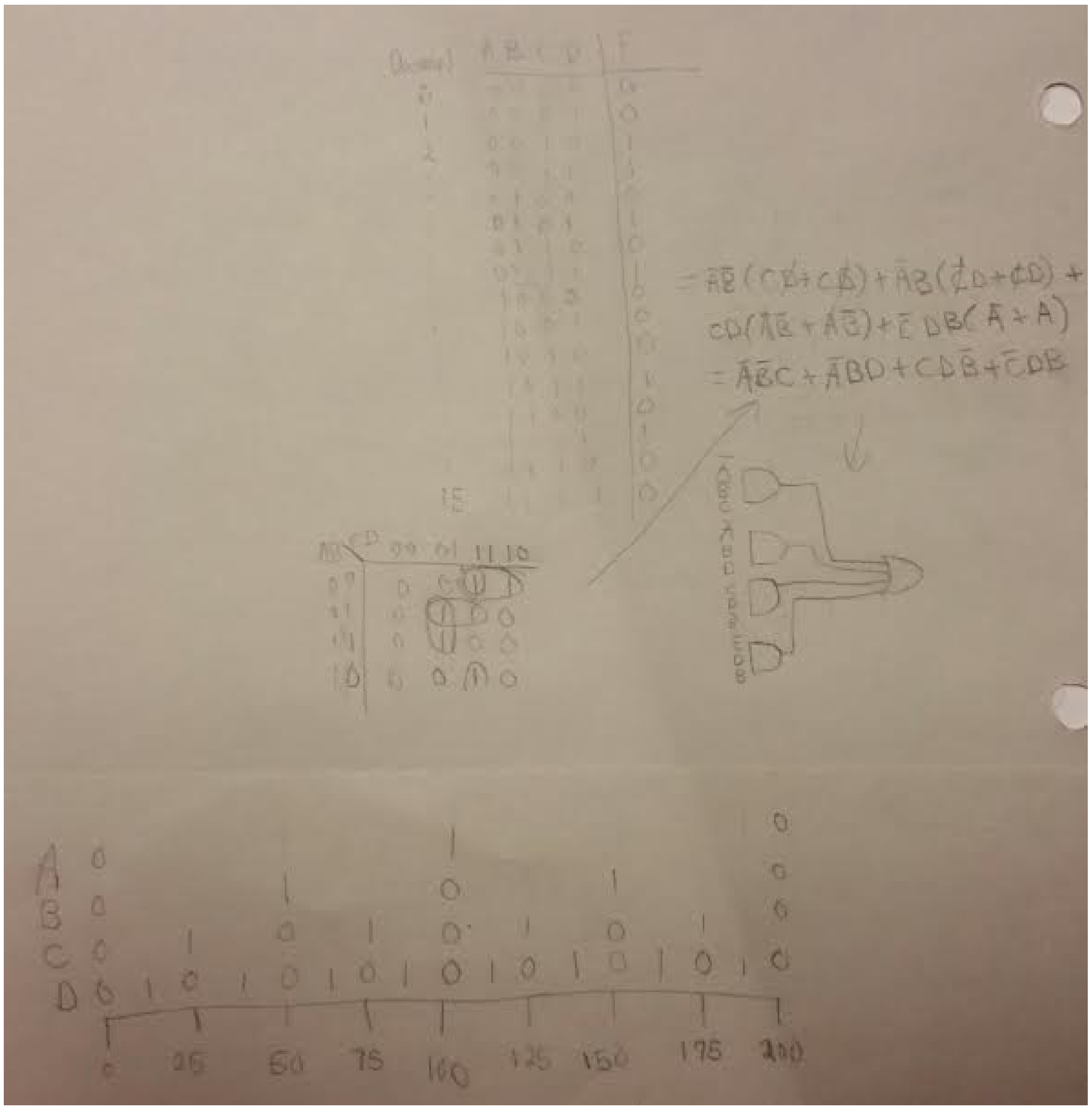
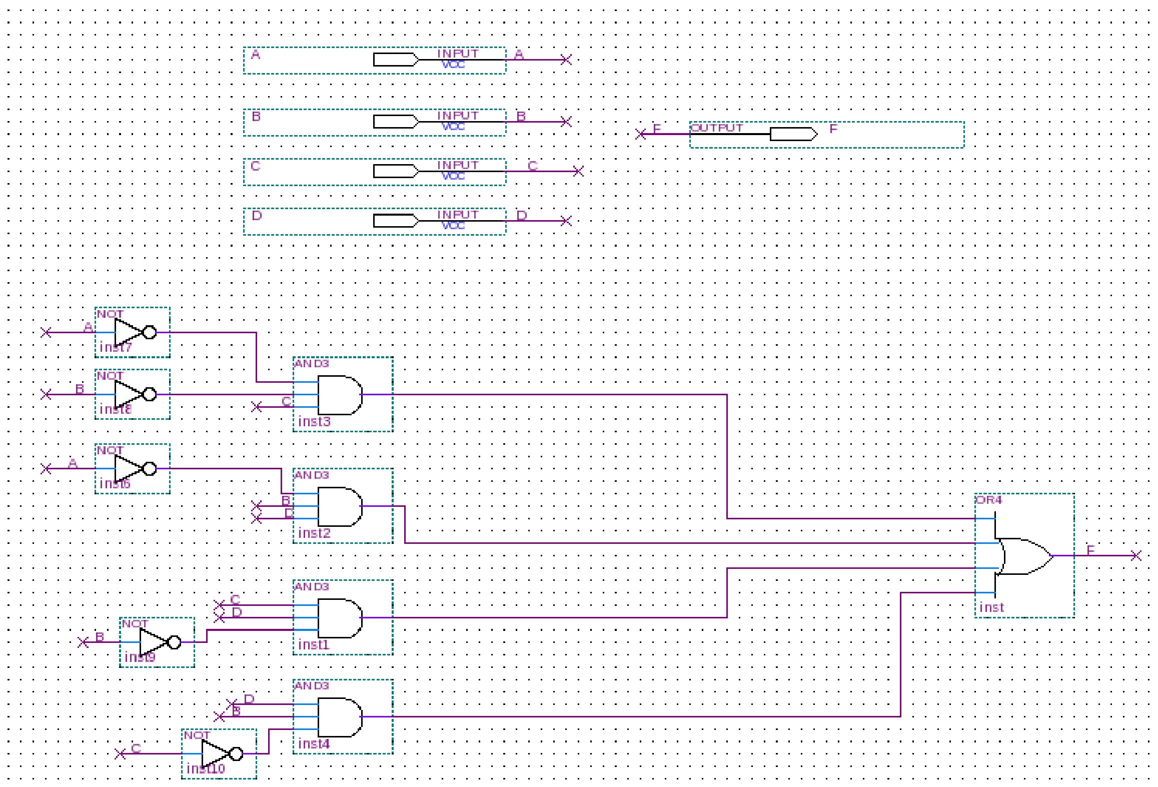
Zena's CS232 Project 1

For this first project, the task was to design a prime number circuit that will take 4-bit input (meaning the numbers could range from 0 to 15) and if the number is prime, the value of the output will be true/1. To create this, the first thing I did was make a truth table that had 16 inputs (0000-1111) and 16 outputs (either 1 or 0). After I found which numbers were prime, I made a Karnaugh map and created a boolean expression from it. Next, I drew the circuit that the simplified expression. The picture of my scrap work is shown below:



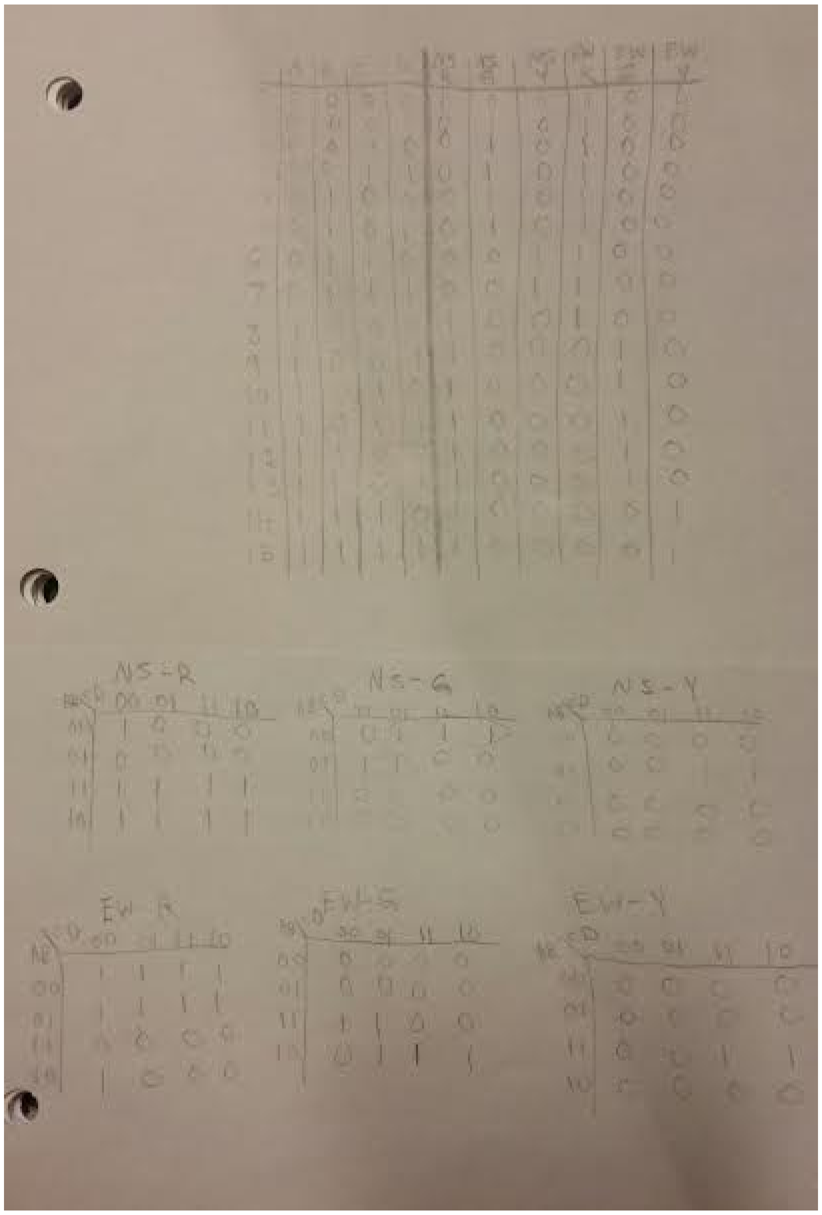
I modeled the circuit in Quartus, which resulted in the picture below:



There are four inputs, A-D, for each bit of the four. The output, F, takes the four inputs, and being an OR gate, if any of them are true, it will make the result 1/true. To simulate it, I made a driver file that changes the value of on of the inputs at certain increments. The plan for this is shown in the calculation picture above, where one can see that A, for example, changes to 1 halfway through, representing how the number starts with 0 half the time and 1 the other half. On a similar vein, the value of B is 1 half the time A is one, and 0 the other. This pattern continues until one can see how the timing of the changes should be done in order to make all of the combinations go through the simulation. A picture of the simulation is below:

One can see from the picture that the circuit works, because where the last row's lines (the output's lines) are high, there is a true/1 value. The places they are high correspond to 2, 3, 5, 7, 11, and 13, the prime numbers in the range 0-15.

The next task was to make a stoplight circuit, which would "control the lights" of a four-way traffic light. This meant that there were six outputs: north and south green, west and east green, north and south red, west and east red, north and south yellow, and west and east yellow. I first planned this out using 4-bit input which resulted in either true or false values for each of the 6 possible outputs. The task had a table provided that established the rule for telling when an output was 1 or 0. I made Karnaugh maps from the truth table, then wrote out and simplified the Boolean expressions from each. My work is shown below:



	A	B	C	D	NS R	NS G	NS Y	EW R	EW G	EW Y
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1	0	0	1	0	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	1	0	1	0	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	1	0	1	0	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	1	0	1	0	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	1	0	1	0	0
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	1	0	1	0	0
6	[illegible]	[illegible]	[illegible]	[illegible]	0	0	1	1	0	0
7	[illegible]	[illegible]	[illegible]	[illegible]	0	0	1	1	0	0
8	[illegible]	[illegible]	[illegible]	[illegible]	1	0	0	[illegible]	0	0
9	[illegible]	[illegible]	[illegible]	[illegible]	1	0	0	0	1	0
10	[illegible]	[illegible]	[illegible]	[illegible]	1	0	0	0	1	0
11	[illegible]	[illegible]	[illegible]	[illegible]	1	0	0	0	1	0
12	[illegible]	[illegible]	[illegible]	[illegible]	1	0	0	0	1	0
13	[illegible]	[illegible]	[illegible]	[illegible]	1	0	0	0	1	0
14	[illegible]	[illegible]	[illegible]	[illegible]	1	0	0	0	0	1
15	[illegible]	[illegible]	[illegible]	[illegible]	1	0	0	0	0	1

NS-R

AB\CD	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

NS-G

AB\CD	00	01	11	10
00	0	1	1	1
01	1	1	0	0
11	0	0	0	0
10	0	0	0	0

NS-Y

AB\CD	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	0	0	0
10	0	0	0	0

EW-R

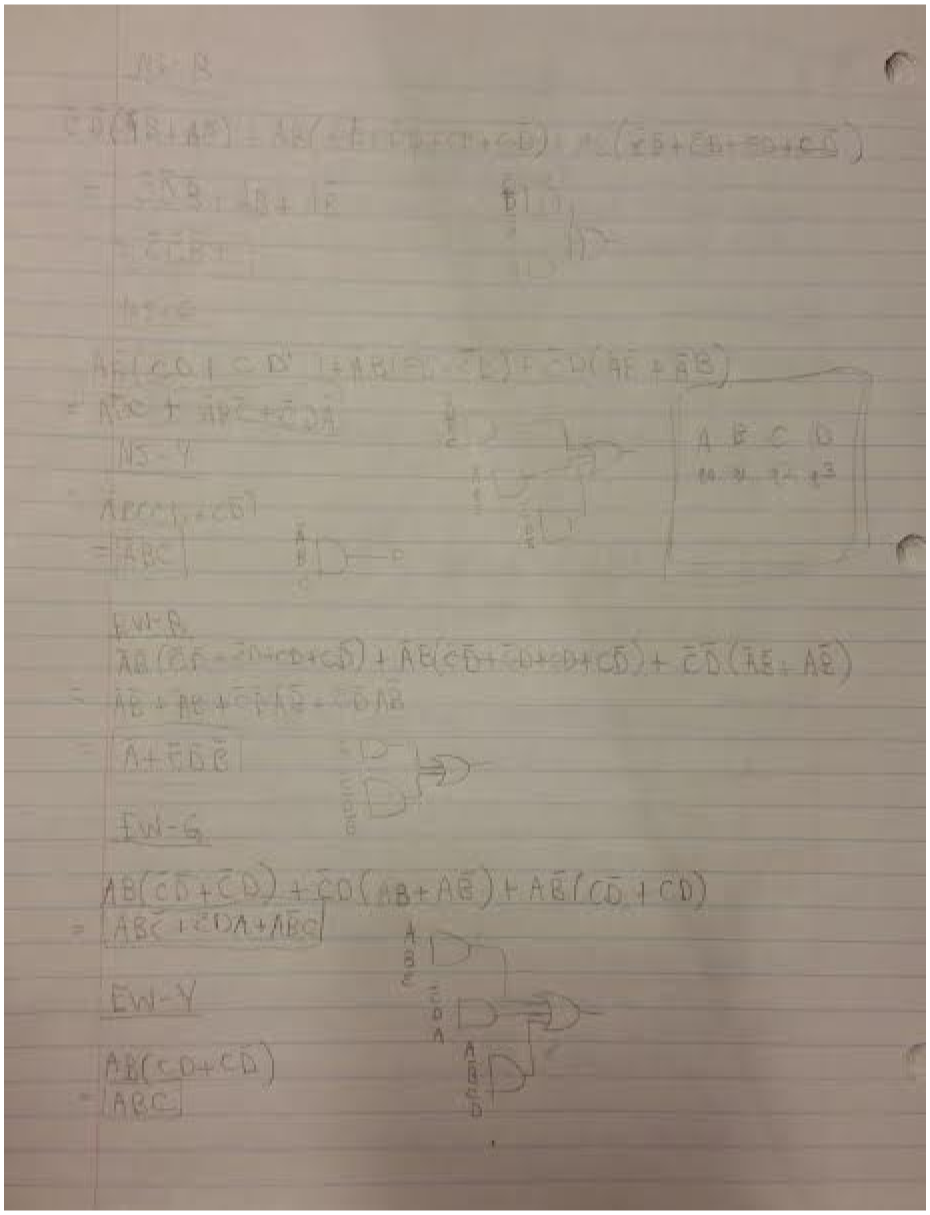
AB\CD	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	0	0	0	0
10	1	0	0	0

EW-G

AB\CD	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	0	0
10	0	1	1	1

EW-Y

AB\CD	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	0	1	1
10	0	0	0	0



NS-R

$\bar{C}\bar{D}(\bar{A}B+A\bar{B}) + AB(\bar{C}\bar{D}+\bar{C}D+CD+C\bar{D}) + AB([illegible]B+\bar{C}B+\bar{C}D+C\bar{D})$

$= \bar{C}\bar{D}B + AB + A\bar{B}$

$= \bar{C}\bar{D}B + [illegible]$

NS-G

$A\bar{B}(\bar{C}D + C\bar{D}) + \bar{A}B(\bar{C}D+\bar{C}\bar{D}) + \bar{C}D(A\bar{B}+\bar{A}B)$

$= A\bar{B}C + \bar{A}B\bar{C} + \bar{C}D\bar{A}$

NS-Y

$\bar{A}BC(\bar{C}D + C\bar{D})$

$= \boxed{\bar{A}BC}$

A B C D

14 11 [illegible] [illegible]

EW-R

$\bar{A}B(\bar{C}\bar{D}+\bar{C}D+CD+C\bar{D}) + \bar{A}B(C\bar{D}+\bar{C}D+CD+C\bar{D}) + \bar{C}\bar{D}(\bar{A}B+A\bar{B})$

$= \bar{A}B + AB + \bar{C}\bar{D}A\bar{B} + \bar{C}\bar{D}\bar{A}B$

$= \boxed{A + \bar{C}\bar{D}\bar{B}}$

EW-G

$AB(C\bar{D}+\bar{C}D) + \bar{C}D(\bar{A}B+A\bar{B}) + A\bar{B}(\bar{C}D+CD)$

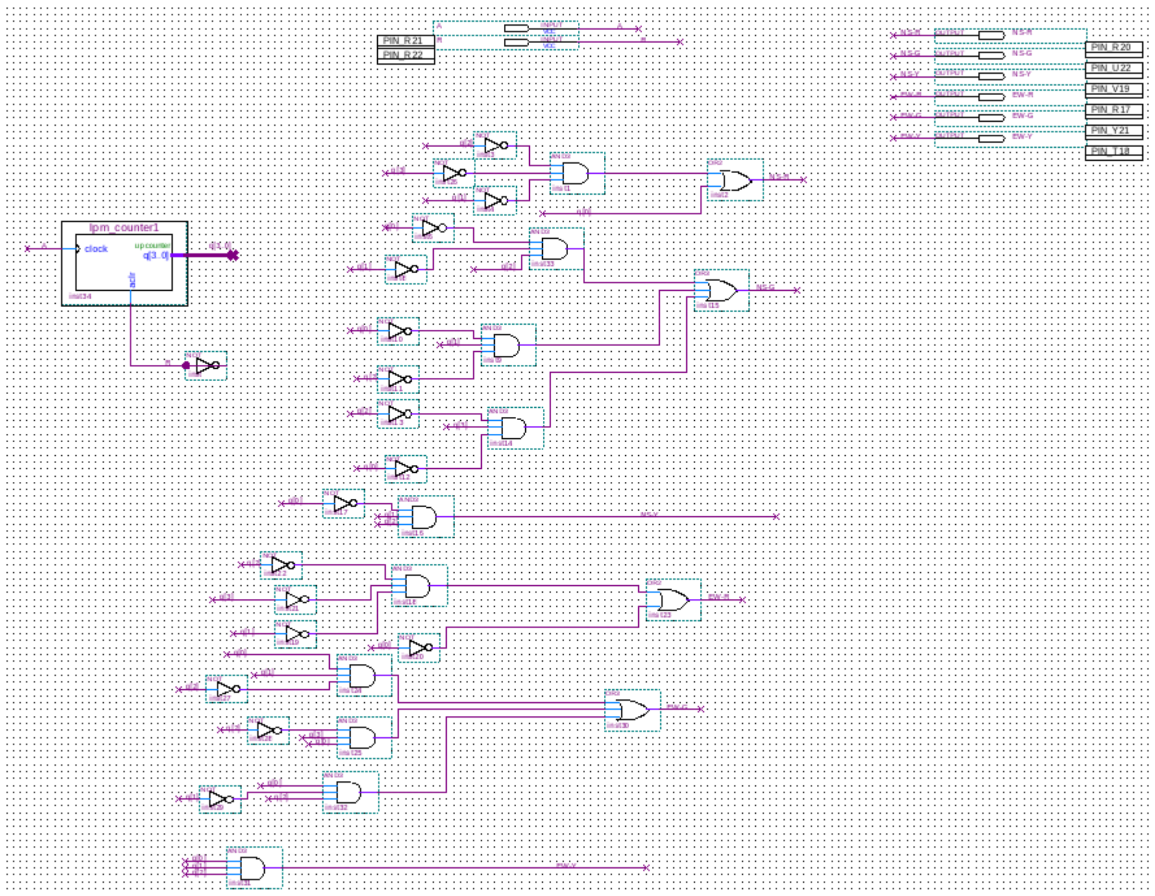
$= \boxed{AB\bar{C} + \bar{C}DA + A\bar{B}C}$

EW-Y

$AB(\bar{C}D+C\bar{D})$

$= \boxed{ABC}$

After I had this to work with, I modeled it in Quartus. The image of this is below:



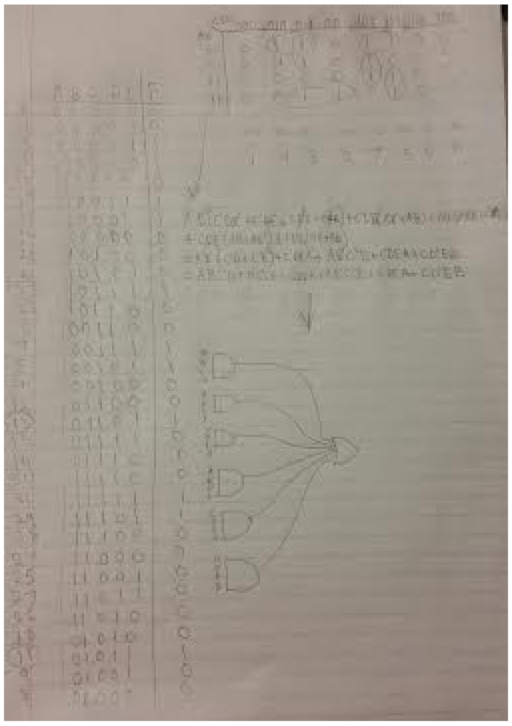
Since the task said to use a counter, which cycles through the numbers, I only had one input, which corresponded with the one button I used to increase the input by one on the breadboard, which I used to test. There are six results, as one can see, and the output of the machine has 4 values, corresponding to the number of bits. If one of the gates gets all of the machine outputs into it that it needs for it to be true, it will move on into the or gate, rendering the output, the light display, true. I tested my design using the machine, and a video of me cycling through all 16 inputs (then doing two more from the start of the cycle) can be seen below (sorry, it's upside-down):

20160215_173343.mp4

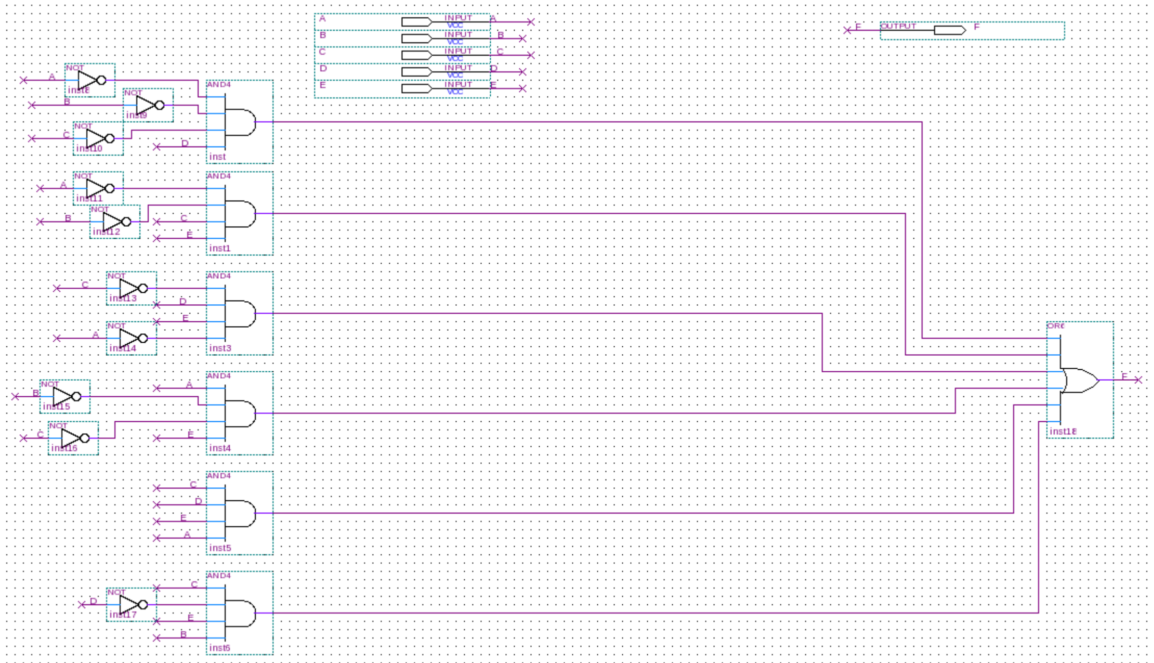
The two lights, when green, represent green for each cardinal direction combination. The red lights represent red for each when farther away, and yellow for each when closer to each other (and towards the middle).

Extension 1 was undertaken in that I showed my work to get the simplified, minimal logic circuit.

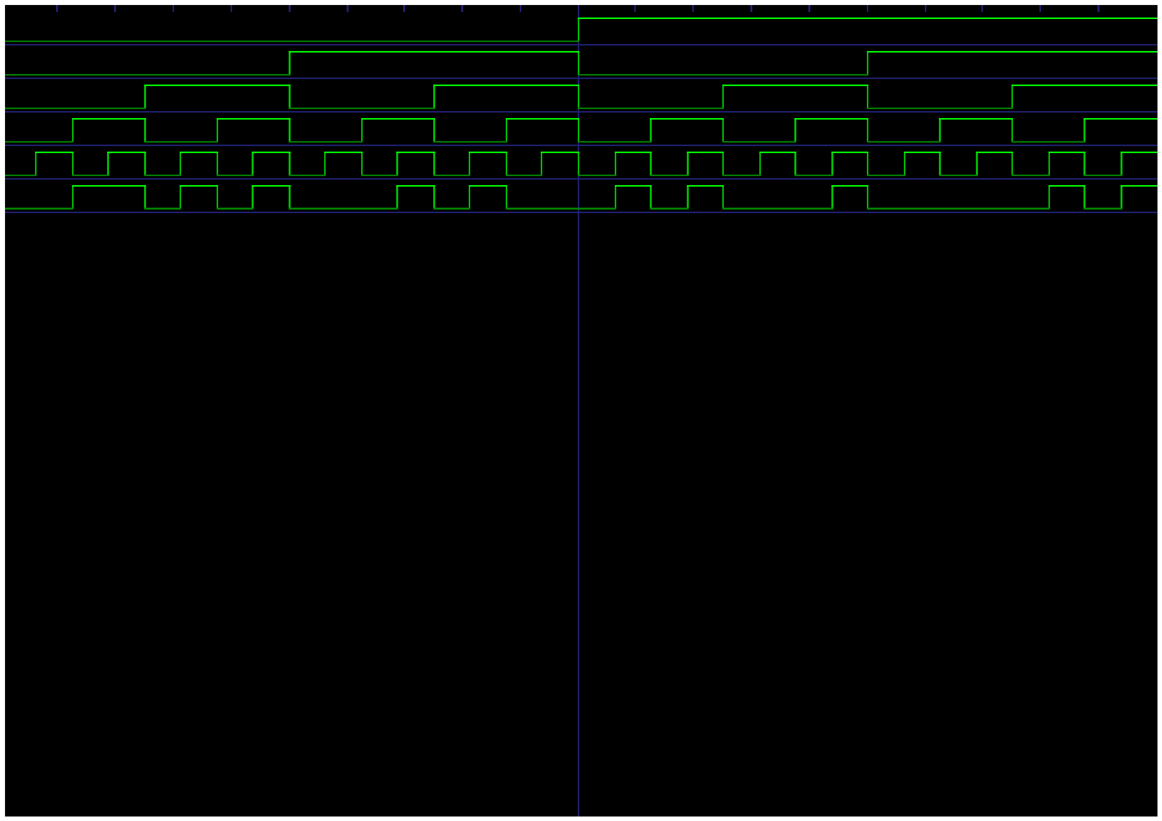
I also undertook extension 4, in which I changed my prime number circuit so it had 5 digits, meaning there were 32 numbers (0-31) to get an output from. Instead of A, B, C, and D, I now had to make a truth table and Boolean expressions with A, B, C, D, and E. The work to get the expression for the new circuit is shown below:



After that, I built the circuit inside Quartus. The image is shown below. It follows the same logic as the original, but with more inputs, gates, etc.



After that, I figured out how to increment E, the additional digit, for the timing. Every 6.25 seconds, it would change value from 0 to 1 and vice versa. In the simulation that I made after, one can see that in 11 spots (corresponding to the 11 prime numbers in the range 0-31), F (the output) is equal to 1, meaning that F is true and the number is prime. The picture of this is shown here:



This project taught me the process of planning a circuit using Boolean logic and Karnaugh maps, as well as how to use Quartus to make and simulate circuits both virtually and on breadboards. It also taught me how to interpret simulation results and how to check if my logic translated well from paper to computer.