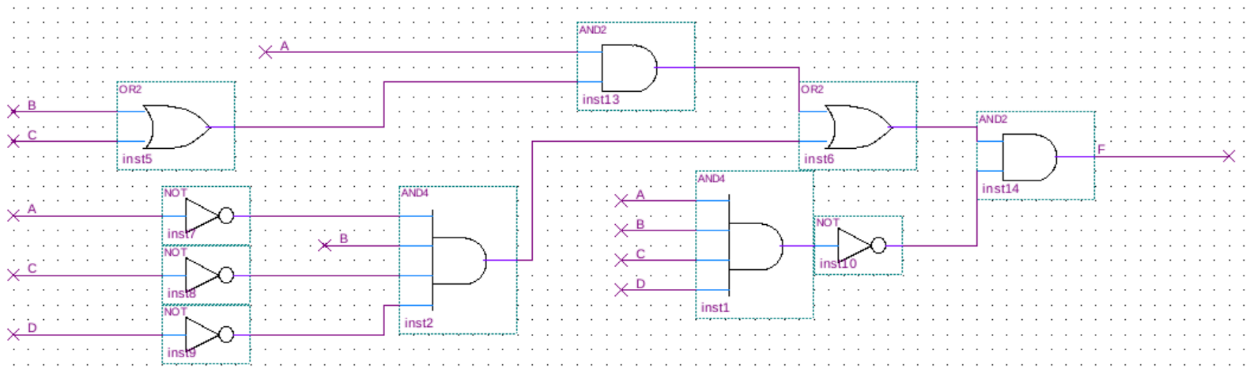
CS232 Project 1: Combinational Circuits

September 10th, 2024

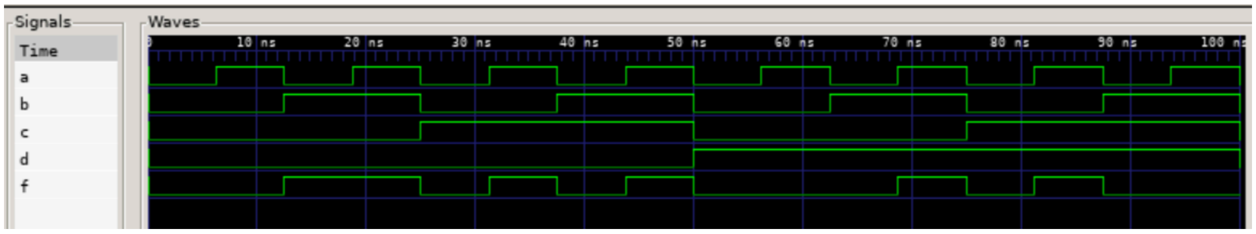
Task 1: Prime Finder

Create a digital circuit using Quartus to determine whether a 4-bit binary input represents a prime number. The circuit outputs a '1' if the number is prime and '0' otherwise.

Circuit:



GHDL Waveform:



Simulation:

https://drive.google.com/file/d/1poJ08QLO12rp2ZaHx6jAnKYDB1wrCDiO/view?usp=sharing

Explanation:

I optimized the testbench code so that it tests all combinations of the 4 switches A, B, C, and D by modifying the time delays to ensure that each possible combination of values (from "0000" to "1111") is tested. Essentially, I created a pattern where each switch toggles after a certain delay, and the delays should be set in a way that all combinations appear in sequence. Just like with binary, it looks like a fractal cascade, with each change in any of the 4 switches being a different number, from 1 to 15. (15 is the largest 4 digit binary number, 1111)

The prime numbers from 1 to 15 are 2, 3, 5, 7, 11, and 13. If we view the timeline and ticks associated with those numbers, we can see that F, the signal for whether a number is prime or not, is on at all those points, meaning that it works!

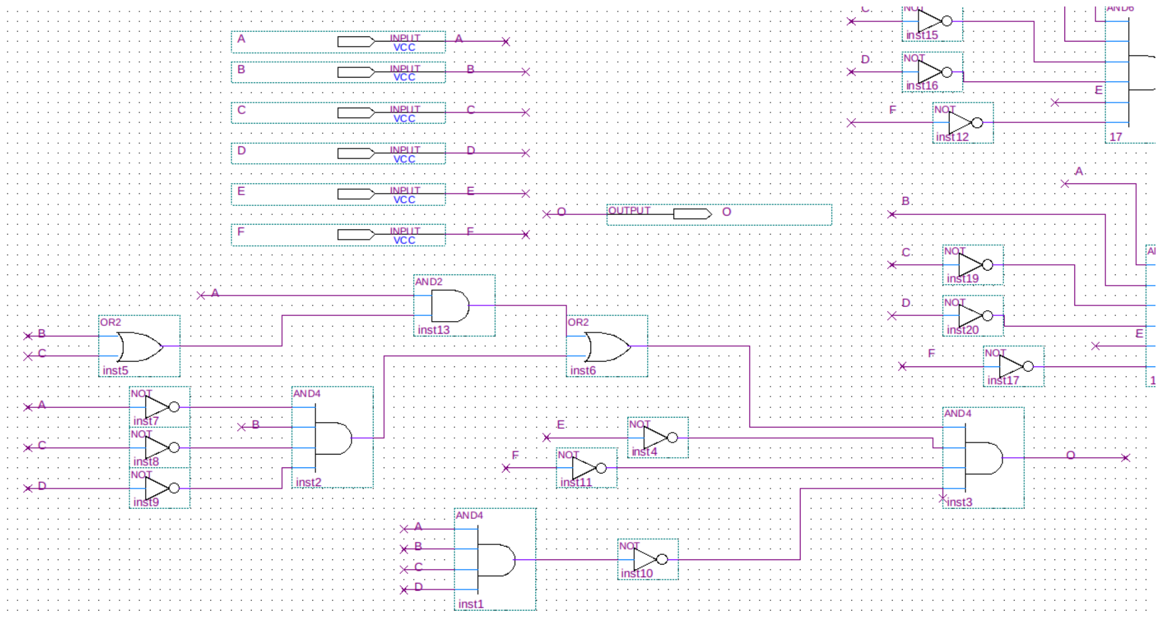
Hardware:

I did not use hardware for this.

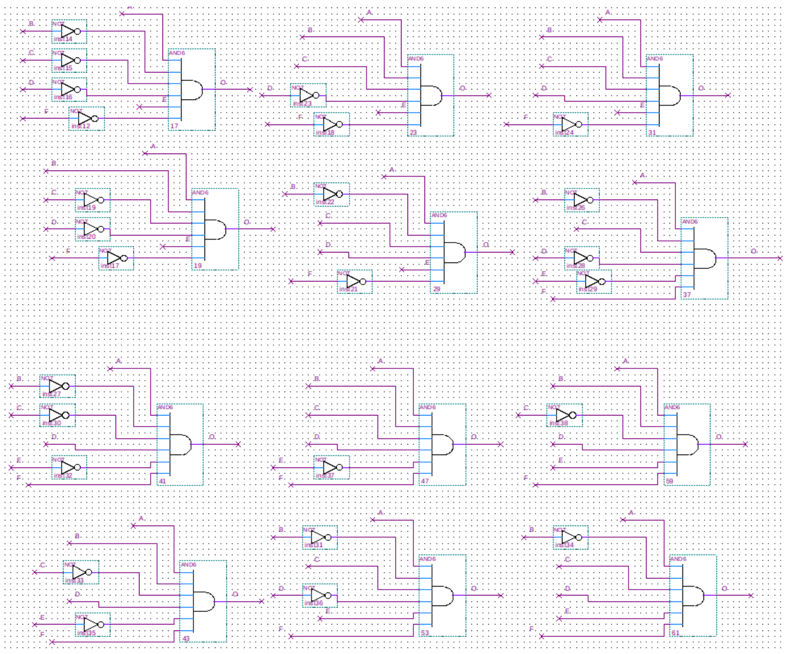
Extension #1: 5 and 6 digit prime numbers

A 6 digit binary number has a maximum value of 63. Unfortunately, while I would love to simplify the circuits as much as possible, there is no actual formula towards finding a prime number, so I just had to brute force it and make a signal for each prime number beyond 13. I also needed to make sure that this code doesn't interfere with the old one.

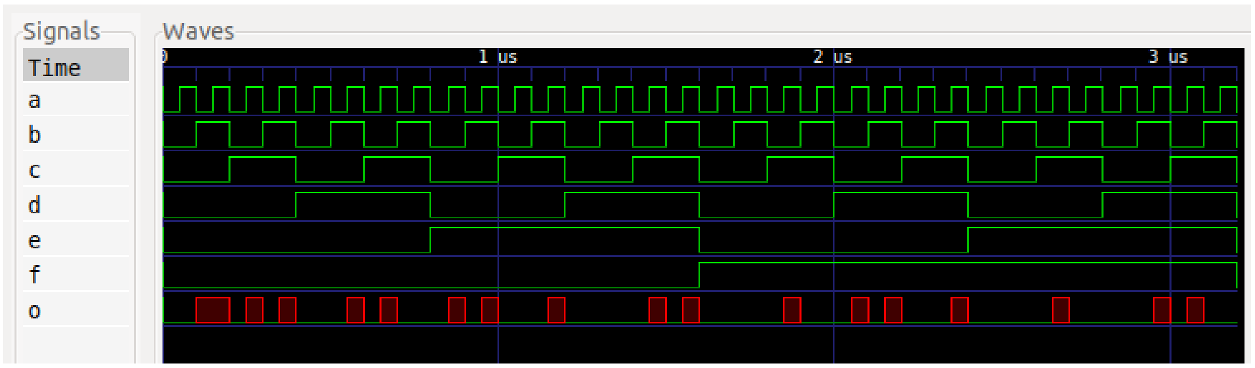
Edited Circuit:



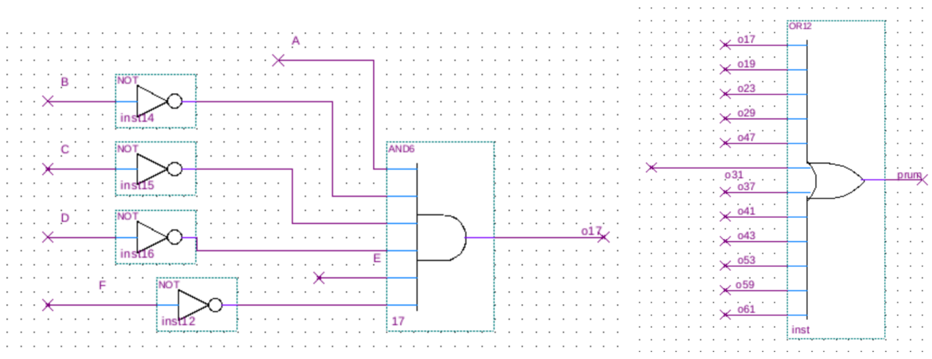
This image shows the previous code for the base project updated so that it doesn't influence any number past 13, since I know it works. Digits beyond 15 use the 5th and 6th bit, so as long as I made sure the input didn't include them, the base case would remain unchanged.

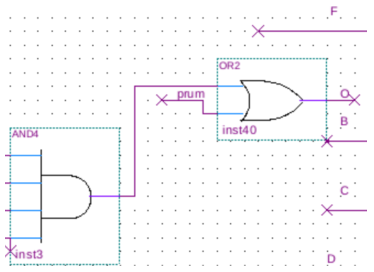


This code is the brute force part: Checking each digit for every prime number and returning an O signal if possible.



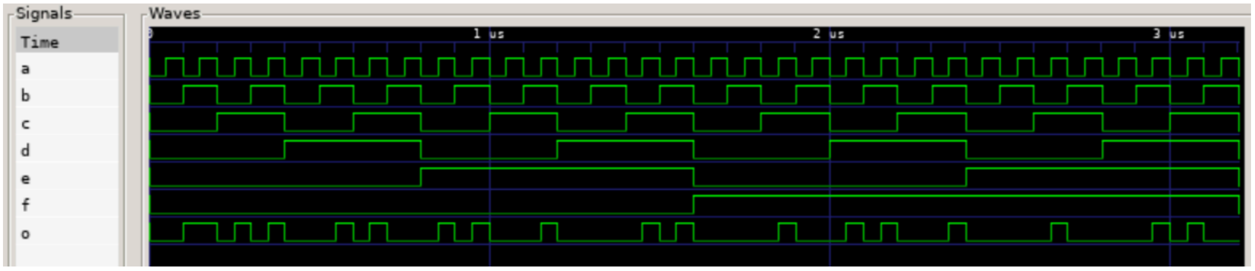
This is where I ran into a problem: Having so many disconnected circuits with the same output overloaded the compiler, to which it didn't know whether to output 1 or some random value, which is why although the signals are at the right spots, the value they output is undefined, hence the red.





To fix this problem, I a wire label to each of the numbers (labeled o_) because I forgot I could do that, added it into a giant OR gate that had a label "prum" that attached to another OR gate placed at the end of original base circuit, so the final check would be the checks for 1-15, OR 16-63.

GHDL Waveform:

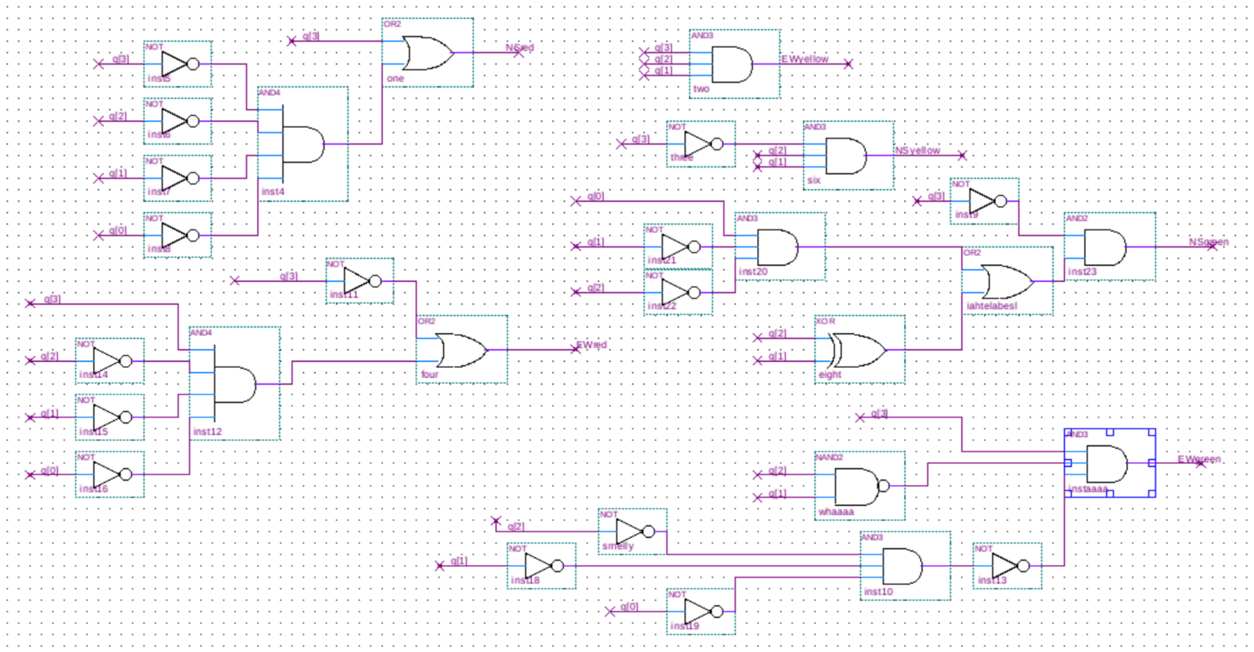


In the end, I got it to work!

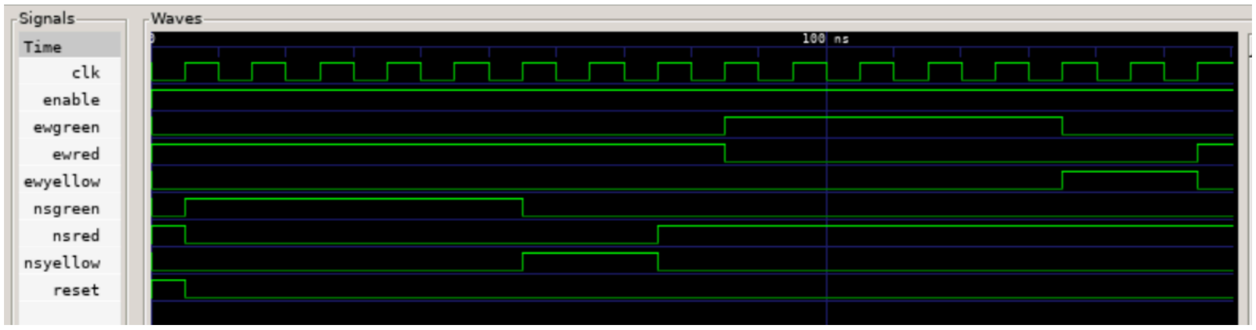
Task 2: Traffic Lights

This project simulates two traffic lights, one for north/south traffic and another for east/west traffic. Each light has red, green, and yellow signals, which change based on a 16-step time cycle controlled by a counter. In hardware, the counter increments when a button is pressed, but in simulation, a clock signal automatically increments the counter. The goal is to correctly simulate the lights' behavior using Quartus and verify it with GHDL waveform output.

Circuit:



GHDL Waveform:



Simulation:

https://drive.google.com/file/d/1tyamFnQQ7J006a4fEeq1YLFb0ZeKmfZt/view?usp=sharing

(LEDS 9-7 are East-West Red, Yellow, Green Respectively, LEDS 2-0 are North-South.)

Explanation:

Similar to the last task, but the code to account all possible combinations of a 4 digit binary number was all given. Although, the symbol itself may not have allotted 0 correctly, which is why the time is short at the beginning. It seems to also be cut off at the end with the supposed 16th tick in clk. 16 in binary is 10000, and since I don't have the 4th digit to account for the 1, it's just 0000, so it should loopback, making it work out in the end!

As we can see from the image, the times at which each light is supposed to be on lines up with their appropriate numbers, and no switches overlap, meaning no 2 switches are on at either ew or ns.

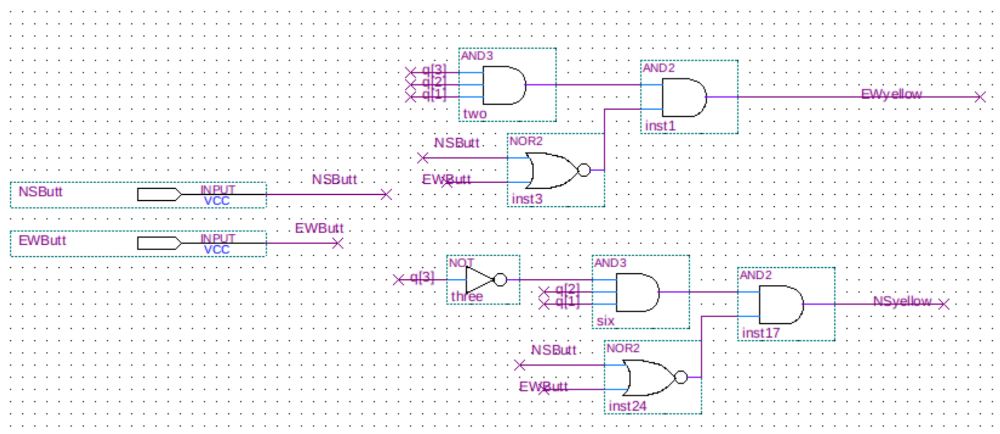
Hardware:

I did not use hardware for this. Unfortunately, it takes a while to walk over to use the boards! :(

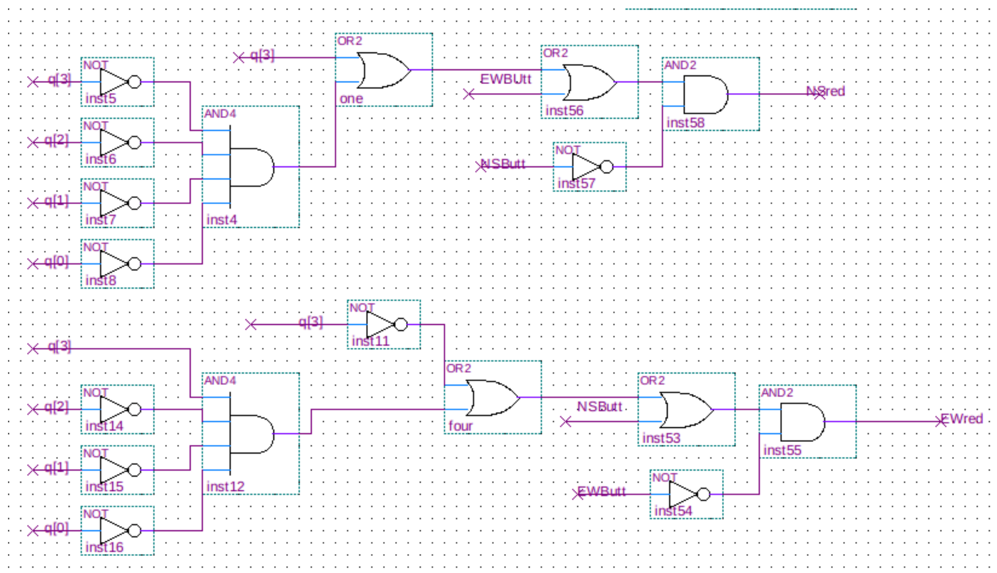
Extension #2: Walk Button

I'm not entirely sure how to make a button that can be pressed anytime since time is only being simulated, but I can simulate one being pressed! There are 2 new signals on the GHDL: EWButt and NSButt (haha) that turn 1 to simulate someone hitting a button.

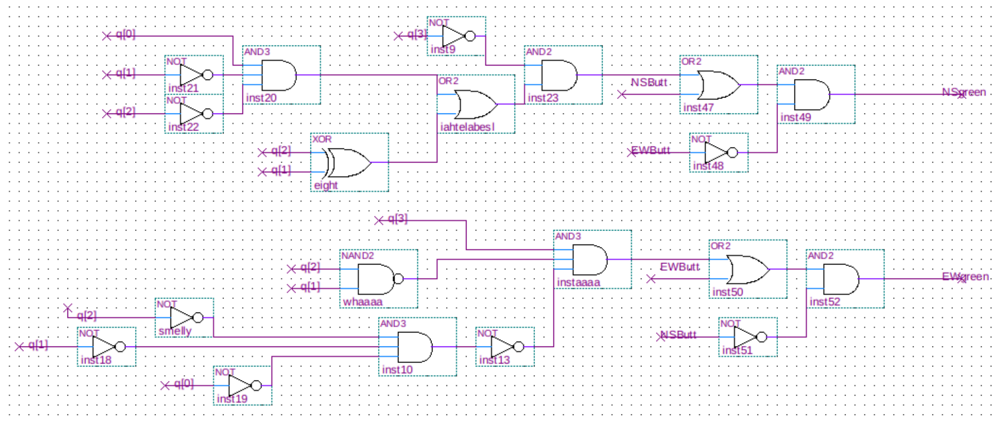
Edited Circuit:



To implement this, I added 2 new input modules (one for each lane) that whenever turned on, automatically makes the other lane's signal red. To do this, I added new AND gates to each of the previous detectors as a final benchmark before triggering so it acts as an override.

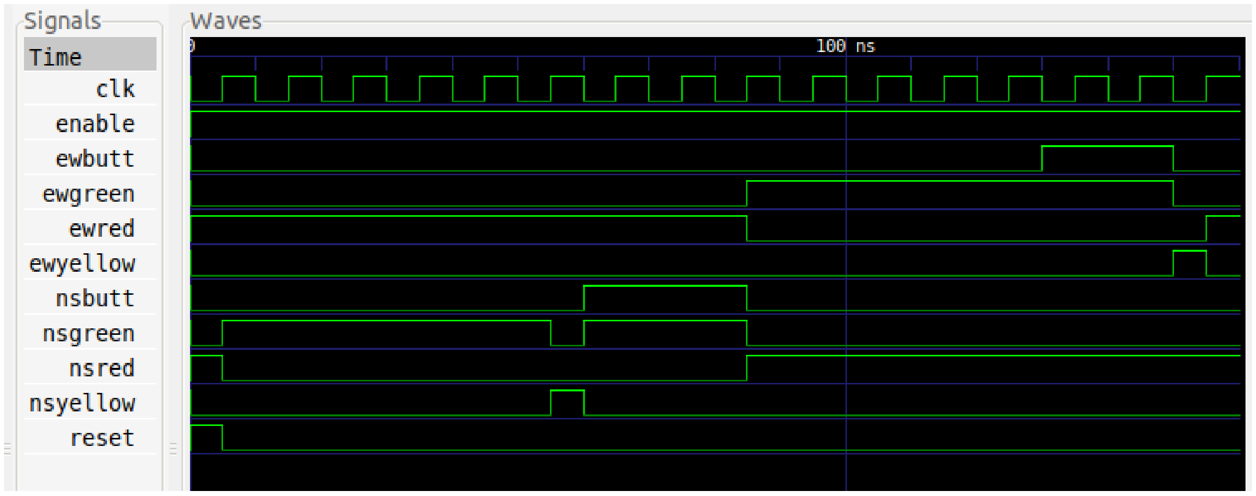


Initially, I was just going to make each button explicitly turn on its respective lane green light on itself in its own circuit, but I experienced the same error as the last extension (multiple circuits having the same output) so I combined it!



Each original switch detector now has a new gate to check if either it should turn on anyway because its corresponding button was pressed, or turn off because its opposing button was pressed.

GHDL Waveform:



This GHDL takes the same counter and rules as the base case, but adds a random input from both buttons at random intervals. From the GHDL, we can see that as soon as the NSButton is pressed, NSGreen instantly switches on and EWRed doesn’t change (since it was already red) but NSYellow also instantly cuts out. When the EWButton was pressed, EWGreen went for longer than it should have, while EWYellow lit up for less as a result. So it works!