Power Window Control System

Using Tiva C & FreeRTOS

Team 17:

- Anas Salah Abdelrazek 19P9033
- Alaa Mohamed Hamdy 19P6621
- Nada Amr Attia 19P1621
- Ahmed Amr Mohyeldin Elgayar 19P8349
- Mohamed Mahmoud Hussein Ahmed 19P7975

Agenda

- Team Members Contribution
- GitHub & video Link
- Project Requirements
- Project Approach
- Overview on How our system Works
- State Diagrams
- Circuit Topology
- Handling Different Cases Using Timing Diagrams
- Uses of Semaphores , Queues and Mutex

Team Members Contribution

Task Assigned	Members
Planning Project Approach	All members
LCD	Nada , Alaa
Button Task	Ahmed , Anas
Driver Tasks	Mohamed , Nada
Passenger Tasks	Anas , Alaa
Sporadic Tasks	Ahmed
Hardware Connections	Mohamed , Anas
Drivers	Mohamed
Testing	All members

GitHub And Video Link

- GitHub Repository : https://github.com/anassalah24/Power-Window-Control-System-Using-Tiva-C-And-FreeRTOS.git

(code Contains Detailed Comments)

- Video Link: https://drive.google.com/file/d/19InKF6QktM48Px8oUl6x1lapc-dWLjln/view?usp=sharing

Project Requirements

System basic features

1. Manual open/close function

When the power window switch is pushed or pulled continuously, the window opens or closes until the switch is released.

2. One touch auto open/close function

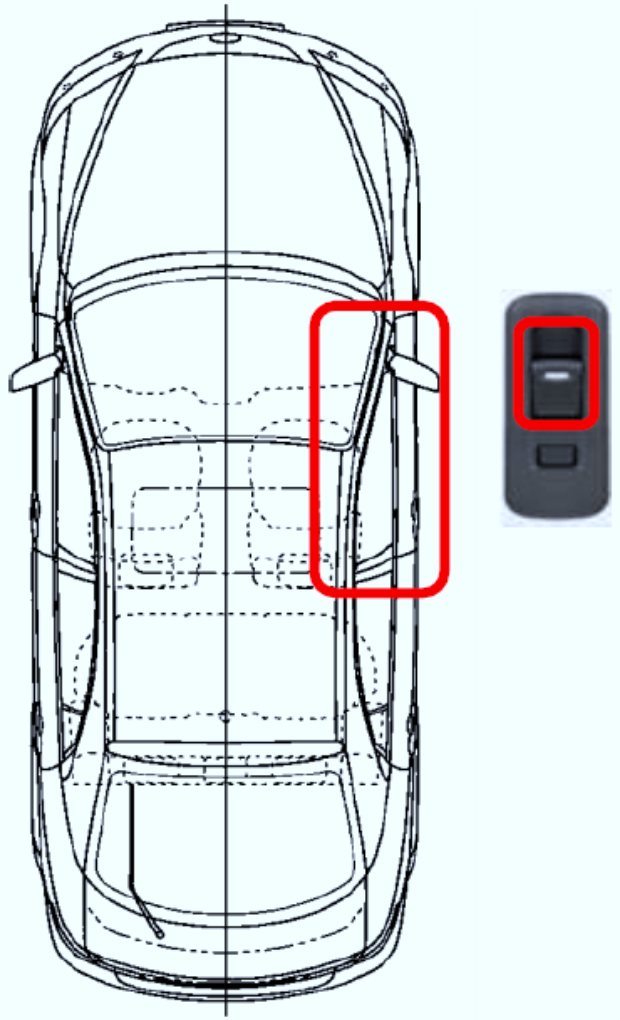
When the power window switch is pushed or pulled shortly, the window fully opens or closes.

3. Window lock function

When the window lock switch is turned on, the opening and closing of all windows except the driver's window is disabled.

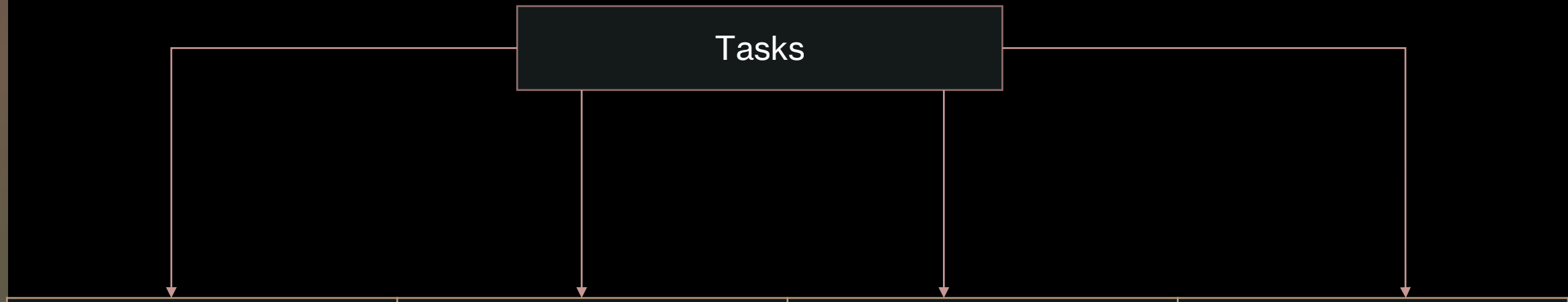
4. Jam protection function

This function automatically stops the power window and moves it downward about 0.5 second if foreign matter gets caught in the window during one touch auto close operation.



Project Approach

- Our Project Contains 4 main Types of Tasks



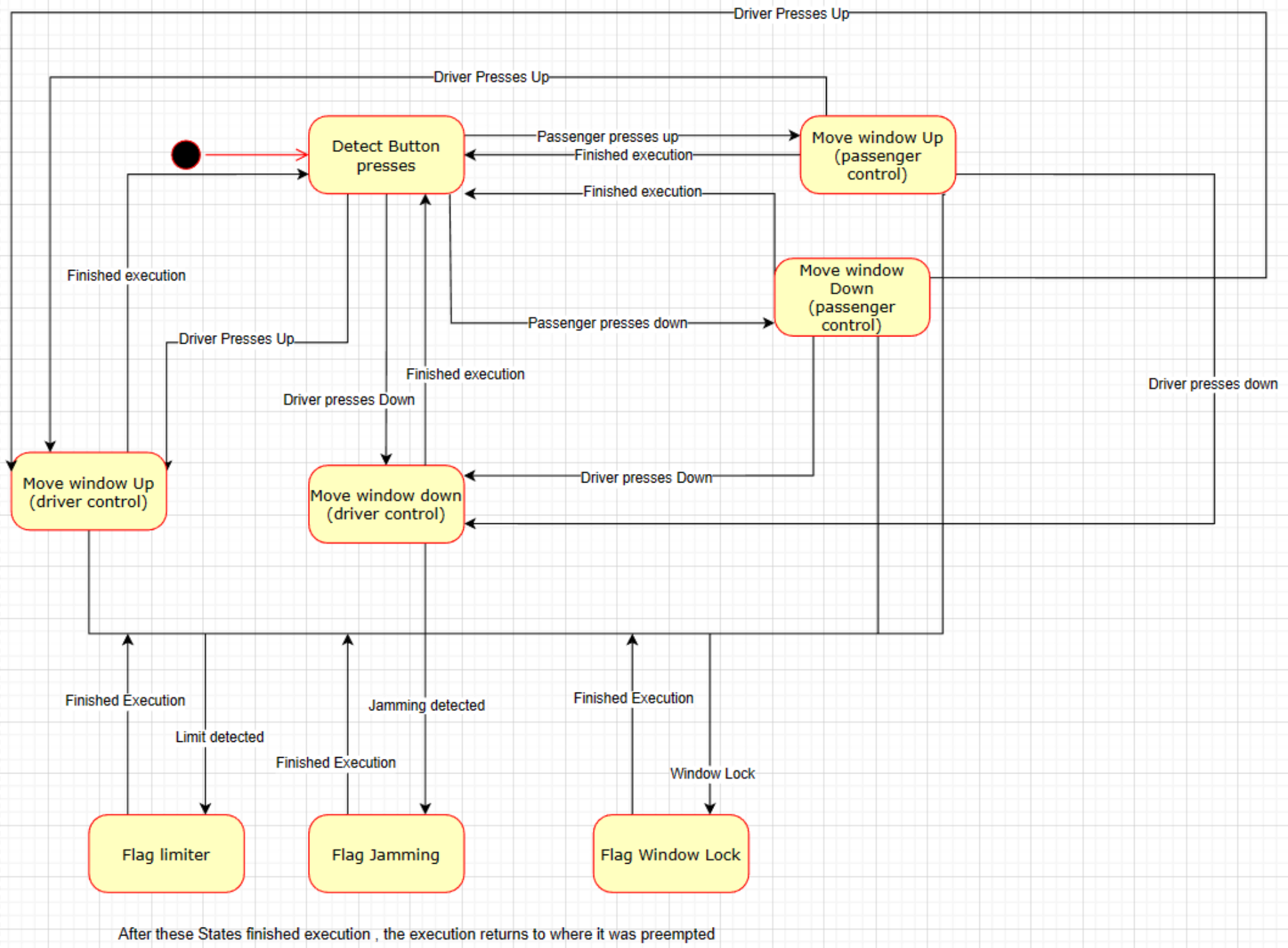
Continuous Button Task	Driver Tasks	Passenger Tasks	Interrupt Handler Tasks
This Task is responsible for detecting which button (driver and passenger UP & Down) was pressed and if it was automatic mode or manual mode	These tasks handles the driver's up and down (automatic or manual) actions for button presses	These tasks handles the passenger's up and down (automatic or manual) actions for button presses	These tasks handles the interrupt button presses such as Jamming , Window Lock , upper limiter and down limiter

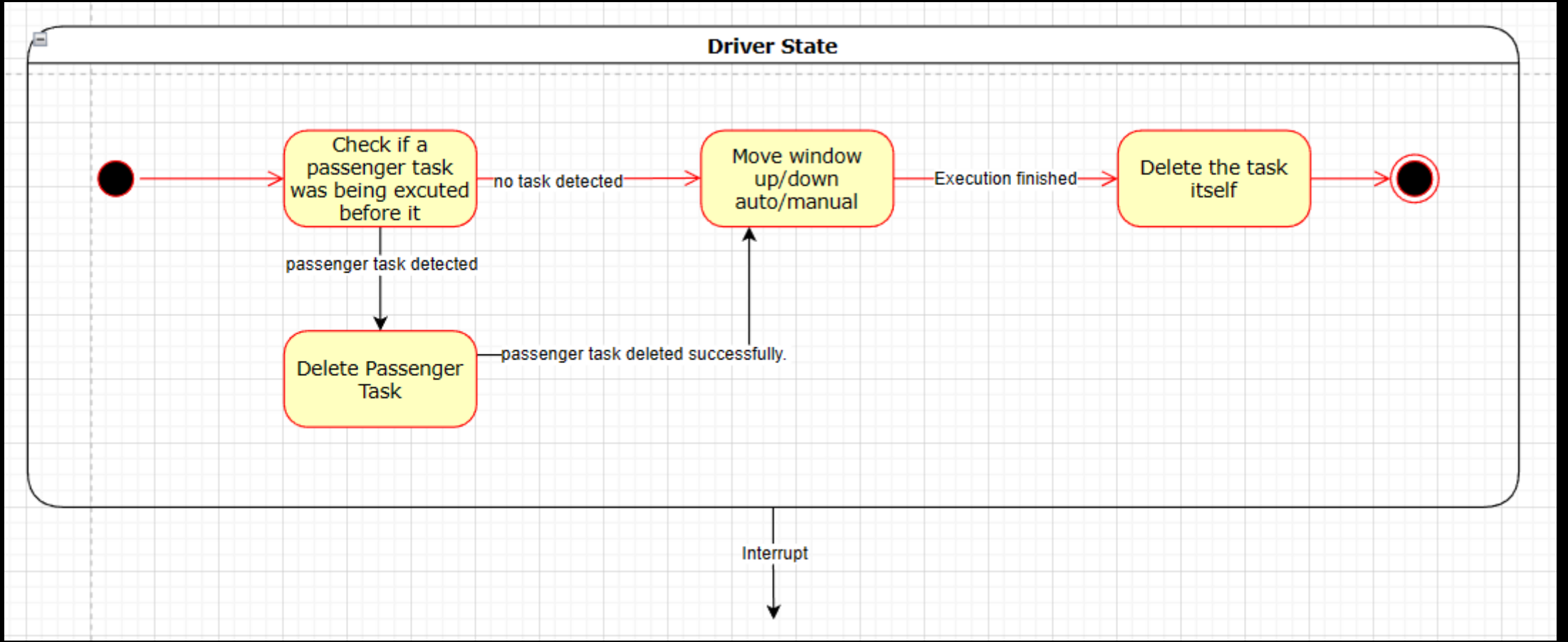
Overview on How our system Works

- Initially a single continuous Task (Button Task) is created with lowest priority (1) and it detects if any up or down button was pressed from either side of the vehicle (Driver or Passenger), This task is never deleted.
- If a button was pressed (say the driver up) a task is created with higher priority (2) which deals with the actions needed to handle the window moving up (either manual or automatic)by the driver , after this task has finished execution, it deletes itself and CPU is returned to the Button Task.(similarly with driver down button and passeneger up and down buttons)
- Similarly, if the passenger presses any button the same thing happens exactly but passengers' task have a little bit extra logic in them which is it to detects if any of the driver's buttons were pressed because if it was (since driver has more power than passenger) we need to switch execution to driver so we create a driver task with higher priority (3) that handles its action but once we start the driver task we delete the passenger task because we don't want to return back to it obviously.
- If during execution of any of the above tasks , an interrupt driven (sporadic task) occur such as Jamming button , Window Lock , Upper and lower limiters were pressed , the above tasks are interrupted and a ISR handler tasks are executed which alters flags of the mentioned interrupt events.

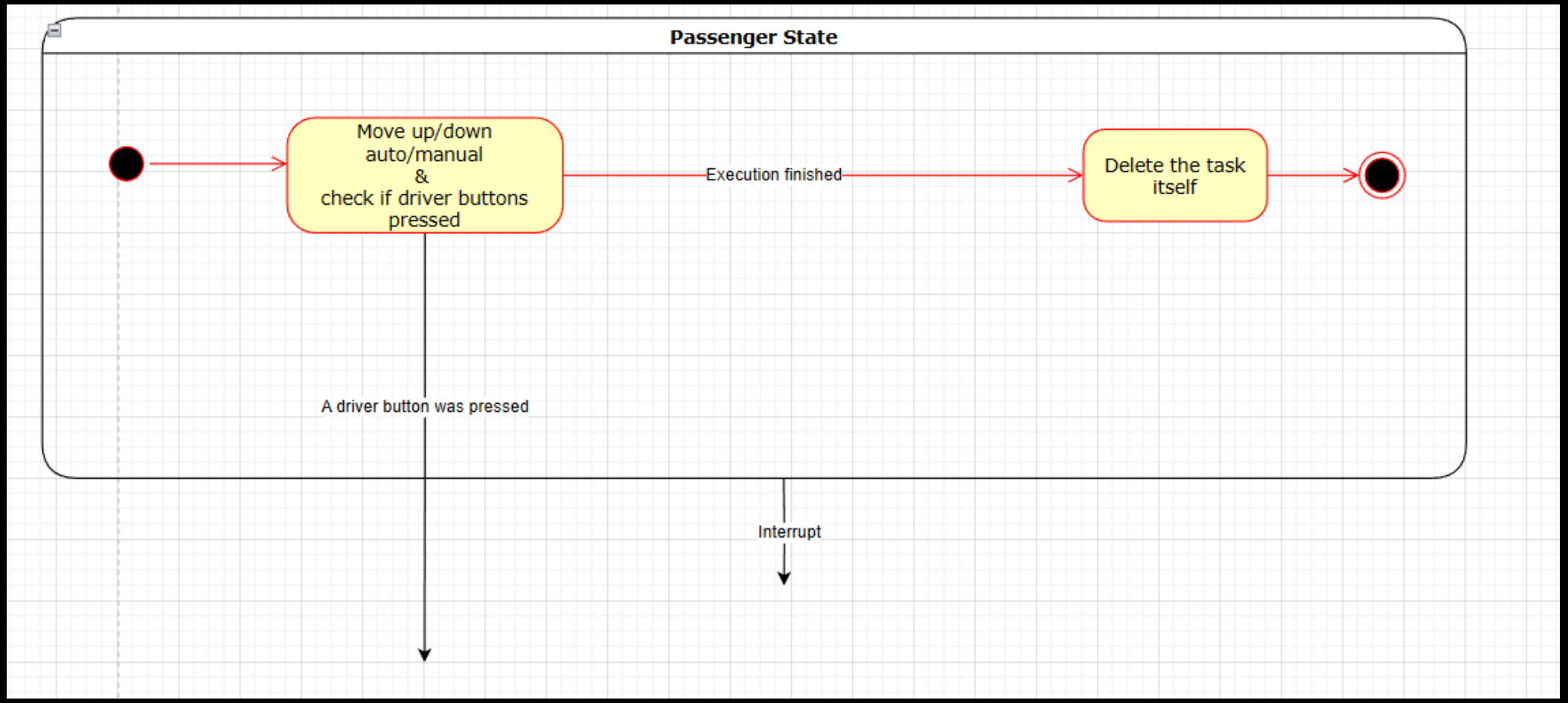
- NOTE : The above logic is visualized in diagrams in the next slides throughout the presentation

Overall System State Diagram





Driver State (container Diagram)



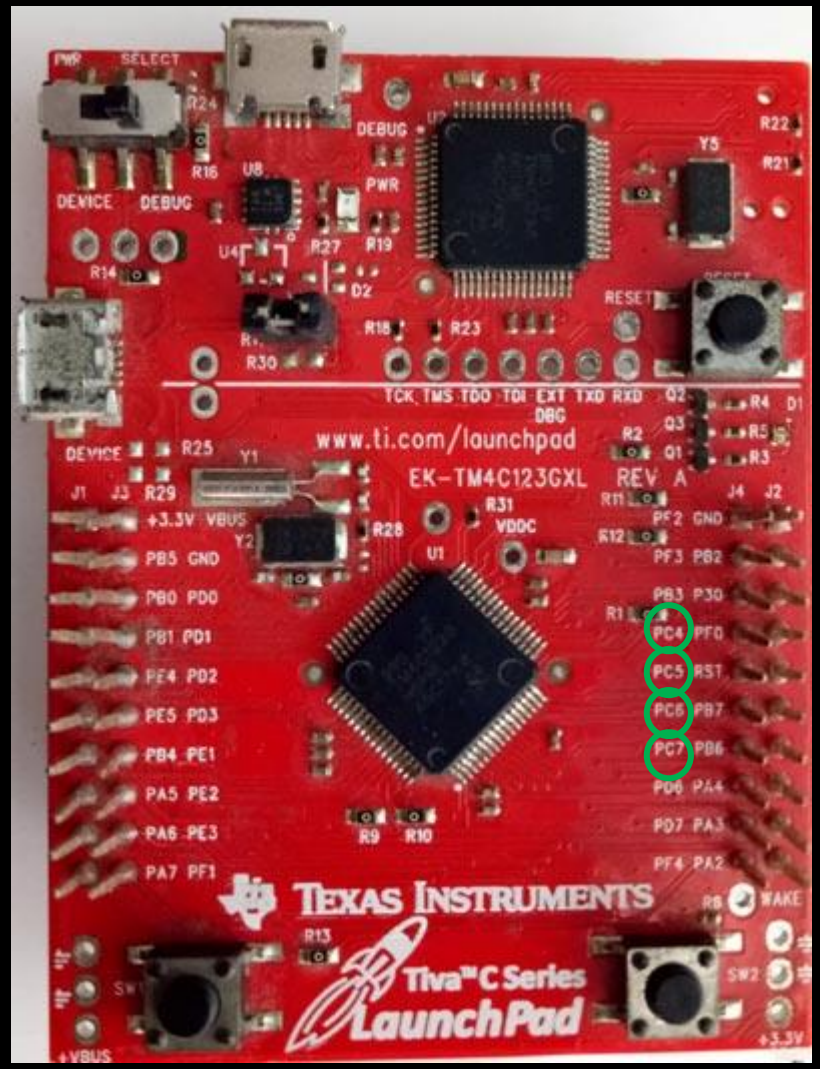
Passenger State (container Diagram)

Circuit Topology

Tiva C Connections

- Window Buttons (Port C)

PC4	Driver Up
PC5	Driver Down
PC6	Passenger Up
PC7	Passenger Down

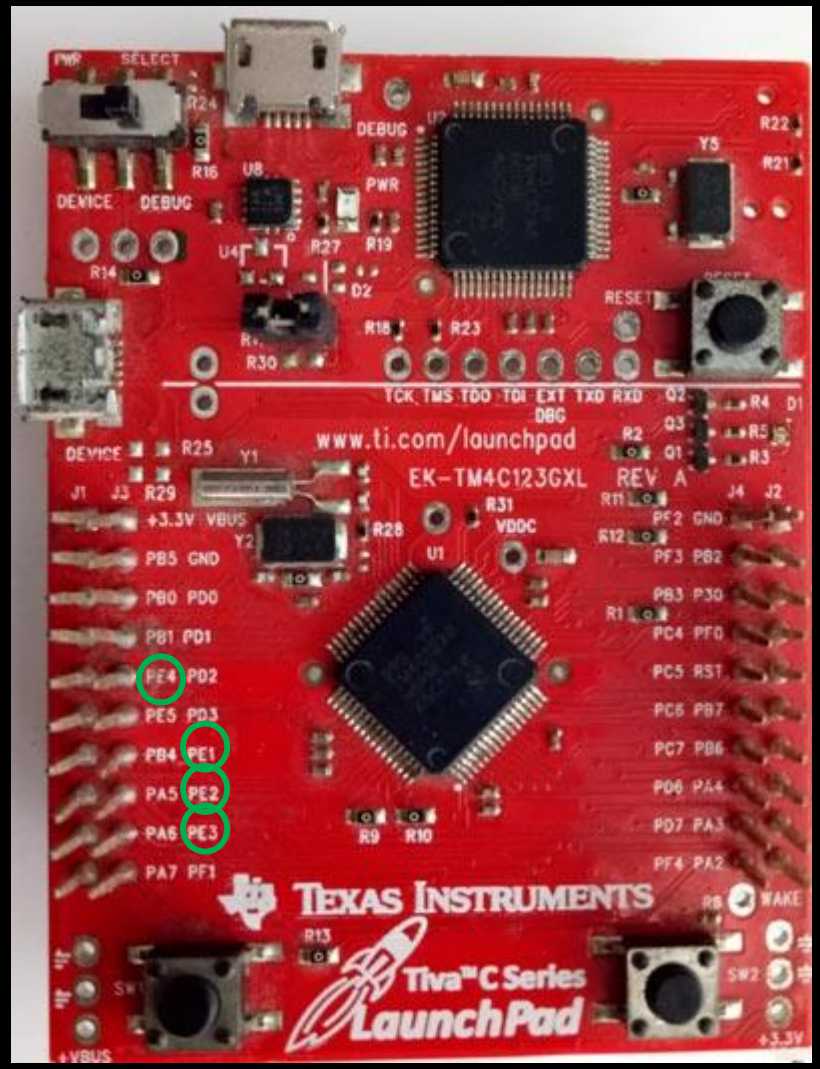


Circuit Topology

Tiva C Connections

- Interrupt Buttons (Port E)

PE1	Upper Limit Switch
PE2	Lower Limit Switch
PE3	Window Lock Push Button
PE4	Jam Detected Push Button

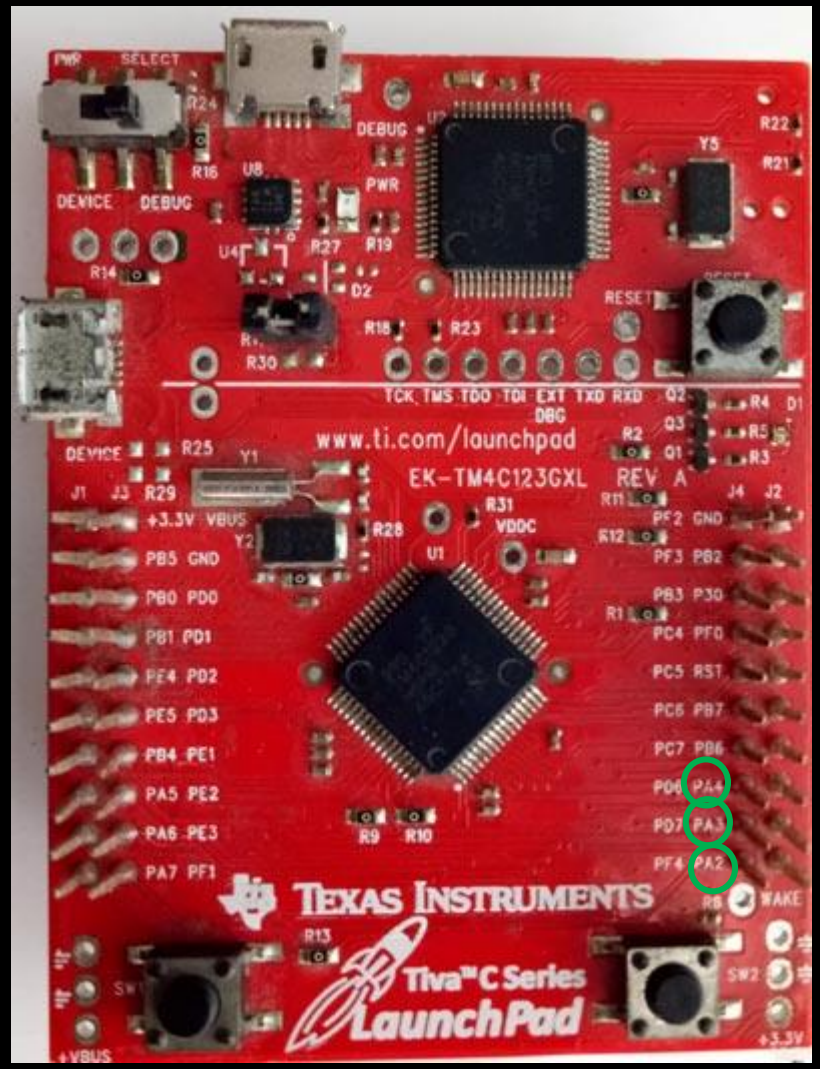


Circuit Topology

Tiva C Connections

- Motor Module (Port A)

PA2	ENA
PA3	IN1
PA4	IN2

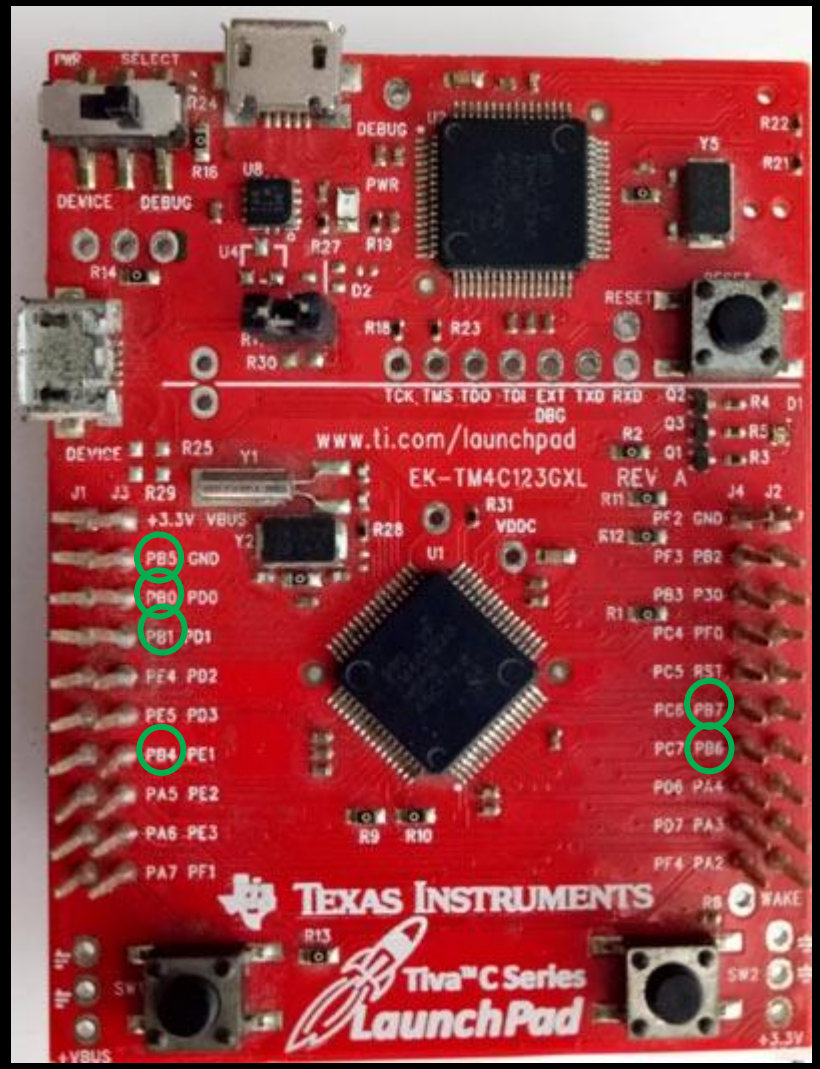


Circuit Topology

Tiva C Connections

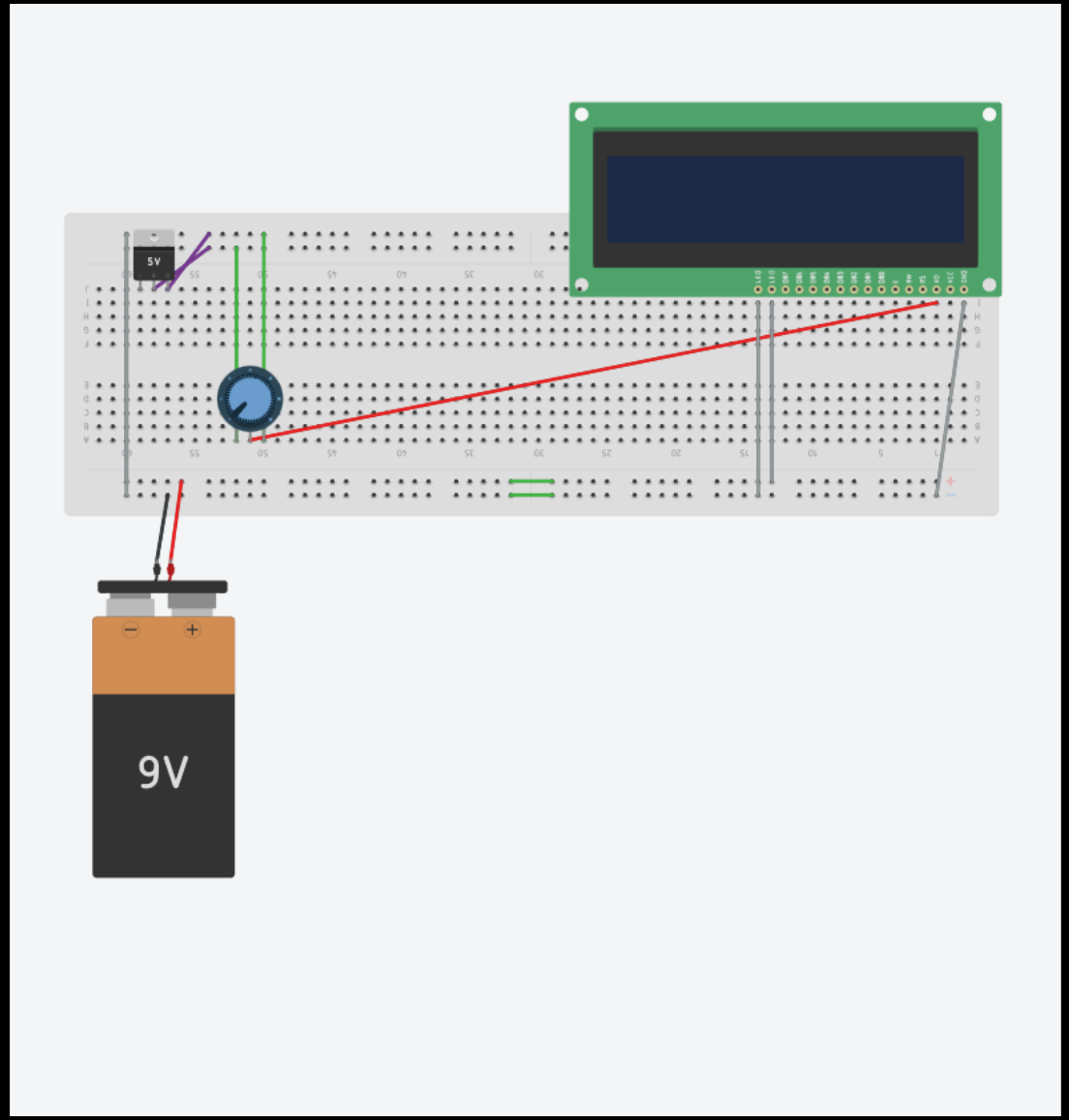
- LCD Module (Port B)

LCD Pins	Stellaris Launchpad Pin
VSS	GND
VDD	VBUS
V0	Potentiometer
RS	PB0
RW	GND
E	PB1
D4	PB4
D5	PB5
D6	PB6
D7	PB7
A	5V
K	GND



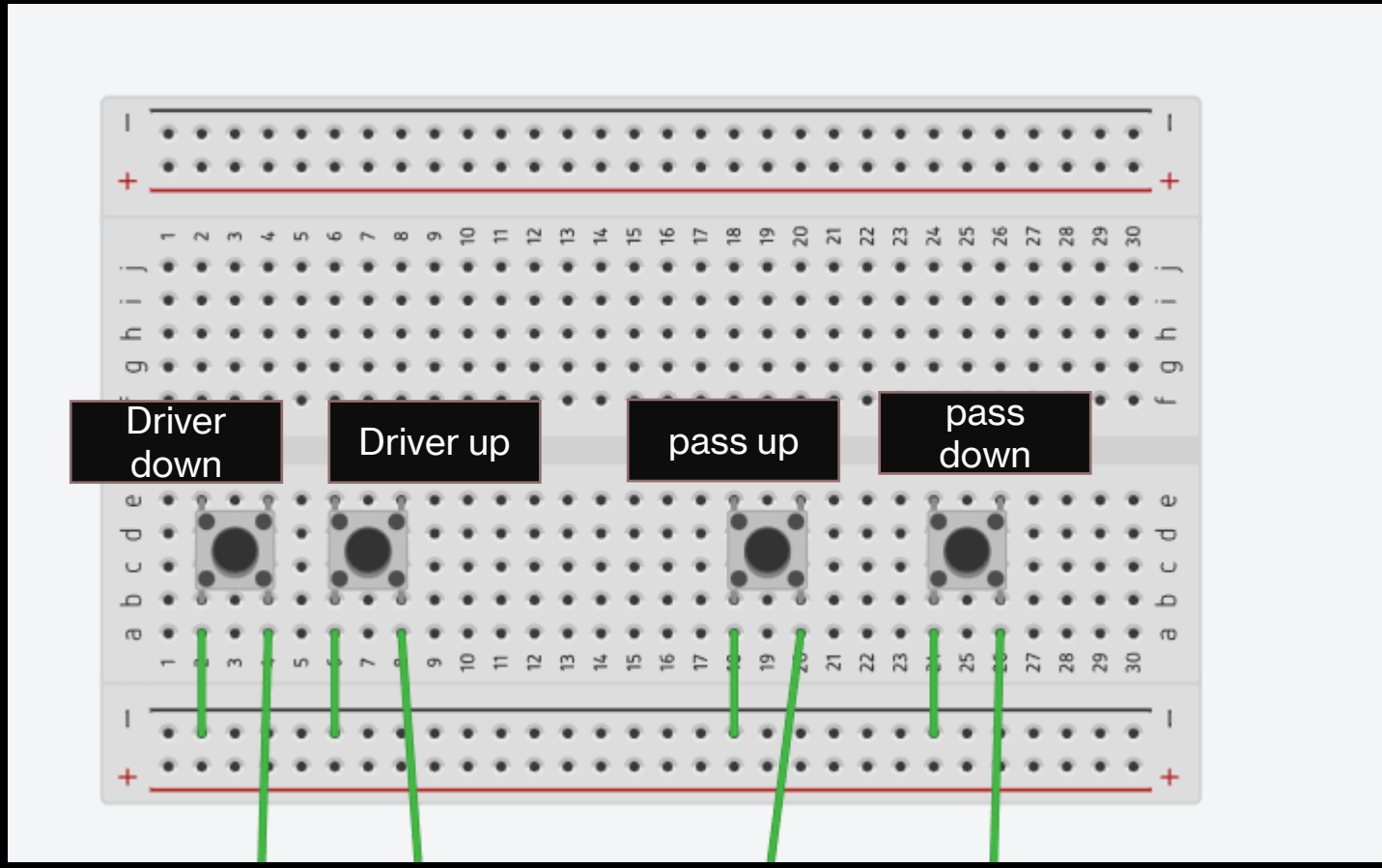
Circuit Topology

LCD Connections
(breadboard 1)



Circuit Topology

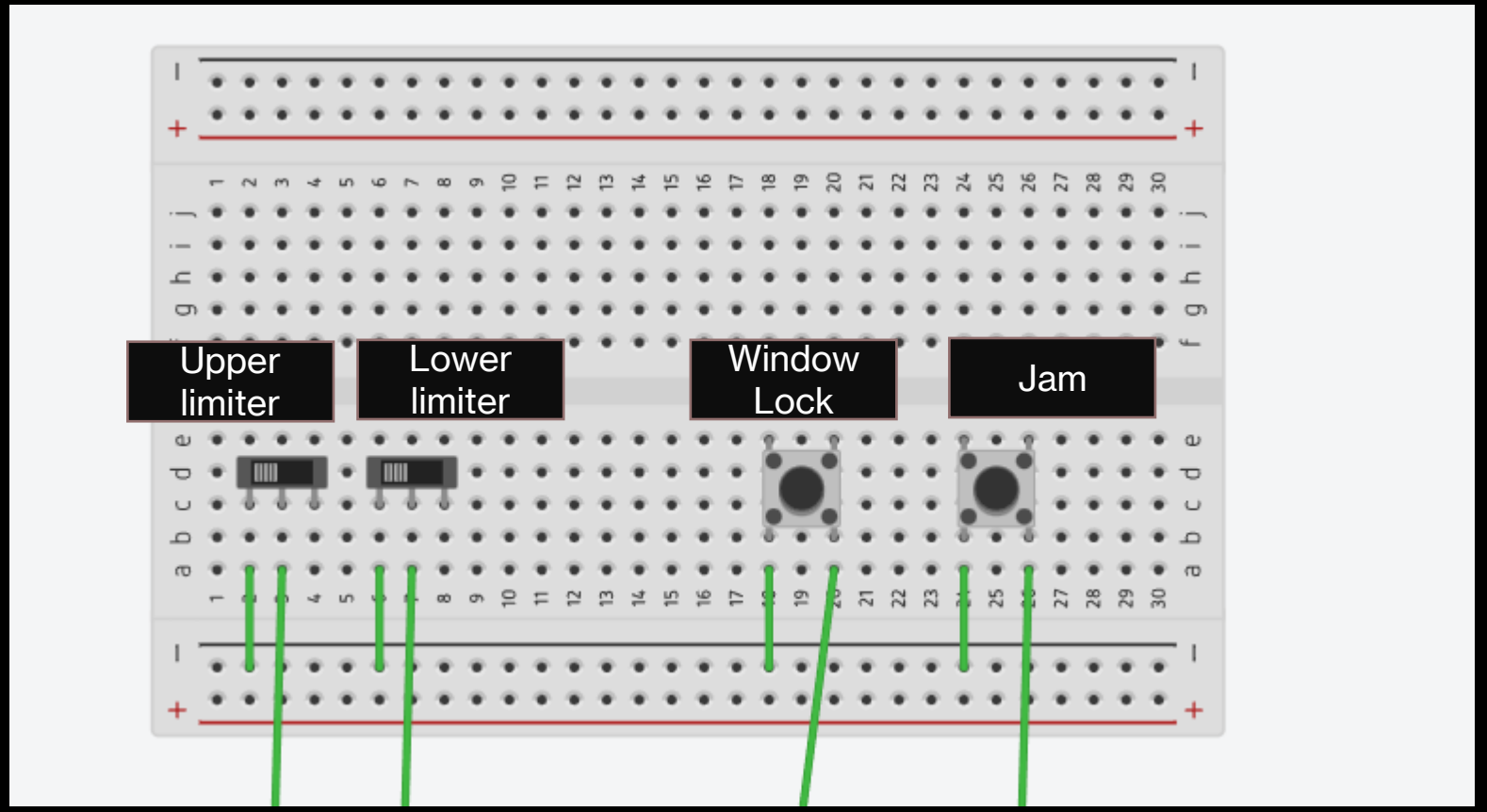
Driver and Passenger buttons Connections (breadboard 2)



Circuit Topology

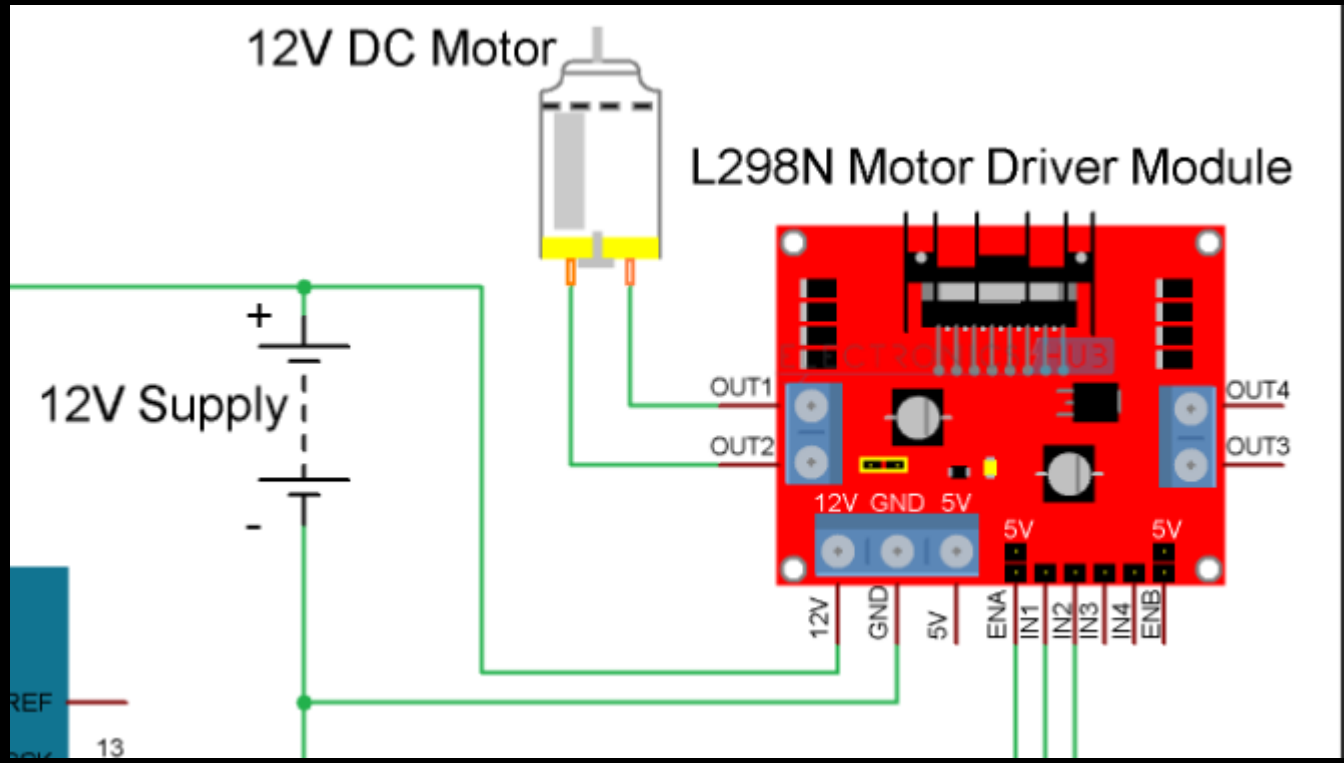
Interrupt Buttons

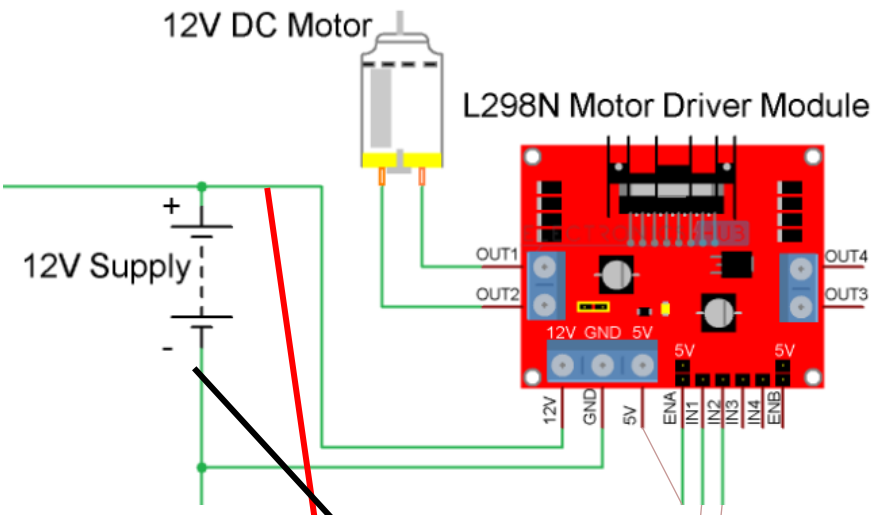
(Jam , Lock , limits)

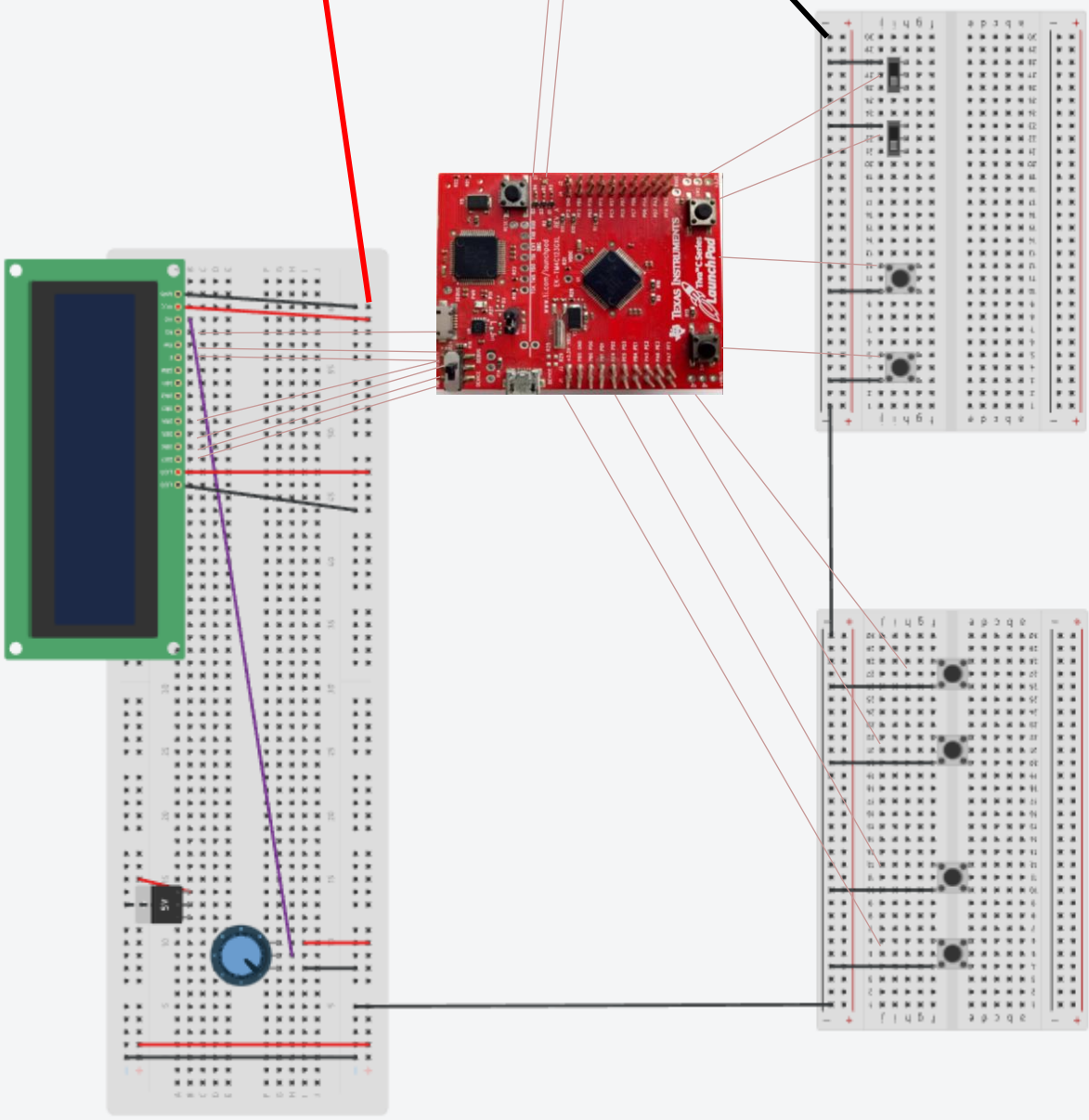


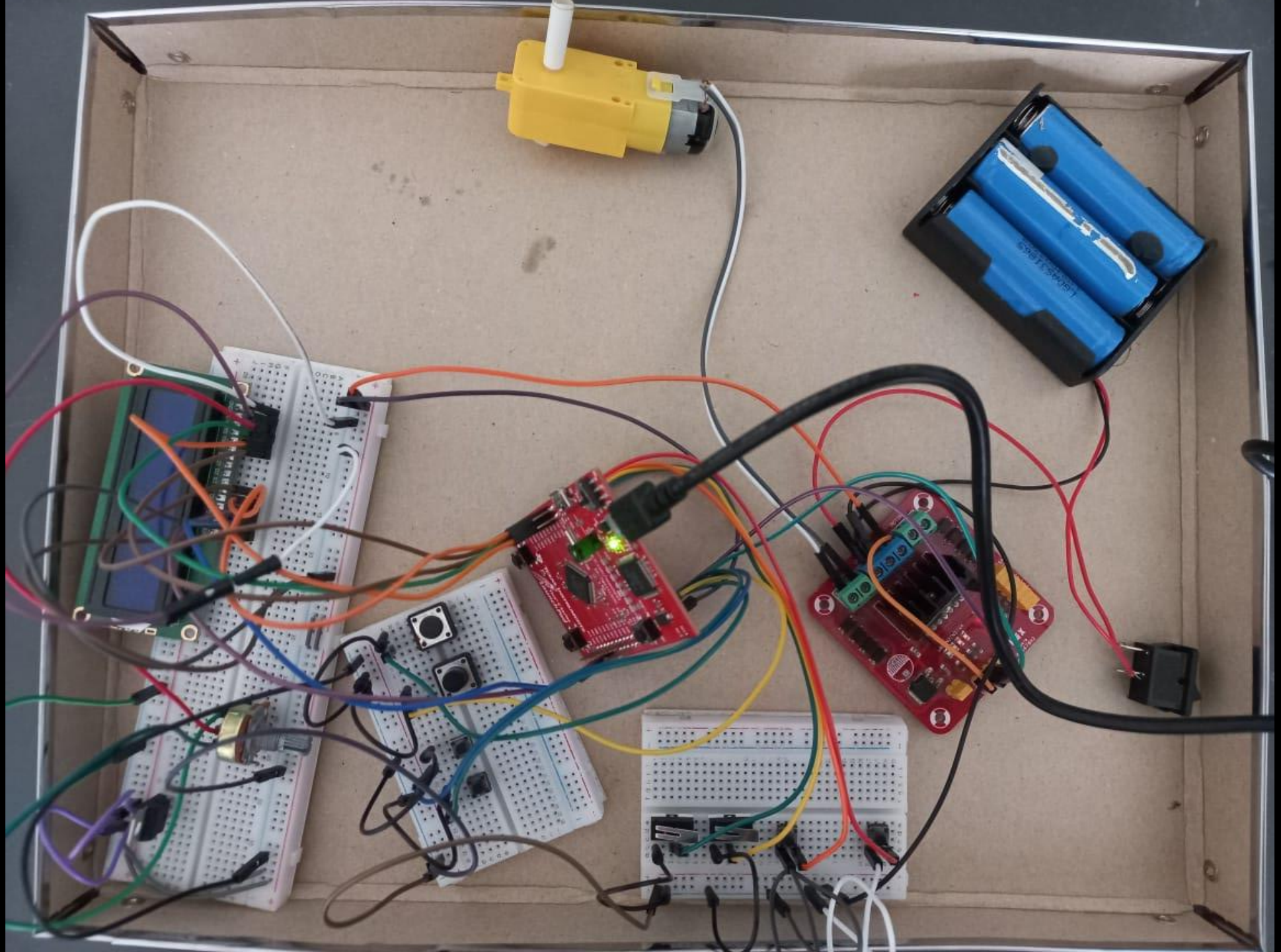
Circuit Topology

L298N Motor Driver Module



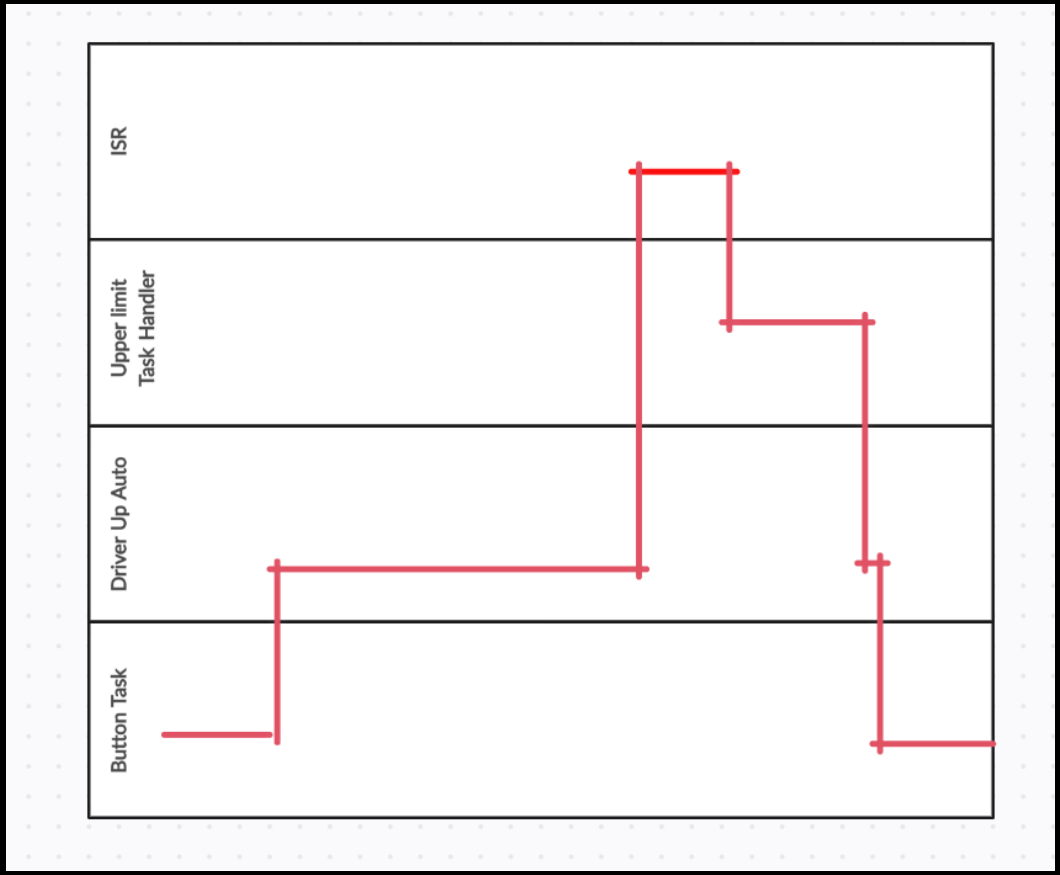

12V DC Motor
L298N Motor Driver Module
12V Supply
+
-
OUT1
OUT2
OUT4
OUT3
12V GND 5V
5V
5V
12V
GND
5V
ENA
IN1
IN2
IN3
IN4
ENB





Handling Different Cases Using Timing diagrams

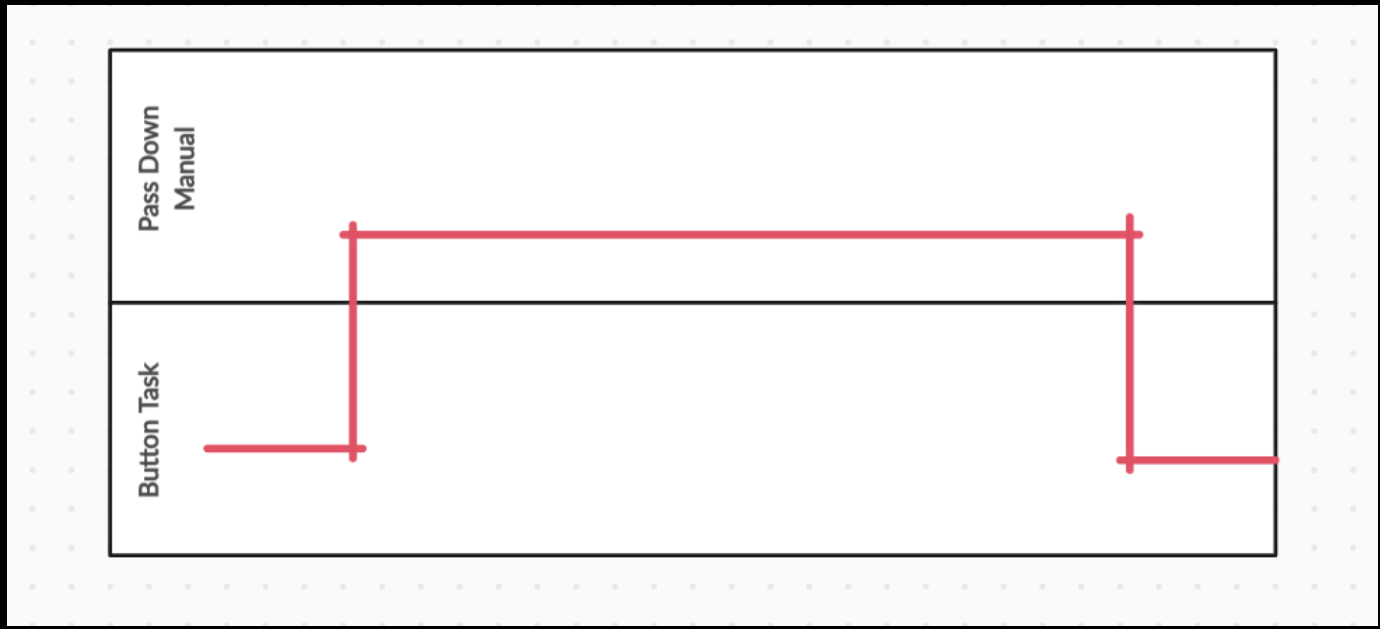
- Action :Driver presses up automatic mode , till upper limit reached



- A similar timing diagram is for any motion of window by any user interrupted by any sporadic task

Handling Different Cases Using Timing diagrams

- Action: Driver / Passenger pressed button down manually until button is released

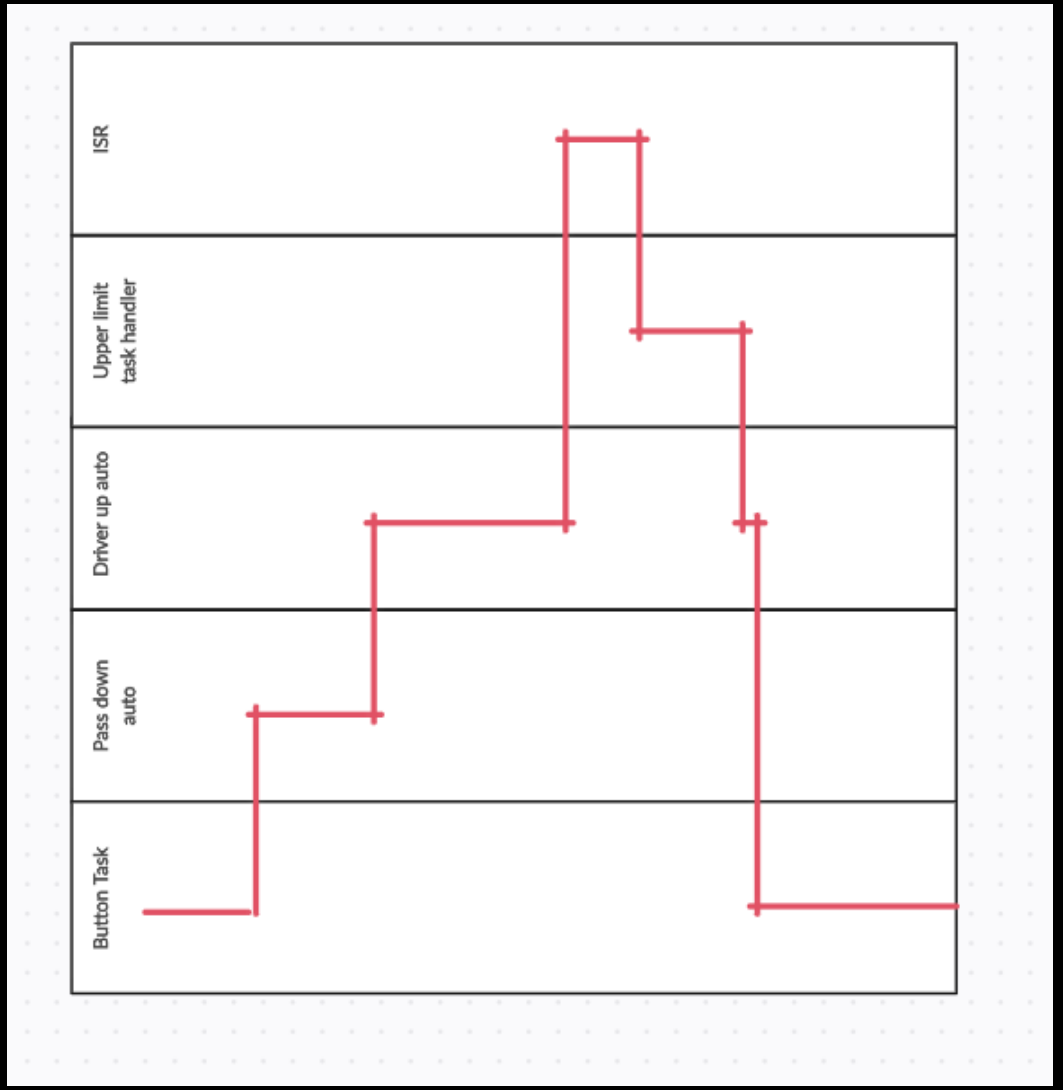


- A similar timing diagram is for any manual motion of window by any user

Handling Different Cases Using Timing diagrams

- Action: Pass down , driver up , upper limit reached

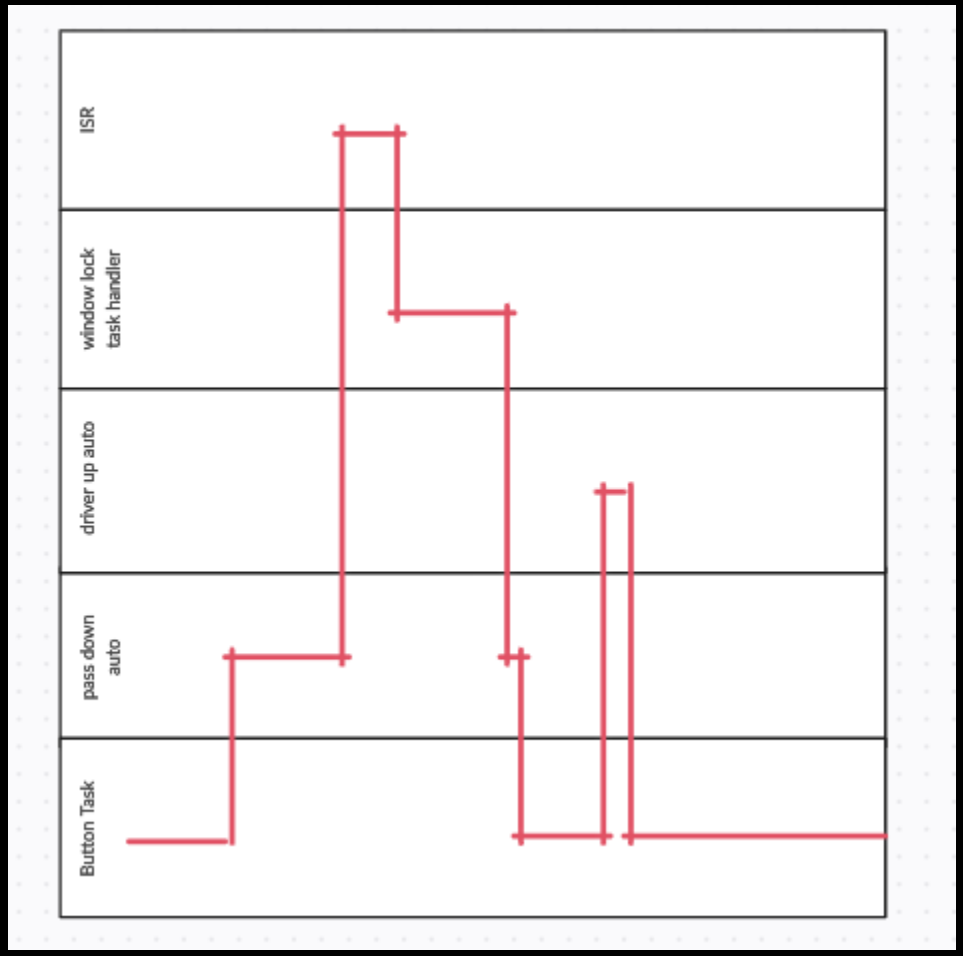
- A similar timing diagram is for any passenger action that it will always be overridden by the driver if he make any action
- Passenger task is deleted when we enter driver task



Handling Different Cases Using Timing diagrams

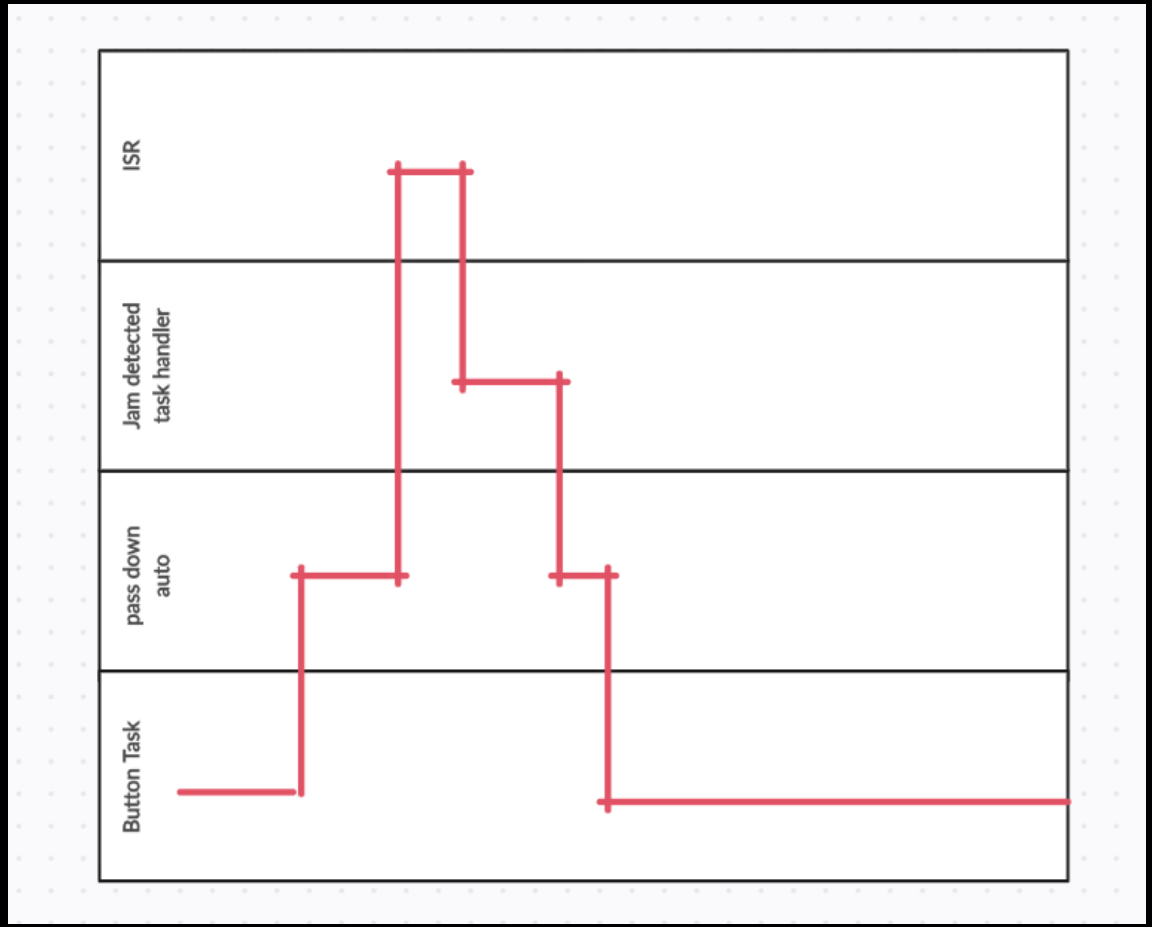
- Action: pass down auto , window locked , driver up auto

- A window lock will lock the window from any action from any user in any direction



Handling Different Cases Using Timing diagrams

- Action: driver up manual , jam detected
- Jam detected will cause the motor to move in opposite direction for 500ms inside the task itself



Use of Semaphores

- We Used binary semaphore to allow sporadic tasks to happen
- Initially these tasks block on taking semaphore
- Whenever we enter the ISR , a semaphore is given which unblocks the appropriate task

```
97
98  //Binarysemaphores needed for (sporadic) tasks that occur because of an interrupt
99  SemaphoreHandle_t xupperLimitReachedSemaphore;
100 SemaphoreHandle_t xlowerLimitReachedSemaphore;
101 SemaphoreHandle_t xwindowLockSemaphore;
102 SemaphoreHandle_t xjamDetectedSemaphore;
103
```

```
    if(interrupt_status & GPIO_PIN_4) // Check if Pin 4 caused the interrupt
    {
        // Pin 4 was pressed (jamming Detected)
        //pass semaphore to appropriate task to unblock it
        xSemaphoreGiveFromISR(xjamDetectedSemaphore, &xHigherPriorityTaskWoken);
    }
    GPIOIntClear(GPIO_PORTE_BASE, interrupt_status); // Clear the interrupt status of the GPIO port E
    portEND_SWITCHING_ISR(xHigherPriorityTaskWoken); //force context switch
}
```

```
1119 void setJamDetected(void *pvParameters)
1120 {
1121
1122   xSemaphoreTake(xjamDetectedSemaphore, 0);
1123   // Wait for the semaphore
1124   while(1){
1125     xSemaphoreTake(xjamDetectedSemaphore, portMAX_DELAY);//semaphore is given in ISR
1126
1127     //change jam detected flag to 1 to indicate jam was detected and initiate jamming protocol
1128     jamDetected = 1;
1129   }
1130
1131 }
```

Use of Queues

- We used queues to allow task to task communication
- So When do we need task communication?
- When a passenger task is preempted by a driver task , we send the passenger task handler in a queue to the driver task , and the driver task then receive the handler and delete the passenger task because we don’t want to return to the passenger task again

Inside Passenger Task

```
        else {
          // PC4 button has been pressed and released almost immediately (Automatic Mode)
          LCD_Clear();
          // retrieve the current tasks's Handle and pass it to the queue so that the current task can be deleted from driver task
          // in orer not to return to it again after finishing driver task
          xTaskHandlePassed = xTaskGetCurrentTaskHandle();
          xQueueSend(xQueue, &xTaskHandlePassed, 0);
          xSemaphoreGive(xDriverUpButtonMutex);//give up mutex
          //create driver task and give higher priority than current task (3) to preempt the current task
          xTaskCreate( drivUpAuto, "driverUpAuto", 128, NULL, 3, &xdrivUpAutoHandle );
          LCD_Clear();//should never reach here
        }
      }
```

Inside Driver Task

```
//check if queue contained a handler for a passeneger task to delete it (Because i dont want to return to a passneger task if i was called form one)
if (xQueueReceive(xQueue, &xReceivedHandle, 0) == pdTRUE) {
    vTaskDelete(xReceivedHandle);//delete passeneger task
}
```

Use of Mutex

- We Used mutex in our project to protect a critical section shared between the BUTTON Task and Any passenger task , which is detecting if a driver button was pressed or not
- Even though it might seem starnge that why would I go back to button task (priority 1) even though I am in a higher priority passenger task (priority 2) but due to vTaskDelay (which blocks) in passenger task , the execution is given to Button task , which can then read also the driver button which will cause faulty readings and unnecessary duplicated actions.
- So we use mutex to protect this critical section (detecting driver's button presses)

Inside Button Task

```
//Take Mutex of driver up button to read its state (blocked if mutex is with another task later on)
if (xSemaphoreTake(xDriverUpButtonMutex, portMAX_DELAY) == pdTRUE){
```

```
        //Release mutex after finished reading the button state
        xSemaphoreGive(xDriverUpButtonMutex);
```

Inside Passenger Tasks

```
//take Mutex to read buttons (here mutex was needed becasue any delays here caused the task to block and if a button was pressed
// the polling task (button task) may read button and so the driver task will be performed twice so we needed mutex to lock polling on button).
//same polling mechanism as the (button Task) with extra steps explained below
if (xSemaphoreTake(xDriverUpButtonMutex, portMAX_DELAY) == pdTRUE){
```

```
        //Release mutex after finished reading the button state
        xSemaphoreGive(xDriverUpButtonMutex);
```


THANK YOU