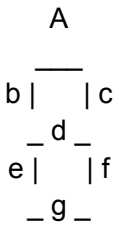
7 - Segment LED. I'll label each segment

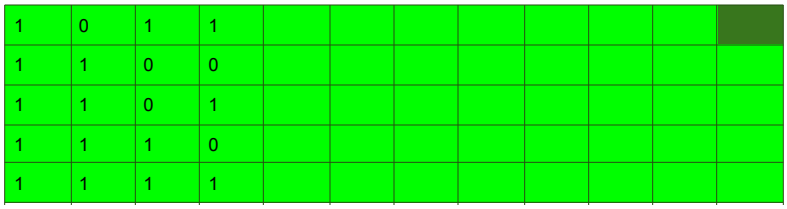
| | | |
|---|---|---|
| | Top (a) | |
| b | | c |
| | Middle (d) | |
| e | | f |
| | g | |

The follow table shows which segments light up for which number.




Truth table for the 7-segment decoder

| w | x | y | z | | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | | 1 | 1 | 0 | 1 | 1 | 0 | 1 |




Kmap - A segment

| wx\yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | | 1 | 1 |
| 01 | | 1 | 1 | |
| 11 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1 |

A = w + x'z' + xz + yz

Kmap - B segment

| wx\yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 0 | 0 |
| 01 | 1 | 1 | 0 | 1 |
| 11 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1 |

B = w + y'z'+xy'+xz'

Kmap - C segment

| wx\yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 1 | 1 | 1 |
| 01 | 1 | | 1 | |
| 11 | | | | |

| | | | | |
|---|---|---|---|---|
| 10 | 1 | 1 | | |

C = w’x’ + x’y’ + w’y’z’ + w’yz

Kmap - D segment

| wx\yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | 1 | 1 |
| 01 | 1 | 1 | | 1 |
| 11 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1 |

D = w + xy’ + xz’ + x’y

Kmap - E segment

| wx\yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | | | 1 |
| 01 | | | | 1 |
| 11 | 1 | 1 | 1 | 1 |
| 10 | 1 | | 1 | 1 |

E = x’z’ + wx + yz’ + wy

Kmap - f segment

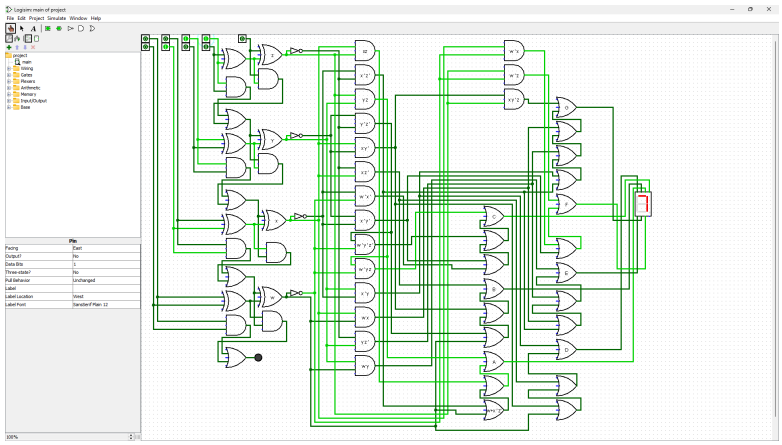
| wx\yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 1 | 1 | |
| 01 | 1 | 1 | 1 | 1 |
| 11 | | | | |
| 10 | 1 | 1 | | |

F = x’y’ + w’x + w’z

Kmap - g segment

| wx\yz | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | | | 1 |
| 01 | | 1 | | 1 |
| 11 | 1 | 1 | 1 | 1 |
| 10 | 1 | | 1 | 1 |

G = x'z' + x'y + yz' + wx + xy'z



**Penny for My Thoughts:**

The most difficult part of the project was verifying whether my strategy was correct. I was initially confused about why half-adders and full-adders existed. I mistakenly believed they were redundant since you can easily add bit strings in a programming language. What I didn't realize

is that, for any computer to perform addition, it must first have the physical ability to add numbers. The state of a pin represents a 0 or 1, and a full-adder allows a computer to physically add two numbers. The Logisim diagram is merely a representation of what the physical circuits would look like. Once I realized this, the first part of the problem became trivial.

The next part involved converting a bit string into, essentially, a drawing that represents the decimal number of that bit string. Initially, when we first covered the sum of products, I didn't understand its purpose. I knew how to do it, but its use cases eluded me. I remember a question in class that asked something to the effect of "represent abc in its sum of products," but I didn't realize that we had to represent each individual letter using sum of products—first A, then B, then C. When I started thinking about how I could draw a number, given a bit string, I immediately remembered that I could break down the problem by framing it as "how can I draw one segment of the number at a time?" Specifically, I needed to draw each segment only for certain bit strings. For whatever reason, I thought back to this lesson, and it became immediately obvious what the sum of products allows you to do: it lets you turn a variable on or off as a function of its inputs.

The next part involved actually building the circuit in Logisim. First, I had to build the truth table for each segment, which involved drawing each number on paper. I named each segment and, for each number, I made a list of which segments were on. Once I had this list, building the truth table became trivial.

Using K-maps to reduce the complexity of the sum of products was also straightforward, thanks to the multiple examples we covered in class. I tried further simplifying the results from the K-map but wasn't sure if they could be simplified further. Eventually, I asked ChatGPT to verify whether my result was the simplest form, but ChatGPT wasn't super helpful for this. I started Googling how to verify if I had done the K-maps correctly and found an online calculator that does both simplifications and truth table verification. This tool also generated truth tables for the results, so I could confirm they matched the initial truth table.

Finally, it was time to build the circuits. I initially tried grouping each segment into one part of the Logisim diagram but quickly realized that some computations occurred multiple times. For example, multiplying two bits (xz) might appear in the expressions for multiple segments. So, I decided to scrap that idea and instead first build all the AND circuits to minimize redundant work. Once I had the AND circuits, I started adding them together to build each segment.

The last OR circuit was the culmination of my work and the point where I tested whether all the prior steps were correct. I immediately tested whether different combinations of bit strings resulted in the expected segments turning on or off. Along the way, I ran into multiple segments that weren't working as expected. I had to track down the errors: did I build the circuits incorrectly? Did I use the K-maps wrong? At one point, I accidentally wrote a 0 instead of a 1 for a cell in the truth table. That's when I realized I should probably color-code the truth table to make typos easier to spot visually. Once I finished building the OR gates for each segment, I tested whether the circuit worked properly for numbers between 0 and 15.

At one point, while deciding how to build the circuits for each segment, I thought to myself: since the segments look the same for every number between 10 and 15, I could just check whether the bit string represented numbers greater than or equal to 10. I even found a guide on implementing comparison operators. However, I quickly realized how much extra work that would require and decided to scrap the idea.