cs281: Introduction to Computer Systems

Lab 5: Combinational Logic with Binary Coded Decimal

1 Binary-Coded Decimal

Binary-Coded Decimal (BCD) is the encoding of the 10 decimal digits (0 through 9) in discrete sets of four bits per digit. When we use a set of four bits, we have 16 possible bit patterns, and in BCD, we only need to encode ten digits, thus some of the bit patterns are simply designated as *invalid*. The table below shows this mapping between the binary encoding and the encoded decimal digit:

BCD Bit Encoding	*Decimal Digit Encoded*
`0000`	0
`0001`	1
`0010`	2
`0011`	3
`0100`	4
`0101`	5
`0110`	6
`0111`	7
`1000`	8
`1001`	9
`1010`	*invalid*
`1011`	*invalid*
`1100`	*invalid*
`1101`	*invalid*
`1110`	*invalid*
`1111`	*invalid*

We can group digits together to represent decimal values, so if I want to use BCD to represent 432_{10}, I would use the bit pattern:

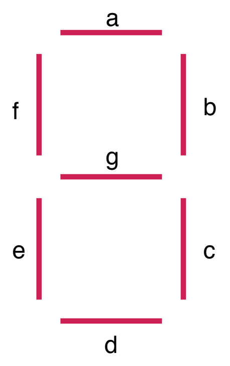
```
0100 0011 0010
```

Note that this encoding does **not** have the property that we can use binary addition on two BCD encoded operands and have the resultant bit pattern be the correct BCD encoding of the sum.

2 BCD to Seven Segment Code Converter

Recall that a combinational circuit may have more than one output. You have already seen this in the c_{out} and s outputs of your half adder and your full adder. In such a situation, *each* output must be expressed as a function of the inputs. A digital circuit called a *code converter* is an example of a multiple-output circuit. A code converter transforms information from one binary code to another.

In this lab, you will design and implement a code converter circuit whose input is the 4-bit code for a BCD bit encoding and the output is a set of seven binary values that correspond to which lighted segments are on and off in a seven-segment display used to display a decimal digit. Note that you have seen two such seven segment displays on your breadboards, and the breadboard internally has logic for the BCD to Seven-Segment code converter taking your four bits of input and lighting the correct segments for the digit. So, for each valid BCD pattern, we can enumerate the segments that should be lit that correspond to that decimal digit. In the figure below, we have given letter names of a-g for the seven segments.



So the table below lists the outputs that should be 1 for each decimal digit, under the assumption that all other segments should be 0, indicating a segment that is *not* lit.

Digit	*Segments*
0	a, b, c, d, e, f
1	b, c
2	a, b, g, e, d
3	a, b, g, c, d
4	f, g, b, c
5	a, f, g, c, d
6	a, f, g, c, d, e
7	a, b, c
8	a, b, c, d, e, f, g
9	a, f, g, b, c

So we are designing a circuit with 4 inputs, the W, X, Y, Z bits of the BCD encoding, and 7 outputs, the appropriate 0/1 values for segments a, b, c, d, e, f, g depending on the inputs and the associated decimal digit. We proceed as we have in the past, as follows:

1. Build a full truth table that has 16 rows for the combinations of the four inputs (W, Y, Y, Z). For the combinations that correspond to an invalid BCD encoding, place an X, indicating "don't care" in the output column for each of the seven outputs.

2. Construct 7 K-maps, one for each output. Fill in the K-map as before, and include the X in the correct table entries for the don't-cares.

3. Circle a minimal set of prime implicants. Use the X's, which by definition could be *either* 0 or 1, since we don't care, to help build the biggest possible prime implicants, which helps result in smaller boolean expressions.

4. Write down the corresponding boolean algebra expression for each output.

5. Realize the set of circuits using Logisim.

2.1 Logisim Realization

We are going to use this opportunity for you to learn how to work with a subordinate circuit in Logisim. I have created a template circuit file for you, available on the course web site and named `lab5.circ`. Download that file and open it in Logisim.

In the left pane, you will see two circuits, the one named 'main', which is what you have used for previous work in Logisim and should have a magnifying-glass icon layered on top, and a circuit named bcdcoder. If the bcdcoder circuit has the magnifying glass, double click the main circuit.

In the canvas, you should see, from left to right, four input pins corresponding to the BCD encoding: W, Y, Y, Z, a rectangular circuit labelled 'BCD-7', 7 input pins, and a set of LEDs arrayed to simulate a seven-segment display, with groups of 3 wired together and wire-routed to the a through g input pins.

Your job will be to fill in the bcdcoder circuit, with its 4 inputs and 7 outputs corresponding to your design from above, and then to remove the seven segment input pins on the main circuit and connect the wires together so that the digit segments will be driven by your circuit implementation.

Double click on the bcdcoder circuit and you will see the inputs and outputs. As you lay out your circuit, you can change the spacing, but you *must* retain the four inputs, in their current order, on the top of the bcdcoder circuit, and the seven outputs, in their current order, on the right side of the bcdcoder circuit.

You may play with the main circuit and try using different 0/1 values for a through g to get a feel for what these outputs are supposed to do. Then go ahead and implement your bcdcoder circuit. Avoid repeating use of gates and keep things as organized and simple as possible.