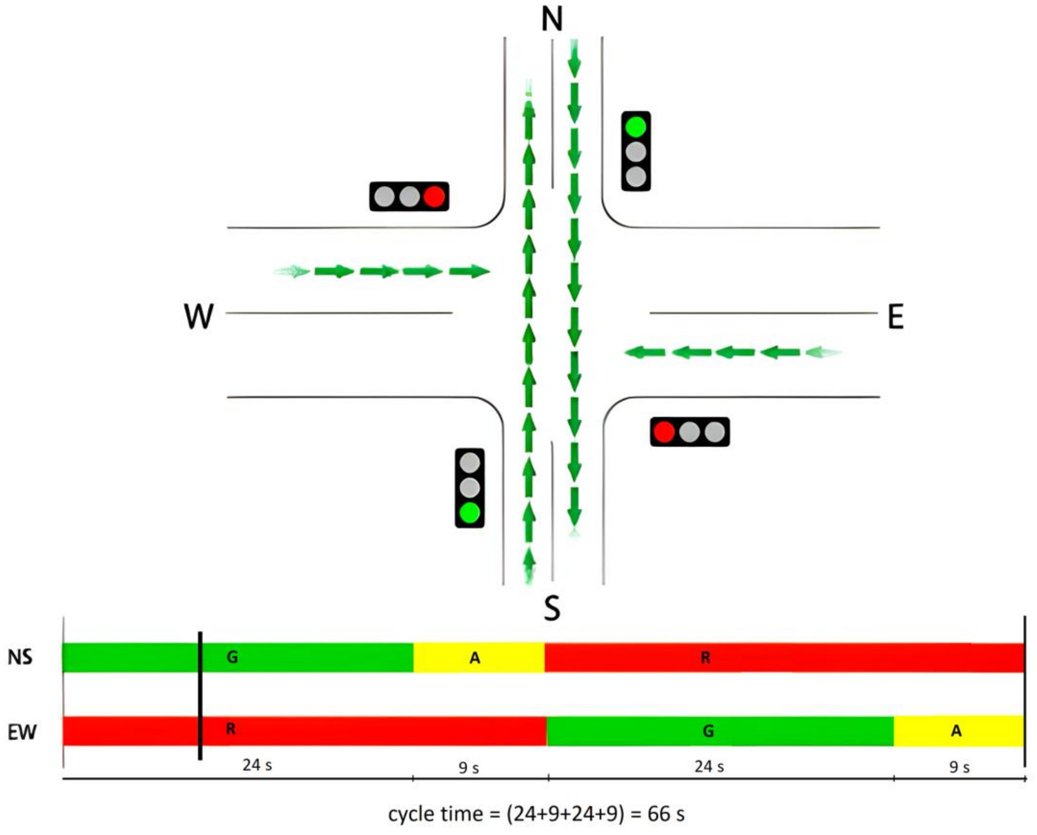
Traffic Light Control System with Pedestrian Crossing Feature (FSM-Based)

Design

The design is based on having the north-south direction as the primary axis, while the east-west direction serves as the secondary axis of the road. These two axes are coupled, meaning that the time for the green and yellow lights in one direction equals the time for the red light in the opposite direction. the green light signifies "go," while the yellow light prepares for the end of "go." This ensures that when cars are passing in one direction due to the green and yellow lights, the red light appears in the opposite direction at the same time. Then, the system switches between the two directions.



For the pedestrian system (Pedestrian Mode), pressing the designated pedestrian button interrupts the "Normal Mode" and switches to "Pedestrian Mode," which stops all four roads, similar to practices in some European countries for enhanced safety. In this mode, the transition stage from "Normal Mode" is carefully considered to prevent errors. The goal is to make the transition between modes as smooth as possible. Subsequently, a unified sequence is executed, such as halting traffic in all directions or turning on the red light in the required direction. In general, the system takes into account the state from which it transitioned in "Normal Mode" to avoid abrupt transitions, such as switching directly from green to red, which could lead to sudden stops in traffic.

How to Build the System

The system was built using a **bottom-up** approach, meaning I started by constructing the system's foundational components and ensuring that each function worked independently before moving on to higher levels.

1. I began by researching traffic light systems, studying the different modes within "Normal Mode" and "Pedestrian Mode," as well as the meaning of various lights, the relationships between them, and the timing for each light. I also examined how the coupling works when four roads intersect perpendicularly.
2. Next, I developed the internal functions needed for "Normal Mode." Once this mode was functional, I move forward to the "Pedestrian Mode."
3. The "Pedestrian Mode" respects the internal state of the currently active "Normal Mode" and only takes action after that state completes. When "Pedestrian Mode" takes over, it pulls CPU control to execute its sequence. It also considers the interrupted state in "Normal Mode" to properly resume the sequence afterward and avoid an aggressive stop.
4. After "Pedestrian Mode" completes its sequence, it returns to the "Normal Mode" state that was next in line before the interruption. For example, if "Pedestrian Mode" was triggered after step 1 in "Normal Mode," the system resumes step 2 in "Normal Mode."

Future Recommendations

There was a proposal for future improvements where the system would not stop all directions but only stop the direction where pedestrians request to cross. For this reason, I added an additional button for the vertical road.

Cons

1. **Delay Inaccuracy:** The delay is not precise due to limitations in the library used and the simulation environment in Proteus.

Challenges

Phase One:

1. Initially, I planned to work with the STM32 Blue Pill board since I have physical hardware. However, I discovered that the ST-Link was broken, prompting me to switch to simulation.
2. I started simulating the STM32 on Proteus, using Eclipse and GCC to build the Blink LED, but encountered issues with no working results.
3. I tested different libraries for STM32 in Proteus, including the original library from Proteus and two custom libraries from external sources, but none of them worked.

4. Suspecting an outdated version of Proteus, I upgraded to the latest version and tested the libraries again, but still, nothing worked.
5. I doubted the STM32 drivers and tested both ready-made drivers from the internet and my own, but none succeeded.
6. Lastly, I questioned the Eclipse IDE, so I downloaded STM32Cube and tested STM-provided code and libraries on different STM32 versions in Proteus, but none of them worked.

Phase Two:

1. I decided to switch to the ATmega32. Using older drivers I developed for Proteus, I built the project with Eclipse, and everything worked well.
2. I tested Blink LED successfully and began constructing the traffic light system based on a finite-state machine.
3. After the finite-state machine system worked correctly, I attempted to use RTOS to address challenges found in the normal design.
4. I tried porting RTOS to ATmega32 from its original GitHub repository, but the build failed due to multiple errors.
5. I then ported an older version of RTOS I had on my laptop, which surprisingly built successfully.
6. I tested Blink LED using RTOS on ATmega32 in Proteus, but the simulation failed. It worked briefly before stopping and generating numerous errors I could not resolve.

Phase Three:

I decided to build the project using ATmega32 without RTOS and test the code in Proteus.