep in mind that groupings can wrap around any edges of the K-map grid. Multiple groupings may also overlap, if necessary, in order to make each one larger. **For segment *A*, the resulting K-map with one possible set of optimal groupings is shown in Figure 3.** Finally, we use these groupings to write a minimized expression for *A*: $\sim b_2 b_1 + b_1 \sim b_0 + b_2 \sim b_1 + b_3$.

- **Compute the Karnaugh maps and minimized expressions for segments *A* through *H*. If you write this out (neatly) by hand, scan it and convert to PDF as **lab07part1.pdf**. Alternatively, you may submit your K-maps and expressions as a Word document, **lab07part1.docx**.**

Input Digit					Seven-Segment Display								
Decimal	BCD Encoding				Appearance	Output Signals							
	b3	b2	b1	b0		A	B	C	D	E	F	G	H
0	0	0	0	0		0	1	1	1	1	1	1	0
1	0	0	0	1		0	0	0	1	0	0	1	0
2	0	0	1	0		1	0	1	1	1	1	0	0
3	0	0	1	1		1	0	1	1	0	1	1	0
4	0	1	0	0		1	1	0	1	0	0	1	0
5	0	1	0	1		1	1	1	0	0	1	1	0
6	0	1	1	0		1	1	0	0	1	1	1	0
7	0	1	1	1		0	0	1	1	0	0	1	0
8	1	0	0	0		1	1	1	1	1	1	1	0
9	1	0	0	1		1	1	1	1	0	1	1	0
Error	1	0	1	0		1	1	1	0	1	1	0	1
	1	0	1	1									
	1	1	0	0									
	1	1	0	1									
	1	1	1	0									
	1	1	1	1									

*Figure 2. Truth table for a seven-segment display. Invalid BCD input results in the error output, which activates segment **H**, the decimal point.*

b1b0					
b3b2		00	01	11	10
	00	0	0	1	1
	01	1	1	0	1
	11	1	1	1	1
	10	1	1	1	1

Figure 3. Karnaugh map for segment A of the seven-segment display described in Figure 2.

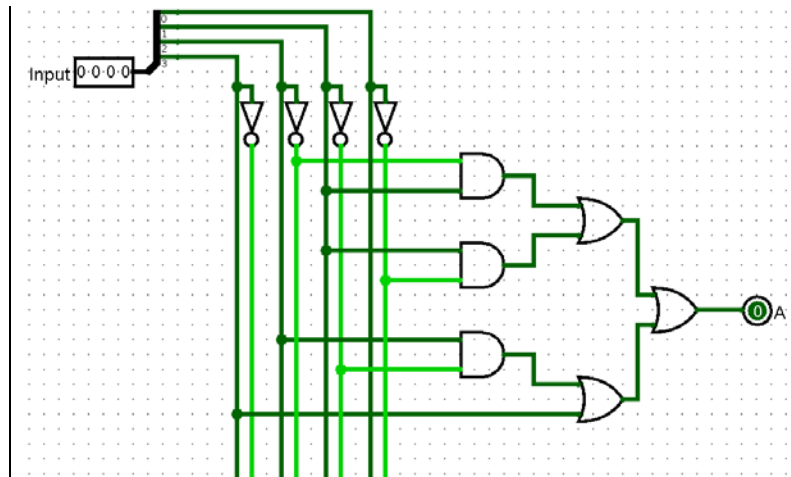


Figure 4. Circuit for segment A, based on the minimized formula $\sim b_2 b_1 + b_1 \sim b_0 + b_2 \sim b_1 + b_3$.

B. Building a two-digit BCD display

Now that we have the minimal expressions for outputs A through H, let's use them to build a subcircuit which takes a BCD-encoded digit as a four-bit input and outputs the eight one-bit signals necessary to produce the appropriate digit on a seven-segment display.

Click **"Project"** then **"Add Circuit"** to create a subcircuit. The benefit of using a subcircuit is that we can design it once, then instantiate multiple copies of it. Name this subcircuit **"BCD digit"**. We will design this circuit first, then instantiate it in our main circuit later.

Recall the minimized expression for segment A: $\sim b_2 b_1 + b_1 \sim b_0 + b_2 \sim b_1 + b_3$. From this, we can straightforwardly create the circuit shown in Figure 4. You may find the use of a **splitter** (found in the "Wiring" library) to be helpful in splitting the four-bit input into four individual one-bit signals which can be used to form the main lines through your circuit.

Now, while staying in the same **"BCD digit"** subcircuit, continue this process for segments B through H based on your work from Part 1. You can simply extend the main lines from your input signals downward, and branch off the necessary AND gates and OR gates. **Be sure to give labels to your outputs**, as these will be important once we instantiate the subcircuit.

Once your **"BCD digit"** subcircuit is complete, return to your main circuit by **double-clicking** **"main"** in the left-hand pane. To **instantiate** the **"BCD digit"** subcircuit here, **single-click** on the **"BCD digit"** entry, then click in the main circuit to place it somewhere. It will appear as a small rectangle. If you did everything correctly, you should see nine pins for this component: one for a four-bit input, and eight for the one-bit outputs.

Now, let's build a two-digit BCD display. Place two of your “BCD digit” subcircuits in the main circuit, alongside two seven-segment display units, and an eight-bit input pin. Use a splitter to split the eight-bit input into two four-bit signals — one for the “upper” digit and one for the “lower” digit. Send one of these signals to each of your “BCD digit” subcircuits, then connect the corresponding outputs to the seven-segment displays.

The end result should somewhat resemble Figure 5. Note that, depending on the layout of your “BCD digit” subcircuit, the pins may be laid out differently on the instantiated component, so your wires may connect in a different arrangement from what is shown in the figure. As always, we can mouse over these pins to see the labels we created earlier, to help us determine what's what and connect our wires accordingly.

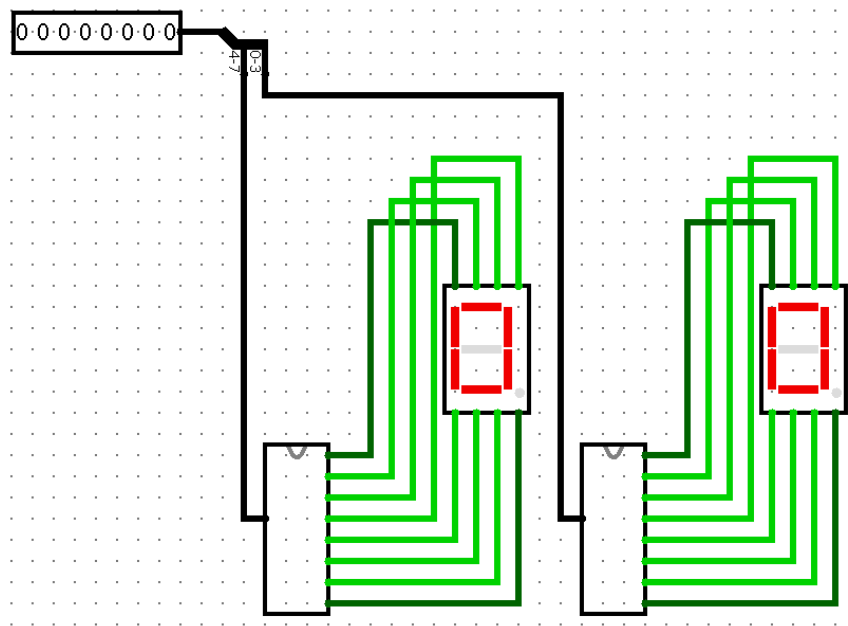


Figure 5. A sample two-digit BCD display, using subcircuits to decode a BCD digit into signals for a seven-segment display.

Lastly, **poke your input bits** to make sure that everything is working as expected. Try encoding various decimal numbers into BCD, such as 12, 47, and 86, and ensure that the appearance of each digit is as you expect. Try to make each BCD-encoded digit invalid and ensure that the error display works properly.

If you encounter any unexpected results which stem from the logic of your subcircuit, simply double-click on the subcircuit name in the left-hand pane, and modify the subcircuit accordingly. When you return to the main circuit, each instance will take on the updated behavior.

- ▶ Save your completed Logisim file as **lab07part2.circ**.
- ▶ Zip your two files for this assignment into a single zip file called **lab07.zip** and submit it via CourseWeb.