1. Traffic Light Control

- **Normal Cycle:**
    - **N-S and E-W directions alternate.**
    - **Timing:**
        - **N-S Green: 8s → Yellow: 2s → Red: 10s.**
        - **E-W Green: 8s → Yellow: 2s → Red: 10s.**
    - **Total cycle time: 40 seconds.**
- **Key Functionalities:**
    - **No overlapping Green states.**
    - **Smooth transitions managed by timers.**

---

2. Pedestrian Crossing

- **Signal Timing:**
    - **Walk Signal (Green):**
        - **5 seconds steady Walk.**
        - **5 seconds flashing Walk (indicating crossing is about to end).**
    - **Don't Walk Signal (Red): Active during traffic Green and Yellow phases.**
- **Button Behavior:**
    - **If Green Time ≤ 2s: Queue the request for the next Red phase.**
    - **If Green Time > 2s:**
        - **Record the current Green time.**
        - **Transition to pedestrian mode:**
            - **Dedicate 2 seconds to remaining Green for vehicles.**
            - **Activate pedestrian crossing (5s steady + 5s flashing Walk).**
        - **Resume the traffic cycle seamlessly.**

---

3. Hardware Setup

- **Core Components:**
    - **Microcontroller (STM32, Arduino, or ESP32).**

- **LEDs for traffic (Red, Yellow, Green) and pedestrian signals (Walk, Don't Walk).**
- **Two push buttons for pedestrian crossing requests.**
- **Breadboard, resistors, wires, and power supply.**

---

4. Software Implementation

- **Traffic Light States:**
  - **N-S Green, N-S Yellow, N-S Red.**
  - **E-W Green, E-W Yellow, E-W Red.**
  - **Pedestrian Walk, Pedestrian Don't Walk.**
- **Event Handling:**
  - **Timers for state transitions.**
  - **Interrupts or polling for pedestrian button presses.**
- **Pedestrian Mode Integration:**
  - **Respond to button presses dynamically.**
  - **Ensure pedestrian crossing is only allowed when traffic is safe (e.g., during Red or extended Green).**

---

5. Testing and Validation

- **Test Scenarios:**
  - **Normal traffic light cycles.**
  - **Button press during active Green phases and edge cases like simultaneous button presses.**
  - **Ensure no Green state overlap.**
  - **Validate pedestrian crossing timings (5s steady Walk + 5s flashing Walk).**
- **Edge Cases:**
  - **Pressing buttons near the end of Red phases.**
  - **Recovery from power failure or reset.**

---

6. Optional Features

- **Emergency Stop Mode: Flash Red in all directions.**
- **Countdown Timer for each state.**
- **Power failure recovery to resume the correct state.**

Tasks

1. **Traffic Light Control Task**:
    - Responsible for controlling the **traffic lights** (Green, Yellow, Red) for both N-S and E-W directions.
    - This task will manage the **state transitions** based on timing.
2. **Pedestrian Crossing Task**:
    - Controls the **Pedestrian signals** (Walk, Don't Walk).
    - It should ensure pedestrian crossing is only activated during Red traffic light phases.
3. **Button Interrupt Task** (Button Press Handler):
    - Monitors **button presses** to trigger pedestrian crossing requests.
    - This task should use **interrupts** to minimize response time and should queue the request to the Pedestrian Task.
4. **Timer/Delay Management Task**:
    - Handles **timers** for all state transitions (traffic light timings and pedestrian crossing durations).
    - This task manages delays, transitions, and periodic updates for the system.

State Diagram and Task Interaction

States:

- **N-S Green, N-S Yellow, N-S Red, E-W Green, E-W Yellow, E-W Red, Pedestrian Walk, Pedestrian Don't Walk.**

Task Breakdown and Details

1. Traffic Light Control Task

- **Priority**: Medium
- **Function**:
    - Controls the traffic light state transitions for **N-S** and **E-W** directions.

- Uses a simple **finite state machine** to manage transitions: Green → Yellow → Red.
- Each state has a predefined **duration** (Green: 8s, Yellow: 2s, Red: 10s).
- When it's time to switch, it triggers the transition to the opposite direction (E-W or N-S).
- **Event Handling**:
  - Wait for the **appropriate signal change** (from timer or interrupt).
  - Change the lights and send updates to the corresponding LEDs.

Task Logic:

- **State Handling**: Based on time or interrupts.
- **Semaphore**: Use semaphores or flags to sync between traffic states and pedestrian crossing states.

---

2. Pedestrian Crossing Task

- **Priority**: High (since pedestrian safety is critical).
- **Function**:
  - Handles the **pedestrian crossing signals** (Walk/Don't Walk).
  - Responds to the button press by setting a flag or signal for pedestrian mode.
  - The **Walk signal** will turn on only when the traffic light is **Red**.
  - Manages **steady Walk** (5 seconds) and **flashing Walk** (5 seconds).

Task Logic:

- **State Handling**: Activated when the pedestrian button is pressed, switches to Walk state when traffic is Red.
- **Semaphore/Mutex**: Use a semaphore to ensure mutual exclusion when updating traffic light states.

---

3. Button Interrupt Task (Button Press Handler)

- **Priority**: High
- **Function**:
  - Monitors the pedestrian crossing button for presses.
  - **Button press** triggers the **pedestrian_request** flag.
  - When the button is pressed, it queues the request to the pedestrian crossing task.

Event Handling:

- Use **GPIO interrupts** to detect the button press and signal the pedestrian crossing task.
- This minimizes polling and improves response time.

4. Timer/Delay Management Task

- **Priority**: Low
- **Function**:
    - Manages delays and periodic timers.
    - Can use **FreeRTOS timers** for task scheduling and state transitions.

Synchronization

- **Semaphores** or **queues** can be used to synchronize between traffic and pedestrian tasks.
- **Mutexes** ensure that only one task (e.g., traffic or pedestrian) controls the traffic light or pedestrian signal at a time.

RTOS Design Summary

1. **Traffic Light Task**: Controls the light cycle based on timing and signals.
2. **Pedestrian Crossing Task**: Responds to button presses and updates pedestrian signals.
3. **Button Press Interrupt**: Detects and handles pedestrian button presses.
4. **State Transitions**: Managed by delays and RTOS synchronization primitives like semaphores or queues.
5. **Timers**: Handles all timing, including the pedestrian crossing and light durations.

1. Semaphores

Semaphores are a synchronization tool used to manage access to shared resources and to signal between tasks. In your traffic light control system, semaphores can be used to:

- Ensure that only one task (either the traffic light control or the pedestrian crossing) is active at a given time.
- Prevent race conditions and ensure that resources (such as traffic light state or pedestrian signal) are not accessed simultaneously by multiple tasks.

Example Use of Semaphore for Traffic Light Control

- We can use a semaphore to ensure that **Pedestrian Crossing** does not conflict with the **Traffic Light Control**.
- **Semaphore Initialization**: Before the tasks run, initialize a semaphore to control access.

- **Traffic Light Task (Using Semaphore)**
- When the Traffic Light Task starts, it will first **take** the semaphore to gain control over the traffic lights. After the task finishes its job (changing lights), it will **give** the semaphore to allow the Pedestrian Crossing Task to run.

- **Pedestrian Crossing Task (Using Semaphore)**
- The Pedestrian Crossing Task will wait for the semaphore to be released before it can activate the pedestrian signals.

- ## 2. Task Notifications
- Task notifications are a lightweight mechanism in FreeRTOS that allow tasks to send signals or data to one another. This can be a more efficient alternative to using queues for simple communication, and it's especially useful in interrupt-driven systems like yours.
- **Pedestrian Button Interrupt (Task Notification)**
- Instead of using semaphores, task notifications can be used to send a signal from the **button press interrupt** to the **Pedestrian Crossing Task**.
- In the button press ISR (Interrupt Service Routine), we notify the **Pedestrian Crossing Task** that a button was pressed, triggering pedestrian crossing.

- **Pedestrian Crossing Task (Notification Wait)**
- The Pedestrian Crossing Task waits for a **task notification** to be sent. When the button is pressed, the ISR will notify the Pedestrian Crossing Task to process the pedestrian crossing logic.

- ## 3. Timers for State Transitions
- FreeRTOS supports software timers that can be used to create delays and handle periodic tasks. Instead of relying on `vTaskDelay`, you can use software timers to handle the timing of state transitions.
- **Creating a Timer for Traffic Light Transitions**
- You can create a **timer** that automatically triggers when a state change is required (e.g., for the 8s Green → 2s Yellow → 10s Red cycle).

RTOS Configuration Tips

1. **Priorities**:
    - Assign higher priorities to tasks that are critical (e.g., the Pedestrian Crossing Task).
    - Assign lower priorities to tasks that are less time-sensitive (e.g., the Traffic Light Control Task).
2. **Task Stack Sizes**:
    - Ensure that each task has sufficient stack size based on the complexity of the logic inside the task.
    - For example, the Traffic Light Control task may require more memory if it's handling more states or logic.
3. **Memory Allocation**:
    - Be mindful of the available heap and stack space, especially if you're using a microcontroller with limited memory resources.

Summary of Synchronization Techniques

- **Semaphores**: Used to synchronize tasks that share resources, such as traffic lights and pedestrian signals.
- **Task Notifications**: Used to notify tasks when an event (like a button press) occurs, making it efficient for interrupt-driven actions.
- **Timers**: Used to manage time-based state transitions for traffic light cycles and pedestrian signal management.
