Smart Stoplight Controller

Annie Prasanna Manoharan

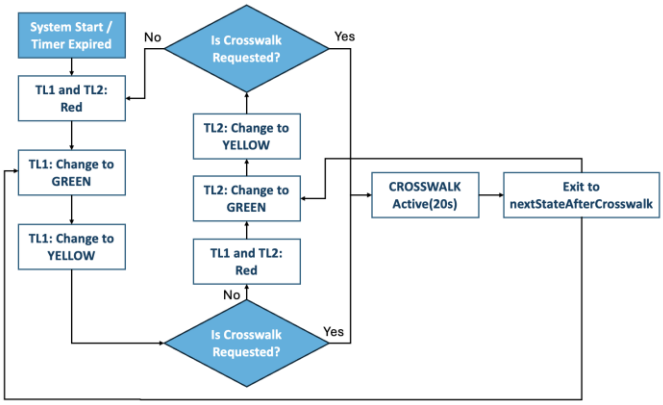
***Overview*— The Smart Stoplight Controller project involves the development of an intelligent traffic light system utilizing an ESP32 microcontroller and the FreeRTOS real-time operating system. This system manages a two-way intersection with adaptive features to enhance efficiency beyond traditional fixed-timing mechanisms. Key components include vehicle detection for dynamic green light adjustments and a pedestrian crosswalk with a countdown display. The integration of FreeRTOS enables concurrent handling of tasks such as sensor monitoring, timer management, and display updates, ensuring responsive operation without blocking delays. This approach addresses the limitations of sequential programming in embedded systems, providing a foundation for scalable traffic management solutions.**

I. Introduction

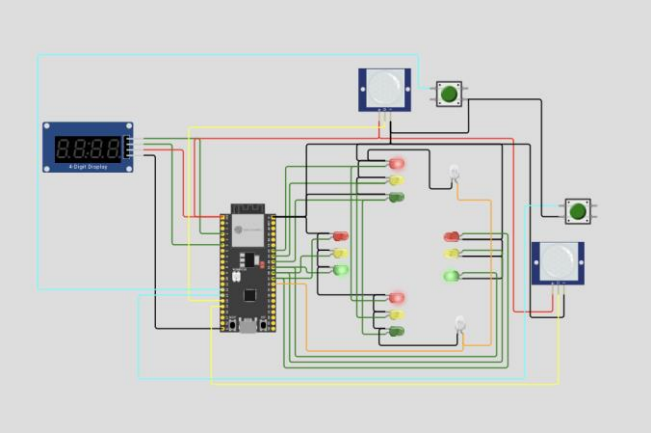
This project details the development of a Smart Stoplight Controller built using an ESP32 and the FreeRTOS real-time operating system. The main goal was to create a traffic control system for a two-way intersection (Traffic Light 1, or TL1, and Traffic Light 2, or TL2) that is more efficient than a simple timed sequence. We integrated features like adaptive timing based on vehicle presence and a responsive pedestrian crosswalk system with a countdown. Using FreeRTOS was essential to handle multiple things happening at once, such as checking sensors, running timers, and updating the display, all without blocking the main process.

II. Application Design

The system architecture leverages FreeRTOS primitives to achieve concurrency and synchronization. Hardware elements encompass LEDs for traffic signals on two roads, passive infrared (PIR) sensors for vehicle detection, push buttons for pedestrian requests, and a TM1637 seven-segment display for countdown visualization. Interrupts are employed for immediate response to sensor activations and button presses, minimizing latency.



The core logic revolves around a state machine that governs light transitions, including green, yellow, and safety red phases for each road, as well as a dedicated crosswalk state. Software timers drive these transitions: a primary one-shot timer controls phase durations, dynamically adjusted based on conditions, while an auto-reload timer manages the crosswalk countdown. Durations are predefined—18 seconds for green (reducible to 8 seconds), 4 seconds for yellow, and 2 seconds for safety gaps—with the crosswalk spanning 20 seconds plus safety.



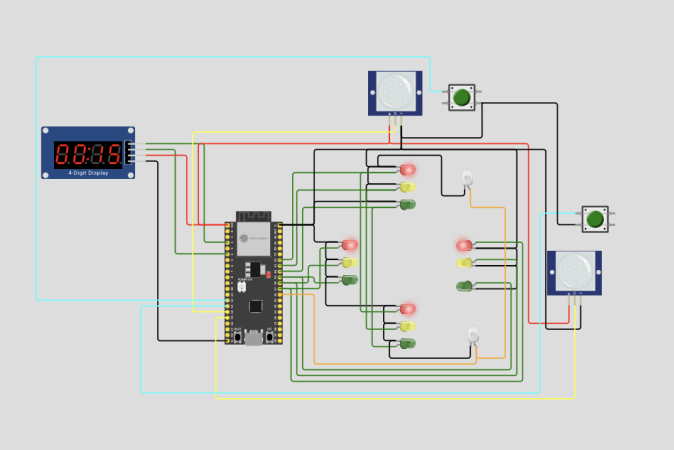
The light sequence is driven by the setLightState function, which determines the phase, sets the appropriate physical lights using the setTrafficLight helper function, and then sets the new duration for the xLightTimer.

The state machine includes the following key states:

State	Purpose	Duration	Next State
TL1_GREEN	TL1 active	18s (or 8s reduced)	TL1_YELLOW
TL1_YELLOW	Warning phase	4s	Safety Gap or CROSSWALK
TL1_TO_TL2_SAFETY	Both Red	2s	TL2_GREEN

TL2_GREEN	TL2 active	18s (or 8s reduced)	TL2_YELLOW
TL2_YELLOW	Warning phase	4s	Safety Gap or CROSSWALK
CROSSWALK_ACTIVE	Pedestrian crossing	22s (20s countdown + 2s safety)	Return to saved phase

Vehicle detection logic utilizes interruptions to set detection flags, which a periodic task evaluates against the current state. If a vehicle awaits the red light while the opposing green exceeds the minimum threshold, the timer period is shortened to expedite the switch, optimizing flow. Pedestrian requests are queued via a semaphore from interrupts, activating the crosswalk only post-yellow phase for safety, with both lights set to red and the display updating sequentially.



Synchronization is ensured through a mutex protecting shared variables from race conditions across tasks and callbacks. A queue and dedicated task handle serial output to prevent garbled messages in the multi-threaded environment. This design partitions responsibilities—light control, status monitoring, and event handling—into distinct tasks, promoting modularity and reliability.

III. Results

The system successfully demonstrates the benefits of using a real-time OS for complex control applications. The main scheduling and the asynchronous button presses run smoothly, ensuring that a sensor input or button press is never missed or delayed by a long light phase.

Testing validated the system's performance in simulated scenarios. Light transitions occurred seamlessly, with adaptive adjustments effectively reducing green durations from 18 seconds to 8 seconds upon detecting waiting vehicles, thereby minimizing idle times. Pedestrian requests were processed without interruption to ongoing phases, activating the crosswalk at safe junctures and displaying accurate countdowns, ensuring no missed events even during extended cycles.

Concurrency was demonstrated through simultaneous operation: sensor checks, display updates, and status logging proceeded without delays or conflicts, as evidenced by consistent debug outputs. Data integrity remained intact, with the mutex preventing inconsistencies in state variables. Overall, the system exhibited robustness, outperforming non-RTOS implementations by responding dynamically to inputs, thus confirming the efficacy of the multi-tasking framework in real-time control applications.

IV. Conclusion

The FreeRTOS-based Smart Stoplight Controller exemplifies an effective concurrent system for traffic management. By employing tasks, timers, semaphores, and mutexes, the design achieves adaptability to vehicle presence and pedestrian needs, enhancing safety and efficiency. This implementation surpasses rigid delay-based systems, highlighting the advantages of event-driven architectures in embedded environments. Future enhancements could incorporate additional sensors for traffic density analysis or network integration for coordinated intersections, advancing toward intelligent urban infrastructure.