Ain Shams University - Faculty of Engineering

CSE411 (UG2018) – Real Time Embedded Systems-spring 25

Team 18

Advanced Power Window Control System

Mohamed Omar ElBialy	21P0068
Omar Mohamed Korkor	21P0065
Ezz Eldin Alaa Eldin Omar	21P0100
Karim Sherif Louis	21p0223
Ali Mohamed Refaat	21p0105

Table of Contents

1 Objective

This project aims to develop an advanced power window control system using the Tiva C Series TM4C123GH6PM microcontroller and FreeRTOS for real-time task management. The system will control a front passenger door window, featuring manual and automatic operation, obstacle detection, window lock functionality, and precise position tracking. The implementation emphasizes safety, reliability, and efficient resource utilization.

2 Components Used

A variety of electronic components were used, these included:

- Basic components
  - Male-Male jumper wires
  - Male-female jumper wires
  - Push buttons
  - Switches
- Sensory components
  - Limit Switch: Used to detect end positions.
  - IR sensor: Used to detect obstacles in the window path.
  - Rotary Encoder: Used to detect the position of the window.
- Alert Components
  - Buzzer: Used to signal that an object was detected.
  - LCD: Used to display different statuses of the system.
- Actuator & Control components
  - 12 DC motor
  - H-Bridge: Used as a control circuit for the motor.

3 System Architecture

The system architecture of the Power Window Control System is designed around the Tiva C microcontroller; the system implements real-time responsiveness and safety. It integrates multiple input sources such as push buttons, limit switches, rotary encoders, and obstacle detection sensors, while controlling outputs like a DC motor, buzzer, and LCD display. FreeRTOS is used to manage concurrent tasks such as handling manual/automatic commands, monitoring safety features, and controlling motor actions. The architecture ensures that each input or event is serviced efficiently and reliably, supporting both driver and passenger control with features like window lock, jam protection, and status reporting.

3.1 Port Configuration

Several ports were configured to interface the Tiva-c with the components used.

```
//global definitions
#define Buttons_Motor_Port            GPIO_PORTA_BASE
#define Sensors_Port                  GPIO_PORTC_BASE
#define Object_Detection_Sensor       (1<<4)
#define Window_Upper_Limit            (1<<5)
#define Window_Lower_Limit            (1<<6)
#define Window_Lock_Switch            (1<<7)
#define Passenger_Elevate_Button      (1<<2)
#define Passenger_Lower_Button        (1<<3)
#define Driver_Elevate_Button         (1<<4)
#define Driver_Lower_Button           (1<<5)

#define GPIO_PORTA_BASE               0x40004000  // GPIO Port A
#define GPIO_PORTB_BASE               0x40005000  // GPIO Port B
#define GPIO_PORTC_BASE               0x40006000  // GPIO Port C
#define GPIO_PORTD_BASE               0x40007000  // GPIO Port D

#define DC_Motor_In1                  (1<<6)
#define DC_Motor_In2                  (1<<7)
#define DC_Motor_Enable                0x02
```

3.1.1 Port A

Pins PA2-PA5 are configured as inputs while pins PA6 and PA7 are configured as outputs. All pins are enabled for digital functionality.

PIN functions are as follows:

- PA2: interfaces the passenger elevate button.
- PA3: interfaces the passenger lower button.
- PA4: interfaces the driver elevate button.
- PA5: interfaces the driver lower button.
- PA6: Used to control motor direction using the H-bridge (motor-input).
- PA7: Used to control motor direction using the H-bridge (motor-input).

3.1.2 Port B

Pins PB2-PB3 are configured as I2C communication lines for the lcd

- PB2: Used as the serial clock line.
- PB3: Used as the serial data line.

3.1.3 Port C

Pins PC4-PC7 are all used as inputs.

PIN functions:

- PC4: interfaces the IR sensor used for object detection.
- PC5: Interfaces the limit switch that marks the upper limit for the window.
- PC6: Interfaces the limit switch that marks the lower limit for the window.
- PC7: Interfaces the passenger lock switch.

3.1.4 Port F

Pin PF1 is configured as output.

PIN functions:

- PF1: Used as control the motor enable (pin in the H-bridge).

3.2 System Components

The power window control system is divided into functional components, each implemented as a dedicated FreeRTOS task. These components work together to create a complete, reliable window control system

3.2.1 Window Movement Control

Window movement is controlled by several tasks, each one serving a purpose based on movement is manual\automatic and whether this is the driver or the passenger.

- Driver Window Control:

```
void vDUpTask(void *pvParameters) {
    xSemaphoreTake(xDriverWindowElevateTaskUnlockerSemaphore, 0);
    while (1){
        xSemaphoreTake(xDriverWindowElevateTaskUnlockerSemaphore, portMAX_DELAY);
        // delay for debounce
        vTaskDelay(800/portTICK_RATE_MS);

        driver_elevate_button_state = GPIOPinRead(Buttons_Motor_Port, Driver_Elevate_Button);
        //manual
        if ( driver_elevate_button_state == HIGH && window_state != WINDOW_CLOSED){
            // Update LCD to show operation
            LCD_Clear();
            LCD_SetCursor(0, 0);
            LCD_Print("Driver Manual");
            LCD_SetCursor(1, 0);
            LCD_Print("Window Moving Up");
            // while its pressed
            while (driver_elevate_button_state == HIGH && window_state != WINDOW_CLOSED){
                // Enable the motor first
                GPIO_PORTF_DATA_R |= DC_Motor_Enable;
                // move window up
                GPIO_PORTA_DATA_R |= DC_Motor_In1;
                GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
                operation = UP;
                window_state = MIDDLE;
                driver_elevate_button_state = GPIOPinRead(Buttons_Motor_Port, Driver_Elevate_Button);
                last_task = DU;
            }
            // Disable the motor first
            GPIO_PORTF_DATA_R &= ~DC_Motor_Enable;
            // Then clear direction pins
            GPIO_PORTA_DATA_R &= ~DC_Motor_In1;
            GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
            last_task = STOP;
            // Update LCD status
```

```c
        //automatic
        else if (driver_elevate_button_state == LOW && window_state != WINDOW_CLOSED){
            // Update LCD to show operation
            LCD_Clear();
            LCD_SetCursor(0, 0);
            LCD_Print("Driver Auto");
            LCD_SetCursor(1, 0);
            LCD_Print("Window Moving Up");
            // Enable the motor first
            GPIO_PORTF_DATA_R |= DC_Motor_Enable;
            // move window up
            GPIO_PORTA_DATA_R |= DC_Motor_In1;
            GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
            operation = UP;
            window_state = MIDDLE;
            last_task = DU;
        }
```

The driver control supports both automatic and manual elevating or lowering the window. On semaphore acquisition, the pins interfaced with the H-Bridge are set to the suitable combination to lower or elevate the window (different motor rotations) and the motor enable is set high. The task figures out whether the user wants a manual or automatic movement by checking if the driver_elevate button is still pressed, if the user is still pressing (elongated press) this is a manual elevation or lowering of the window, if the button status was low that means it was a quick press and the window is elevated or lowered automatically without a continuous press.

- Passenger window control:

```c
}

void vPUpTask(void *pvParameters) {
    xSemaphoreTake(xPassengerWindowElevateTaskUnlockerSemaphore, 0);
    while (1){
        xSemaphoreTake(xPassengerWindowElevateTaskUnlockerSemaphore, portMAX_DELAY);
        vTaskDelay(800/portTICK_RATE_MS);
        passenger_elevate_button_state = GPIOPinRead(Buttons_Motor_Port, Passenger_Elevate_Button);
        // MANUAL MODE
        if ( passenger_elevate_button_state == HIGH && window_state != WINDOW_CLOSED){
            // Update LCD to show operation
            LCD_Clear();
            LCD_SetCursor(0, 0);
            LCD_Print("Passenger Manual");
            LCD_SetCursor(1, 0);
            LCD_Print("Window Moving Up");

            while (passenger_elevate_button_state == HIGH && window_state != WINDOW_CLOSED){
                // Enable the motor first
                GPIO_PORTF_DATA_R |= DC_Motor_Enable;
                // move window up
                GPIO_PORTA_DATA_R |= DC_Motor_In1;
                GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
                last_task = PU;
                operation = UP;
                window_state = MIDDLE;
                passenger_elevate_button_state = GPIOPinRead(Buttons_Motor_Port, Passenger_Elevate_Button);
            }
```

```c
        // AUTOMATIC MODE
        else if (passenger_elevate_button_state == LOW && window_state != WINDOW_CLOSED){
            // Update LCD to show operation
            LCD_Clear();
            LCD_SetCursor(0, 0);
            LCD_Print("Passenger Auto");
            LCD_SetCursor(1, 0);
            LCD_Print("Window Moving Up");
            // Enable the motor first
            GPIO_PORTF_DATA_R |= DC_Motor_Enable;
            // move window up
            GPIO_PORTA_DATA_R |= DC_Motor_In1;
            GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
            operation = UP;
            last_task = PU;
            window_state = MIDDLE;
        }
    }
}
```

The passenger window buttons have the same functionalities as the driver, but the driver can choose to lock window control to driver buttons only.

3.2.2 Window Safety Features

- As mentioned, the driver can choose to disable the passenger buttons.

```c
void vLockWindows(void *pvParameters) {
    xSemaphoreTake(xLockWindowsTaskUnlockerSemaphore, 0);
    while (1){
        // Turn buzzer on
        GPIO_PORTD_DATA_R |= buzzen;
        // Simple delay - consider replacing with a timer-based delay for better accuracy
        for(volatile int i = 0; i < 1000000; i++);
        // Turn buzzer off
        GPIO_PORTD_DATA_R &= ~buzzen;
        xSemaphoreTake(xLockWindowsTaskUnlockerSemaphore, portMAX_DELAY);
        // delay for debounce
        vTaskDelay(500/portTICK_RATE_MS);
        lock_state = GPIOPinRead(Sensors_Port,Window_Lock_Switch);
        if (lock_state == HIGH){
            vTaskSuspend(xPassengerWindowElevateTaskHandle);
            vTaskSuspend(xPassengerWindowLowerTaskHandle);

            // Update LCD with lock status
            LCD_Clear();
            LCD_SetCursor(0, 0);
            LCD_Print("Window Control");
            LCD_SetCursor(1, 0);
            LCD_Print("Status: LOCKED");

            xSemaphoreTake(xLockWindowsTaskUnlockerSemaphore, portMAX_DELAY);
        }
        else{
            vTaskResume(xPassengerWindowElevateTaskHandle);
            vTaskResume(xPassengerWindowLowerTaskHandle);

            // Update LCD with unlock status
            LCD_Clear();
            LCD_SetCursor(0, 0);
            LCD_Print("Window Control");
            LCD_SetCursor(1, 0);
            LCD_Print("Status: UNLOCKED");

            xSemaphoreTake(xLockWindowsTaskUnlockerSemaphore, portMAX_DELAY);
        }
```

This is done by suspending the tasks responsible for passenger lowering and elevating and resuming when the switch is pressed again.

- The system can detect obstacles in window path using IR Sensor

```c
void vODetection(void *pvParameters) {
    xSemaphoreTake(xObstacleDetectionUnlockerSemaphore, 0);
    while (1) {
        // Turn buzzer on
        GPIO_PORTD_DATA_R |= buzzen;
        // Simple delay - consider replacing with a timer-based delay for better accuracy
        for(volatile int i = 0; i < 1000000; i++);
        // Turn buzzer off
        GPIO_PORTD_DATA_R &= ~buzzen;
        xSemaphoreTake(xObstacleDetectionUnlockerSemaphore, portMAX_DELAY);
        vTaskDelay(200/portTICK_RATE_MS);
        ir_sensor_state = GPIOPinRead(Sensors_Port,Object_Detection_Sensor);
        if ( ir_sensor_state == LOW && operation == UP) {
            // Update LCD with obstacle detection
            LCD_Clear();
            LCD_SetCursor(0, 0);
            LCD_Print("OBSTACLE DETECTED!");
            LCD_SetCursor(1, 0);
            LCD_Print("Reversing window...");

            // Disable the motor first
            GPIO_PORTF_DATA_R &= ~DC_Motor_Enable;
            // Then clear direction pins
            GPIO_PORTA_DATA_R &= ~DC_Motor_In1;
            GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
            vTaskDelay(1000/portTICK_RATE_MS);
            // Enable the motor first
            GPIO_PORTF_DATA_R |= DC_Motor_Enable;
            // Set motor direction to move down
            GPIO_PORTA_DATA_R &= ~DC_Motor_In1;
            GPIO_PORTA_DATA_R |= DC_Motor_In2;
            vTaskDelay(1000/portTICK_RATE_MS);
            // Disable the motor first
            GPIO_PORTF_DATA_R &= ~DC_Motor_Enable;
            // Then clear direction pins
            GPIO_PORTA_DATA_R &= ~DC_Motor_In1;
            GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
            vTaskSuspend(xDriverWindowElevateTaskHandle);
            vTaskSuspend(xPassengerWindowElevateTaskHandle);
```

On detection the system is interrupted and the ISR points to the object detection task which stops then reverses the direction to lower the window to avoid jamming. A buzzer is enabled to alert the user.

3.2.3 Position Tracking

The system uses a rotary encoder and limit switched to track the position of the window.

There are 2 limit switches for upwards and downwards direction. On Upper limit switch press, an interrupt stops the system and the ISR gives the semaphore for the UpperLimit task.

```
void vUpperLimit(void *pvParameters) {
    xSemaphoreTake(xUpperLimitUnlockerSemaphore, 0);
    while (1) {
        // Turn buzzer on
        GPIO_PORTD_DATA_R |= buzzen;

        for(volatile int i = 0; i < 1000000; i++);
        // Turn buzzer off
        GPIO_PORTD_DATA_R &= ~buzzen;
        xSemaphoreTake(xUpperLimitUnlockerSemaphore, portMAX_DELAY);
        upper_limit_switch_state = GPIOPinRead(Sensors_Port, Window_Upper_Limit);
        // STOPPING MOTOR AT UPPER LIMIT
        if ( (upper_limit_switch_state == LOW) && (operation == UP)) {  // Function to check if an obstacle is detected
            // Disable the motor first
            GPIO_PORTF_DATA_R &= ~DC_Motor_Enable;
            // Then clear direction pins
            GPIO_PORTA_DATA_R &= ~DC_Motor_In1;
            GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
            window_state = WINDOW_CLOSED;
            vTaskSuspend(xDriverWindowElevateTaskHandle);
            vTaskSuspend(xPassengerWindowElevateTaskHandle);
            // Update LCD with window closed status
            LCD_Clear();
            LCD_SetCursor(0, 0);
            LCD_Print("Window Position");
            LCD_SetCursor(1, 0);
            LCD_Print("Status: CLOSED");
        }
        // RE-ALLOW WINDOW CLOSING OPTION
        else {
            vTaskResume(xDriverWindowElevateTaskHandle);
            vTaskResume(xPassengerWindowElevateTaskHandle);
        }
    }
}
```

The task acquires the semaphore and is unblocked, it then checks if for limit switch status and operation, it stops the motor if the operation is up and sets the window as closed meaning the window won't elevate again unless it is lowered first and set as not closed.

The Lower limit switch works with the same logic.

```
}

void vLowerLimit(void *pvParameters) {
    xSemaphoreTake(xLowerLimitUnlockerSemaphore, 0);
    while (1) {
        // Turn buzzer on
        GPIO_PORTD_DATA_R |= buzzen;

        for(volatile int i = 0; i < 1000000; i++);
        // Turn buzzer off
        GPIO_PORTD_DATA_R &= ~buzzen;
        xSemaphoreTake(xLowerLimitUnlockerSemaphore, portMAX_DELAY);
        lower_limit_switch_state = GPIOPinRead(Sensors_Port, Window_Lower_Limit);
        //stop motor at lower limit
        if ((lower_limit_switch_state == LOW) && (operation == DOWN)) {
            // Disable the motor first
            GPIO_PORTF_DATA_R &= ~DC_Motor_Enable;
            // Then clear direction pins
            GPIO_PORTA_DATA_R &= ~DC_Motor_In1;
            GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
            window_state = WINDOW_OPEN;
            vTaskSuspend(xDriverWindowLowerTaskHandle);
            vTaskSuspend(xPassengerWindowLowerTaskHandle);
            // Update LCD with window open status
            LCD_Clear();
            LCD_SetCursor(0, 0);
            LCD_Print("Window Position");
            LCD_SetCursor(1, 0);
            LCD_Print("Status: OPEN");
        }
        //allow windoes to be used again
        else {
            vTaskResume(xDriverWindowLowerTaskHandle);
            vTaskResume(xPassengerWindowLowerTaskHandle);
        }
    }
}
```

The rotary encoder helps the system track the position of the window in intermediary phases other than Closed or Open.

The encoder tracking task is awakened by notification from ISR. In our system min position is 0 and the maximum is 10.

```c
void vEncoderTask(void *pvParameters)
{
    // Initialize display
    LCD_Clear();
    LCD_SetCursor(0,0);
    LCD_Print("Encoder:");
    LCD_SetCursor(1,0);
    {
        char buf[16];
        snprintf(buf, sizeof(buf), "Pos:%2ld Dir:%2ld", encoderPos, encoderDirection);
        LCD_Print(buf);
    }

    for(;;)
    {
        // Wait for notification from ISR
        ulTaskNotifyTake(pdTRUE, portMAX_DELAY);

        // Clamp range
        if(encoderPos < ENC_MIN_POS) encoderPos = ENC_MIN_POS;
        if(encoderPos > ENC_MAX_POS) encoderPos = ENC_MAX_POS;

        // Display updated position and direction info for debugging
        LCD_SetCursor(1,0);
        {
            char buf[16];
            snprintf(buf, sizeof(buf), "Pos:%2ld Dir:%2ld", encoderPos, encoderDirection);
            LCD_Print(buf);
        }

        // Map encoder to window position if needed
        if(encoderPos <= 2) {
            window_state = WINDOW_OPEN;
        } else if(encoderPos >= 8) {
            window_state = WINDOW_CLOSED;
        } else {
            window_state = MIDDLE;
        }
```

```c
if(encoderPos == 0){
    GPIO_PORTF_DATA_R &= ~DC_Motor_Enable;
    // Then clear direction pins
    GPIO_PORTA_DATA_R &= ~DC_Motor_In1;
    GPIO_PORTA_DATA_R &= ~DC_Motor_In2;

    window_state = WINDOW_OPEN;

    vTaskSuspend(xDriverWindowLowerTaskHandle);
    vTaskSuspend(xPassengerWindowLowerTaskHandle);
    // Update LCD with window open status
    LCD_Clear();
    LCD_SetCursor(0, 0);
    LCD_Print("Window Position");
    LCD_SetCursor(1, 0);
    LCD_Print("Status: OPEN");
```

If the encoder position reaches 0 that means that window is fully open, it behaves like the limit switch, suspending the lower task and setting the window status OPEN effectively stopping the window from lowering below the limit. If the encoder value changes from 0 it resumes the lower tasks.

```c
        if(encoderPos == 10){
            // Disable the motor first
            GPIO_PORTF_DATA_R &= ~DC_Motor_Enable;
            // Then clear direction pins
            GPIO_PORTA_DATA_R &= ~DC_Motor_In1;
            GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
            window_state = WINDOW_CLOSED;
            vTaskSuspend(xDriverWindowElevateTaskHandle);
            vTaskSuspend(xPassengerWindowElevateTaskHandle);

            // Update LCD with window closed status
            LCD_Clear();
            LCD_SetCursor(0, 0);
            LCD_Print("Window Position");
            LCD_SetCursor(1, 0);
            LCD_Print("Status: CLOSED");
        }
        else {
            vTaskResume(xDriverWindowElevateTaskHandle);
            vTaskResume(xPassengerWindowElevateTaskHandle);
        }
    }
}
```

Likewise, the encoder works similarly when it reaches 10 (the maximum), working like the upper limit switch, it stops the motor, suspends the elevate tasks until the position changes and sets the window status as closed.

3.2.4 User Updates

The system notifies the user of status changes using LCD display and buzzer for warnings.

- LCD: The system uses a 16x2 I2C-based character LCD to provide real-time user feedback. During initialization, the LCD is configured in 4-bit mode using a sequence of control commands via I2C.

```c
// Controls backlight
void LCD_SetBacklight(uint8_t on) {
    lcd_backlight = on ? 0x08 : 0x00;
}

// Low-level I2C + LCD
static void I2C0_Init(void) {
    // 1. Turn on I2C and GPIO clocks
    SYSCTL->RCGCI2C |= 1;                        // Enable I2C0 clock
    SYSCTL->RCGCGPIO |= (1U << 1);               // Enable GPIO Port B clock
    while (!(SYSCTL->PRGPIO & (1U << 1)));       // Wait until ready

    // 2. Configure PB2 (SCL) and PB3 (SDA)
    GPIOB->AFSEL |= (1U << 2) | (1U << 3);       // Enable alternate function
    GPIOB->ODR |= (1U << 3);                     // SDA open-drain
    GPIOB->DEN |= (1U << 2) | (1U << 3);         // Digital enable
    GPIOB->PCTL |= (3U << 8) | (3U << 12);       // Select I2C function

    // 3. Configure I2C module
    I2C0->MCR = 0x10;                            // Enable I2C master
    I2C0->MTPR = 39;                             // Set speed to ~50kHz
}

static void I2C0_WriteByte(uint8_t addr, uint8_t data) {
    I2C0->MSA = (addr << 1);
    I2C0->MDR = data;
    I2C0->MCS = 3;
    while (I2C0->MCS & 1);
}

static void LCD_PulseEnable(uint8_t data) {
    I2C0_WriteByte(LCD_ADDR, data | LCD_EN | lcd_backlight);
    I2C0_WriteByte(LCD_ADDR, data & ~LCD_EN | lcd_backlight);
}

static void LCD_Write4Bits(uint8_t data) {
    I2C0_WriteByte(LCD_ADDR, data | lcd_backlight);
    LCD_PulseEnable(data);
}
```

- Elevating and lowering

```
        // Update LCD status
        LCD_Clear();
        LCD_SetCursor(0, 0);
        LCD_Print("Window: Middle");
        LCD_SetCursor(1, 0);
        LCD_Print("Manual Up Done");
    }
    //automatic
```

```
        // Update LCD to show operation
        LCD_Clear();
        LCD_SetCursor(0, 0);
        LCD_Print("Driver Auto");
        LCD_SetCursor(1, 0);
        LCD_Print("Window Moving Up");
```

```
        // Update LCD status
        LCD_Clear();
        LCD_SetCursor(0, 0);
        LCD_Print("Window: Middle");
        LCD_SetCursor(1, 0);
        LCD_Print("Manual Down Done");
    }
    else if (driver_lower_button_state == LOW && window_state
        // Update LCD to show operation
        LCD_Clear();
        LCD_SetCursor(0, 0);
        LCD_Print("Driver Auto");
        LCD_SetCursor(1, 0);
        LCD_Print("Window Moving Down");
```

- Limits reached

```
    // Update LCD with window closed statu
    LCD_Clear();
    LCD_SetCursor(0, 0);
    LCD_Print("Window Position");
    LCD_SetCursor(1, 0);
    LCD_Print("Status: CLOSED");
}
// RE-ALLOW WINDOW CLOSING OPTION
```

```
    // Update LCD with window open status
    LCD_Clear();
    LCD_SetCursor(0, 0);
    LCD_Print("Window Position");
    LCD_SetCursor(1, 0);
    LCD_Print("Status: OPEN");
```

- Driver Lock

```
        // Update LCD with lock status
        LCD_Clear();
        LCD_SetCursor(0, 0);
        LCD_Print("Window Control");
        LCD_SetCursor(1, 0);
        LCD_Print("Status: LOCKED");

        xSemaphoreTake(xLockWindowsTaskUnlockerSemaph
    }
    else{
        vTaskResume(xPassengerWindowElevateTaskHandle
        vTaskResume(xPassengerWindowLowerTaskHandle);

        // Update LCD with unlock status
        LCD_Clear();
        LCD_SetCursor(0, 0);
        LCD_Print("Window Control");
        LCD_SetCursor(1, 0);
        LCD_Print("Status: UNLOCKED");
```

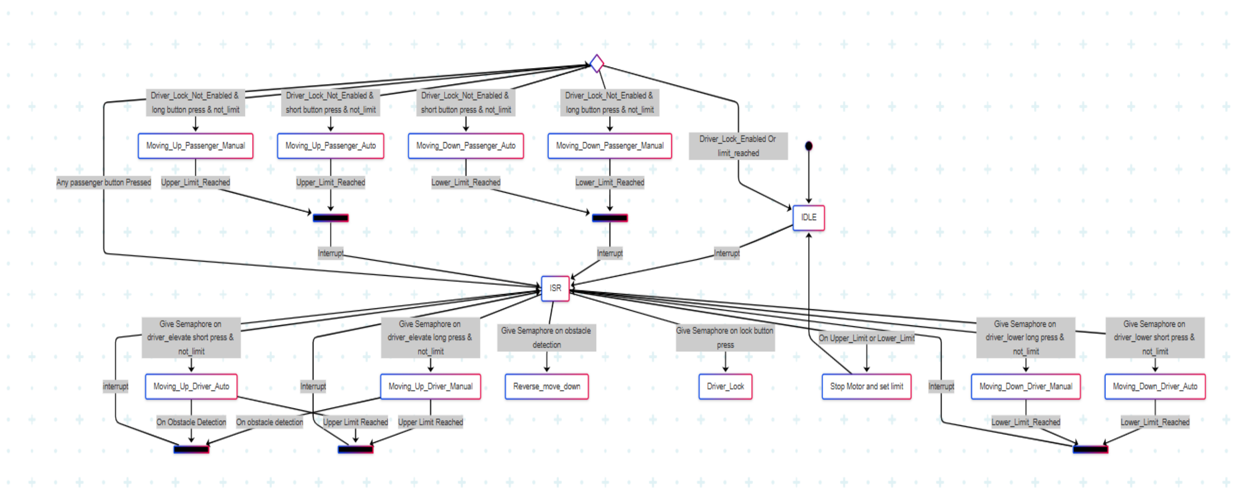
- Obstacle detection

```
            // Update LCD with window status after reversal
            LCD_Clear();
            LCD_SetCursor(0, 0);
            LCD_Print("Safety Activated");
            LCD_SetCursor(1, 0);
            LCD_Print("Window: Middle");
        }
        else {
```

- Encoder position updates

```
// Display updated position and direction info
LCD_SetCursor(1,0);
{
    char buf[16];
    snprintf(buf, sizeof(buf), "Pos:%21d Dir:%21d", encoderPos, encoderDirection);
    LCD_Print(buf);
}
```

3.3 System State Diagram



4 Free RTOS Implementation

The power window system uses a multi-task architecture implemented with FreeRTOS, dividing functionality into specialized tasks with appropriate priorities.

4.1 Task Priorities

Tasks are assigned priorities based on their importance to system and safety.

```
//creatubg tasks
xTaskCreate(vPUpTask        , "PassengerWindowElevate"   , 150, NULL, 1, &xPassengerWindowElevateTaskHandle);
xTaskCreate(vPLowerTask     , "PassengerWindowLower"     , 150, NULL, 1, &xPassengerWindowLowerTaskHandle);
xTaskCreate(vDUpTask        , "DriverWindowElevate"      , 150, NULL, 2, &xDriverWindowElevateTaskHandle);
xTaskCreate(vDLowerTask     , "DriverWindowLower"        , 150, NULL, 2, &xDriverWindowLowerTaskHandle);
xTaskCreate(vLockWindows    , "LockWindows"              , 100, NULL, 2, &xLockWindowsTaskHandle);
xTaskCreate(vUpperLimit     , "UpperLimitAction"         , 100, NULL, 4, &xUpperLimitActionHandle);
xTaskCreate(vLowerLimit     , "LowerLimitAction"         , 100, NULL, 4, &xLowerLimitActionHandle);
xTaskCreate(vODetection     , "ObstacleDetection"        , 100, NULL, 3, &xObsatcleDetectionHandle);
xTaskCreate(vEncoderTask    , "Encoder"                  , 128, NULL, 1, &xEncoderTaskHandle);
```

4.1.1 Critical Tasks

The most important tasks to execute immediately are given the highest priority in our system, which is 4.

- VUpperLimit: Needs to work immediately to mark limit and preempt any window movement tasks.
- VLowerLimit: Needs to work immediately and preempt any window movement tasks.

4.1.2 Important Tasks

- VODetection: Needs to preempt every other task, but doesn’t need to necessarily preempt the limit action tasks as they already stop the window from moving so that is good for evading obstacles. The object detection task can pick off where the limit tasks ended.

4.1.3 Lower importance tasks

Priorities 2 and 1 are reserved for driver and passenger control tasks. The driver tasks are higher priority than the passenger tasks so the driver gets to have more authority. They all get preempted by the other safety critical tasks.

4.2 Task Synchronization

The system uses binary semaphores as the primary mechanism for task synchronization:

- Each task has a dedicated semaphore
- Semaphores are given from ISRs when a relevant event occurs
- Tasks block indefinitely (portMAX_DELAY) waiting for their semaphore
- Task notification mechanism is used for encoder position updates

```
//semaphores
xDriverWindowElevateTaskUnlockerSemaphore         = xSemaphoreCreateBinary();
xDriverWindowLowerTaskUnlockerSemaphore           = xSemaphoreCreateBinary();
xPassengerWindowElevateTaskUnlockerSemaphore      = xSemaphoreCreateBinary();
xPassengerWindowLowerTaskUnlockerSemaphore        = xSemaphoreCreateBinary();
xLockWindowsTaskUnlockerSemaphore                 = xSemaphoreCreateBinary();
xObstacleDetectionUnlockerSemaphore               = xSemaphoreCreateBinary();
xLowerLimitUnlockerSemaphore                      = xSemaphoreCreateBinary();
xUpperLimitUnlockerSemaphore                      = xSemaphoreCreateBinary();
```

4.3 Interrupt Service Routine Integration

The system employs a unified interrupt service routine (ISR) approach with clear handling of different interrupt source.

4.3.1 Passenger Controls

```
{
  BaseType_t xHigherPriorityTaskWoken = pdFALSE;
  // WE ALWAYS CLEAR THE INTERRUPT FLAG BEFORE EACH ISR HANDLING

  // if interrupt from passenger up
  if (GPIOIntStatus(GPIO_PORTA_BASE, Passenger_Elevate_Button) == Passenger_Elevate_Button)
  {
    GPIOIntClear(GPIO_PORTA_BASE, GPIO_INT_PIN_2);
    lock_state = GPIOPinRead(Sensors_Port,Window_Lock_Switch);
    if (lock_state == HIGH){
      //
    }
    else if(lock_state == LOW && last_task == STOP){
      xSemaphoreGiveFromISR(xPassengerWindowElevateTaskUnlockerSemaphore, &xHigherPriorityTaskWoken);
      portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
    }
    else if(lock_state == LOW && (last_task == PU || last_task == DU || last_task == DD || last_task == PD)){
      // Disable the motor first
      GPIO_PORTF_DATA_R &= ~DC_Motor_Enable;
      // Then clear direction pins
      GPIO_PORTA_DATA_R &= ~DC_Motor_In1;
      GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
      last_task = STOP;
    }
  }
```

```c
// int from pass lower
else if (GPIOIntStatus(GPIO_PORTA_BASE, Passenger_Lower_Button) == Passenger_Lower_Button)
{
  GPIOIntClear(GPIO_PORTA_BASE, GPIO_INT_PIN_3);
  lock_state = GPIOPinRead(Sensors_Port,Window_Lock_Switch);

  if (lock_state == HIGH){
    //
  }
  else if(lock_state == LOW && last_task == STOP){
    xSemaphoreGiveFromISR(xPassengerWindowLowerTaskUnlockerSemaphore, &xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
  }
  else if(lock_state == LOW && (last_task == PU || last_task == DU || last_task == DD || last_task == PD)){
    // Disable the motor first
    GPIO_PORTF_DATA_R &= ~DC_Motor_Enable;
    // Then clear direction pins
    GPIO_PORTA_DATA_R &= ~DC_Motor_In1;
    GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
    last_task = STOP;
  }
}
```

If the interrupt comes from passenger buttons, the isr checks whether the driver lock is on or off. If lock is enabled no action will be taken, otherwise it gives the semaphore and passenger elevate or lower task will be woken.

4.3.2 Driver Controls

Work the same as passenger controls but without the check of the lock state.

```c

// int from driver up
else if (GPIOIntStatus(GPIO_PORTA_BASE, Driver_Elevate_Button) == Driver_Elevate_Button)
{
  GPIOIntClear(GPIO_PORTA_BASE, GPIO_INT_PIN_4);
  if(last_task == STOP){
    xSemaphoreGiveFromISR(xDriverWindowElevateTaskUnlockerSemaphore, &xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
  }
  else if(last_task == PU || last_task == DU || last_task == DD || last_task == PD){
    // Disable the motor first
    GPIO_PORTF_DATA_R &= ~DC_Motor_Enable;
    // Then clear direction pins
    GPIO_PORTA_DATA_R &= ~DC_Motor_In1;
    GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
    last_task = STOP;
  }
}


// int from driver lower
else if (GPIOIntStatus(GPIO_PORTA_BASE, Driver_Lower_Button) == Driver_Lower_Button)
{
  GPIOIntClear(GPIO_PORTA_BASE, GPIO_INT_PIN_5);
  if(last_task == STOP){
    xSemaphoreGiveFromISR(xDriverWindowLowerTaskUnlockerSemaphore, &xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
  }
  else if(last_task == PU || last_task == DU || last_task == DD || last_task == PD){
    // Disable the motor first
    GPIO_PORTF_DATA_R &= ~DC_Motor_Enable;
    // Then clear direction pins
    GPIO_PORTA_DATA_R &= ~DC_Motor_In1;
    GPIO_PORTA_DATA_R &= ~DC_Motor_In2;
    last_task = STOP;
  }
}

```

4.3.3 Safety Interrupts

```c

// int from passing object
else if (GPIOIntStatus(GPIO_PORTC_BASE, Object_Detection_Sensor) == Object_Detection_Sensor)
{
  GPIOIntClear(GPIO_PORTC_BASE, GPIO_INT_PIN_4);
  xSemaphoreGiveFromISR(xObstacleDetectionUnlockerSemaphore, &xHigherPriorityTaskWoken);
  portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}


// int from lower limit
else if (GPIOIntStatus(GPIO_PORTC_BASE, Window_Lower_Limit) == Window_Lower_Limit)
{
  GPIOIntClear(GPIO_PORTC_BASE, GPIO_INT_PIN_6);
  xSemaphoreGiveFromISR(xLowerLimitUnlockerSemaphore, &xHigherPriorityTaskWoken);
  portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}


// int from upper limit
else if (GPIOIntStatus(GPIO_PORTC_BASE, Window_Upper_Limit) == Window_Upper_Limit)
{
  GPIOIntClear(GPIO_PORTC_BASE, GPIO_INT_PIN_5);
  xSemaphoreGiveFromISR(xUpperLimitUnlockerSemaphore, &xHigherPriorityTaskWoken);
  portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}


// int from lock switch
else if (GPIOIntStatus(GPIO_PORTC_BASE, Window_Lock_Switch) == Window_Lock_Switch)
{
  GPIOIntClear(GPIO_PORTC_BASE, GPIO_INT_PIN_7);
  xSemaphoreGiveFromISR(xLockWindowsTaskUnlockerSemaphore, &xHigherPriorityTaskWoken);
  portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}

```

Depending on where the interrupt came from, the ISR wakes the task needed to handle the interrupt by giving the semaphore.

4.3.4 Encoder interrupt

```c

    // int from encoder
    else if(GPIOIntStatus(GPIO_PORTD_BASE, ENC_PIN_A | ENC_PIN_B))
    {
        BaseType_t woken = pdFALSE;
        uint32_t status = GPIOIntStatus(GPIO_PORTD_BASE, true);
        GPIOIntClear(GPIO_PORTD_BASE, status);
        uint8_t pinA = (GPIO_PORTD_DATA_R & ENC_PIN_A) ? 1 : 0;
        uint8_t pinB = (GPIO_PORTD_DATA_R & ENC_PIN_B) ? 1 : 0;
        uint32_t currentTime = xTaskGetTickCount();
        if ((currentTime - lastEncoderTime) < ENC_DEBOUNCE_MS) {
            return;
        }
        lastEncoderTime = currentTime;
        //check what pin changed
        if (status & ENC_PIN_A) {        // Pin A changed
            if (pinA == pinB) {          // If A and B are the same
                encoderPos--;
                encoderDirection = -1;
            } else {
                encoderPos++;
                encoderDirection = 1;
            }
        }
        else if (status & ENC_PIN_B) {  // Pin B changed
            if (pinA != pinB) {          // If A and B are different
                encoderPos--;
                encoderDirection = -1;
            } else {
                encoderPos++;
                encoderDirection = 1;
            }
        }
        if (encoderPos < ENC_MIN_POS) encoderPos = ENC_MIN_POS;
        if (encoderPos > ENC_MAX_POS) encoderPos = ENC_MAX_POS;

        xTaskNotifyFromISR(xEncoderTaskHandle, 0, eNoAction, &woken);
        portYIELD_FROM_ISR(woken);
    }
}
```

On interrupt from the encoder, encoder position and direction are updated and encoder task is notified.

5 Challenges Faced

5.1 Faulty Power Supply

One of the challenges encountered during the development of the system was an unreliable power supply. Initially, the system had inconsistent behavior whenever the motor was connected like random resets or unresponsive components. We began troubleshooting by thoroughly inspecting the code and verifying the wiring to rule out any logical or connection errors. We then identified the issue as faulty power supply. When we connected a better power supply the system started working as intended.

5.2 Debounce

The mechanical switches and buttons produce multiple rapid transitions when pressed or released.

Solution:

- Software debounce using task delays
- Timestamp-based debounce for encoder signals to filter out electrical noise
- ISR debounce protection prevents multiple rapid triggering

6 Future Improvements

6.1 Better User Experience

- Allow user to control elevating or lowering speed.
- Customizable behaviors.

6.2 Remote Control

Allow user to control window using mobile app or car key.

6.3 System Optimization

Optimize execution time for tasks.