AUTOMOTIVE POWER-WINDOW IN FREERTOS

Ahmed Gamil Fathy Ibrahim Khalil Rabie **19P4664 : CODE/VIDEO/TESTING**

Ahmed Osama Noaman 19P7926 :CIRCUIT TOPOLOGY/ CODE/GITHUB

Mohamed Ehab Mansour Mohamed 19P5241 :CODE/VIDEO

Maria Mourad Elia Mikhael 19P4894 :TESTING/VIDEO

Sohayla Ihab AbdelMawgoud Ahmed Hamed 19P7343 :DOCUMENTATION/STATE DIAGRAM/TEST

CONTENTS

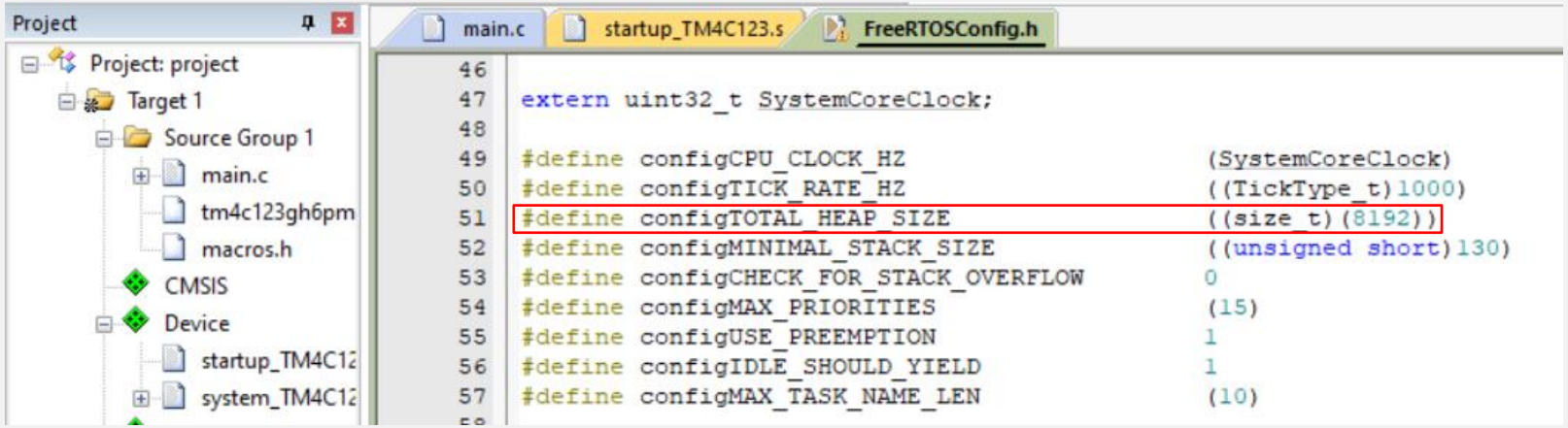
CONTENTS

CODE FUNCTIONALITY: STEPS OF WORKING CODE

1. Code Initialization: Import libraries and check settings
2. Manual Close/Open Function: The window closes/opens only when user stops holding button open/close
3. Automatic Close/Open Function: The window closes/opens only when user presses open/close button for a short duration
4. Window Lock Function: This task allows the window to lock all opening/closing of passenger windows only
5. Jam Function: If an obstacle is detected during auto close, stop and reverse motor direction
6. Queuing Task: This is an imitation of the scheduler that schedules queue elements received with very give/take of a semaphore

CODE INITIALIZATION

```
#include <stdint.h>
#include <string.h>
#include <FreeRTOS.h>
#include <task.h>
#include <queue.h>
#include <semphr.h>
#include "TM4C123GH6PM.h"
#include "macros.h"
#define PortA_IRQn 30
```



MANUAL OPEN/CLOSE FUNCTION: INITIALIZATION

```
void buttonsInit(void) {
//Enable Port D SYSCTL_RCGCGPIO_R |= 0x08;
__asm__("NOP; NOP; NOP; NOP;");
//Configure Pin 0 -> 3 in Port D as input
GPIO_PORTD_DIR_R &= ~((1 << 0)|(1<<1)|(1<<2)|(1<<3));
GPIO_PORTD_CR_R |= (1 << 0)|(1<<1)|(1<<2)|(1<<3);
GPIO_PORTD_PUR_R |= (1 << 0)|(1<<1)|(1<<2)|(1<<3);
GPIO_PORTD_DEN_R |= (1 << 0)|(1<<1)|(1<<2)|(1<<3); }
```

MANUAL OPEN/CLOSE FUNCTION: INITIALIZATION

```
void limitInit(void) {
SYSCTL_RCGCGPIO_R |= 0x08;
__asm__("NOP; NOP; NOP; NOP;");
//Configure Pins FOR limit switches in Port D and A as inputs
GPIO_PORTD_DIR_R &= ~(1 << 6);
GPIO_PORTD_CR_R |= (1 << 6);
GPIO_PORTD_PUR_R |= (1 << 6);
GPIO_PORTD_DEN_R |= (1 << 6);
GPIO_PORTA_DIR_R &= ~(1 << 7);
GPIO_PORTA_CR_R |= (1<<7);
GPIO_PORTA_PUR_R |= (1<<7);
GPIO_PORTA_DEN_R |= (1<<7); }
```

MANUAL CLOSE FUNCTION

```
void driver(void* pvParameters){
  int Val, state;
  portBASE_TYPE xStatus; //Val= (int) pvParameters;
  while(1){
   xSemaphoreTake(xMutex,portMAX_DELAY );  //Always ready to take resource(mutual exclusion)
   if (GET_BIT(GPIO_PORTD_DATA_R,0)==0){ //pullup closed, check data register
   Val=2;
   xStatus = xQueueSendToBack(xQueue,&Val,0); //Queues in main so that OS can mark what is the ongoing task
   vTaskDelay(1000);
   if (GET_BIT(GPIO_PORTD_DATA_R,0)==0) //still pressing then it is manual
  {
    while(GET_BIT(GPIO_PORTD_DATA_R,0)==0); }
  //Automatic Close
  }
```

MANUAL CLOSE FUNCTION

```
void passenger(void* pvParameters){
 int Val;
 portBASE_TYPE xStatus;
 while(1) {
  xSemaphoreTake(xMutex,portMAX_DELAY );
  if (GET_BIT(GPIO_PORTD_DATA_R,2)==0){ //pullup close
   Val=2; xStatus = xQueueSendToBack(xQueue,&Val,0);
   vTaskDelay(1000);
   if (GET_BIT(GPIO_PORTD_DATA_R,2)==0) //still pressing then it is manual {
     while(GET_BIT(GPIO_PORTD_DATA_R,2)==0); }
//Automatic Close
 Val=1; xStatus = xQueueSendToBack(xQueue,&Val,0);
}//END OD WHILE
```

MANUAL OPEN FUNCTION

```
void driver(void* pvParameters){
//Automatic Close }
if (GET_BIT(GPIO_PORTD_DATA_R,1)==0){ //pullup open
 Val=3;
 xStatus = xQueueSendToBack(xQueue,&Val,0);
 vTaskDelay(1000);
 if (GET_BIT(GPIO_PORTD_DATA_R,1)==0) //still pressing then it is manual
 { while(GET_BIT(GPIO_PORTD_DATA_R,1)==0);
 }
 //Automatic Open
 Val=1; xStatus = xQueueSendToBack(xQueue,&Val,0); //Turn off motor
}//Lock Function(ONLY DRIVER CONTROLS IT) has a special condition that can work when either window is open or window is closed
}//END OF WHILE  }//END OF FUNCTION
```

MANUAL OPEN FUNCTION

```
void passenger(void* pvParameters){
//Automatic Close }
 if (GET_BIT(GPIO_PORTD_DATA_R,3)==0){ //pullup open
 Val=3;
 xStatus = xQueueSendToBack(xQueue,&Val,0);
 vTaskDelay(1000);
 if (GET_BIT(GPIO_PORTD_DATA_R,3)==0) //still pressing then it is manual {
        while(GET_BIT(GPIO_PORTD_DATA_R,3)==0); }
 Val=1; xStatus = xQueueSendToBack(xQueue,&Val,0);
 }
 xSemaphoreGive(xMutex); vTaskDelay(100);
}//END OF WHILE
}//END OF FUNCTION
```

AUTOMATIC OPEN/CLOSE FUNCTION: INITIALIZATION

```
void buttonsInit(void) { }
void limitInit(void) {}
```

AUTOMATIC CLOSE FUNCTION

```
void driver(void* pvParameters){
  int Val, state;
  portBASE_TYPE xStatus; //Val= (int) pvParameters;
  while(1){
  if (GET_BIT(GPIO_PORTD_DATA_R,0)==0) //still pressing then it is manual {}
  else if (GET_BIT(GPIO_PORTD_DATA_R,0)==1) // then it will be automatic {
        while(!(GET_BIT(GPIO_PORTA_DATA_R,7)==1 | GET_BIT(GPIO_PORTD_DATA_R,0)==0 |
        GET_BIT(GPIO_PORTD_DATA_R,1)==0));
        }
  Val=1; xStatus = xQueueSendToBack(xQueue,&Val,0); }//END OF CLOSE
  //OPEN //LOCK}
```

AUTOMATIC CLOSE FUNCTION

```
void passenger(void* pvParameters){
 int Val; portBASE_TYPE xStatus;
 while(1) {
  xSemaphoreTake(xMutex,portMAX_DELAY );
  if (GET_BIT(GPIO_PORTD_DATA_R,2)==0) //manual {}
  else if (GET_BIT(GPIO_PORTD_DATA_R,2)==1) // then it will be automatic {
       while(!(GET_BIT(GPIO_PORTA_DATA_R,7)==1 |
       GET_BIT(GPIO_PORTD_DATA_R,2)==0 | GET_BIT(GPIO_PORTD_DATA_R,3)==0));
}
 Val=1; xStatus = xQueueSendToBack(xQueue,&Val,0);
}//END OF CLOSE
```

AUTOMATIC OPEN FUNCTION

```
void driver(void* pvParameters){
 if (GET_BIT(GPIO_PORTD_DATA_R,1)==0){ //pullup open
 Val=3; xStatus = xQueueSendToBack(xQueue,&Val,0);
 vTaskDelay(1000);
 if (GET_BIT(GPIO_PORTD_DATA_R,1)==0) //manual {}
 else if (GET_BIT(GPIO_PORTD_DATA_R,1)==1) // then it will be automatic {
        while(!(GET_BIT(GPIO_PORTD_DATA_R,6)==1 |GET_BIT(GPIO_PORTD_DATA_R,1)==0 |
        GET_BIT(GPIO_PORTD_DATA_R,0)==0)); }
 Val=1;
 xStatus = xQueueSendToBack(xQueue,&Val,0); }
}//Lock Function(ONLY DRIVER CONTROLS IT) has a special condition that can work when
either window is open or window is closed
}//END OF WHILE  }//END OF FUNCTION
```

AUTOMATIC OPEN FUNCTION

```
void passenger(void* pvParameters){
 if (GET_BIT(GPIO_PORTD_DATA_R,3)==0){ //pullup open
 Val=3; xStatus = xQueueSendToBack(xQueue,&Val,0);
 vTaskDelay(1000);
 if (GET_BIT(GPIO_PORTD_DATA_R,3)==0) //manual {}
 else if (GET_BIT(GPIO_PORTD_DATA_R,3)==1) // then it will be automatic {
        while(!(GET_BIT(GPIO_PORTD_DATA_R,6)==1| GET_BIT(GPIO_PORTD_DATA_R,3)==0 |
        GET_BIT(GPIO_PORTD_DATA_R,2)==0));}
 Val=1; //Turn off motor
 xStatus = xQueueSendToBack(xQueue,&Val,0); }
 xSemaphoreGive(xMutex); vTaskDelay(100);
}//END OF OPEN
}//END OF TASK
```

WINDOW LOCK FUNCTION: INITIALIZATION

```
void buttonsInit(void) { }
void lockButtonInit(void) {
//Configure Pin 3 in Port A as input
GPIO_PORTA_DIR_R &= ~(1 << 3);
GPIO_PORTA_CR_R |= (1<<3);
GPIO_PORTA_PUR_R |= (1<<3);
GPIO_PORTA_DEN_R |= (1<<3);
}
```

WINDOW LOCK CLOSE/OPEN FUNCTION

```
void driver(void* pvParameters){
 if (GET_BIT(GPIO_PORTD_DATA_R,0)==0){ //pullup close
  if (GET_BIT(GPIO_PORTD_DATA_R,0)==0) //still pressing then it is manual {}
  else if (GET_BIT(GPIO_PORTD_DATA_R,0)==1) // then it will be automatic {} //Turn off motor}
 if (GET_BIT(GPIO_PORTD_DATA_R,1)==0){ //pullup open
  if (GET_BIT(GPIO_PORTD_DATA_R,1)==0) //still pressing then it is manual {}
  else if (GET_BIT(GPIO_PORTD_DATA_R,1)==1) // then it will be automatic {} //Turn off motor}
// Lock Switch
 if (GET_BIT(GPIO_PORTA_DATA_R,3)==0){
 vTaskPrioritySet(NULL,2); }
 else { vTaskPrioritySet(NULL,1); }
 xSemaphoreGive(xMutex); vTaskDelay(100); } }//END OF TASK
```

JAM FUNCTION : INITIALIZATION

```
void sensorButtonInit(void) {
//Enable Port A SYSCTL_RCGCGPIO_R |= 0x01;
__asm__("NOP; NOP; NOP; NOP;"); //Configure Pin 2 in Port A as input
GPIO_PORTA_DIR_R &= ~(1 << 2);
GPIO_PORTA_CR_R |= (1 << 2);
GPIO_PORTA_PUR_R |= (1 << 2);
GPIO_PORTA_DEN_R |= (1 << 2); //Enable Interrupt on PORT A & set priority to 0
NVIC_PRI0_R |= (1<<7) | (1<<6) | (1<<5);
NVIC_EN0_R |= (1<<0); //Configure Interrupt on Pin 2 to detect FALLING edge
GPIO_PORTA_IM_R &=0;
GPIO_PORTA_IS_R &= ~(1<<2);
GPIO_PORTA_IEV_R &= ~(1<<2);
GPIO_PORTA_ICR_R |= (1<<2);
GPIO_PORTA_IM_R |= (1<<2); }
```

```
void sensorButtonInit(void);//jam function
void timer0Init(void);      //jam function
void timer0_Delay(int time);//jam function
void motorInit(void);       //driver/passenger/jam
void limitInit(void);       //driver/passenger/ jam
void buttonsInit(void);     //driver/passenger function
void lockButtonInit(void);  //lock in driver/passenger function
```

JAM FUNCTION : INITIALIZATION

```
void timer0Init(void) {
SYSCTL_RCGCTIMER_R |= 0x01;
TIMER0_CTL_R=0x00; TIMER0_CFG_R=0x00;  TIMER0_TAMR_R=0x02; TIMER0_CTL_R=0x03; }
void timer0_Delay(int time) {
TIMER0_CTL_R=0x00; TIMER0_TAILR_R=16000*time-1; TIMER0_ICR_R=0x01;
TIMER0_CTL_R |=0x03;
while((TIMER0_RIS_R & 0x01)==0); }
void motorInit(void) {
SYSCTL_RCGCGPIO_R |= 0x20;
__asm__("NOP; NOP; NOP; NOP;");
//Configure Pin 2,3 in Port F as inputs
GPIO_PORTF_DIR_R |= ((1 << 2)|(1<<3));
GPIO_PORTF_CR_R |= (1 << 2)|(1<<3);
GPIO_PORTF_DEN_R |= (1 << 2)|(1<<3); }
void limitInit(void); //driver/passenger/ jam
```

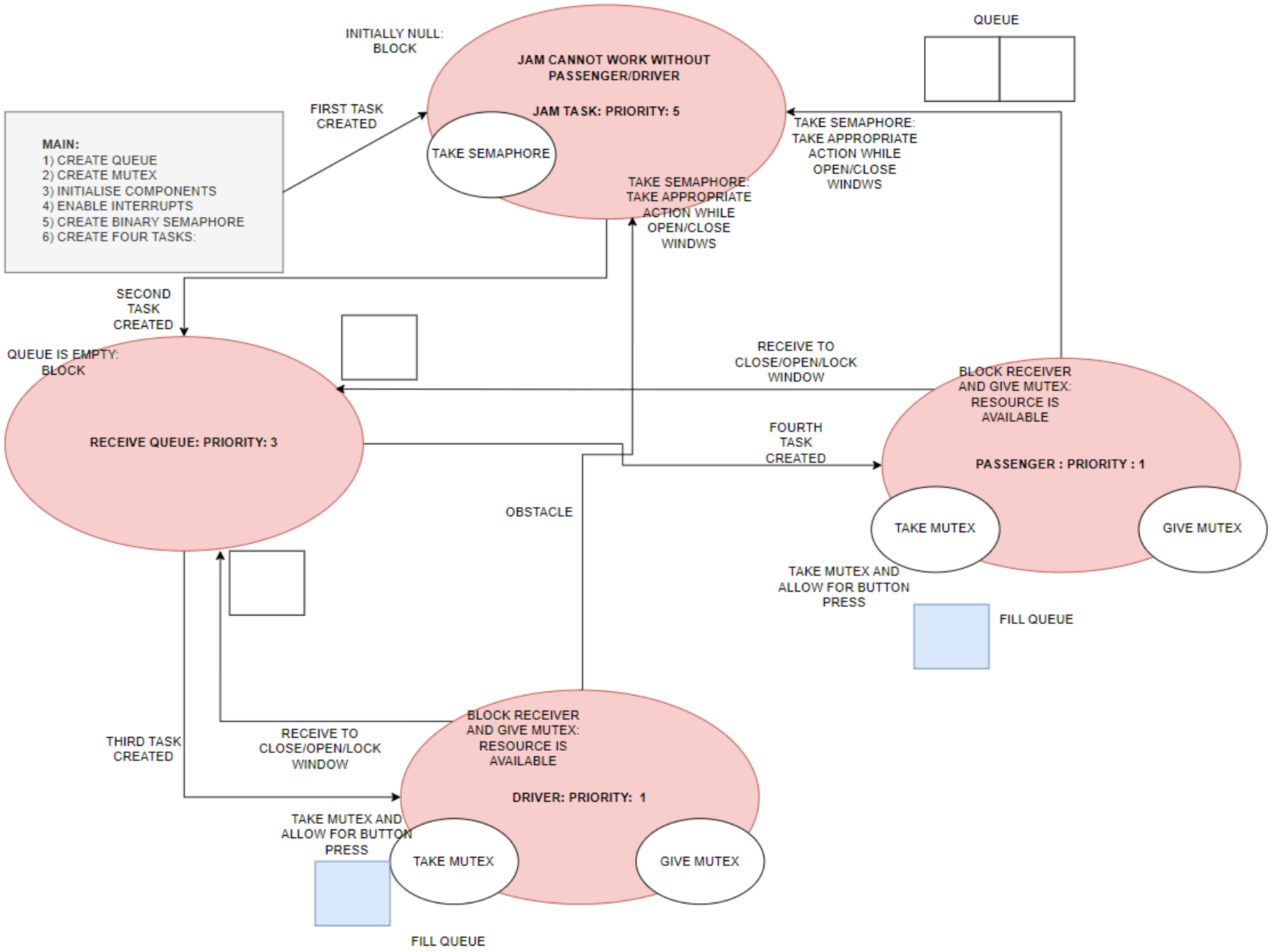
JAM FUNCTION

```
void jamTask(void* pvParameters) {
//TAKE SEMAPHORE
xSemaphoreTake(xBinarySemaphore, 0);
while (1) {
//TAKE SEMAPHORE
xSemaphoreTake(xBinarySemaphore, portMAX_DELAY);
// Set motor direction to reverse
GPIO_PORTF_DATA_R |= (1 << 3);
GPIO_PORTF_DATA_R &= ~(1 << 2);
timer0_Delay(2000);
// Stop motor
GPIO_PORTF_DATA_R &= ~(1 << 3);
GPIO_PORTF_DATA_R &= ~(1 << 2); }//END OF WHILE }//END OF TASK
```

QUEUING TASK

```
void recieveQueue(void* pvParameters) {
int Val;  portBASE_TYPE xStatus;
const portTickType xTicks=100/portTICK_RATE_MS;
while(1) {
xStatus=xQueueReceive(xQueue,&Val,portMAX_DELAY);
if(Val==1) //TURN OFF MOTOR {
GPIO_PORTF_DATA_R &= ~(1 << 3); GPIO_PORTF_DATA_R &= ~(1 << 2); }
else if(Val==2) // CLOSE WINDOW {
GPIO_PORTF_DATA_R |= (1 << 3); GPIO_PORTF_DATA_R &= ~(1 << 2); }
else if(Val==3) //OPEN WINDOW {
GPIO_PORTF_DATA_R &= ~(1 << 3); GPIO_PORTF_DATA_R |= (1 << 2); } } }
```

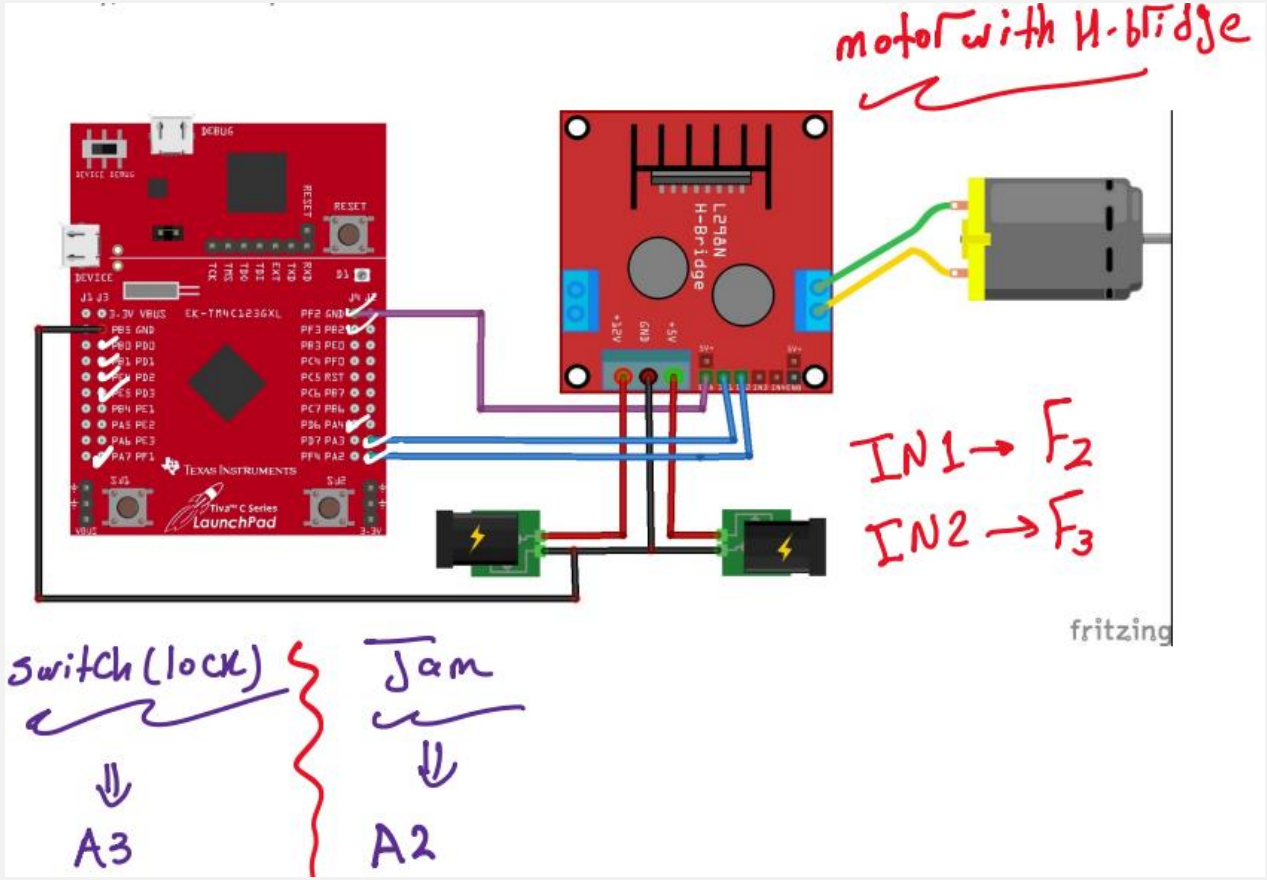
CONTENTS

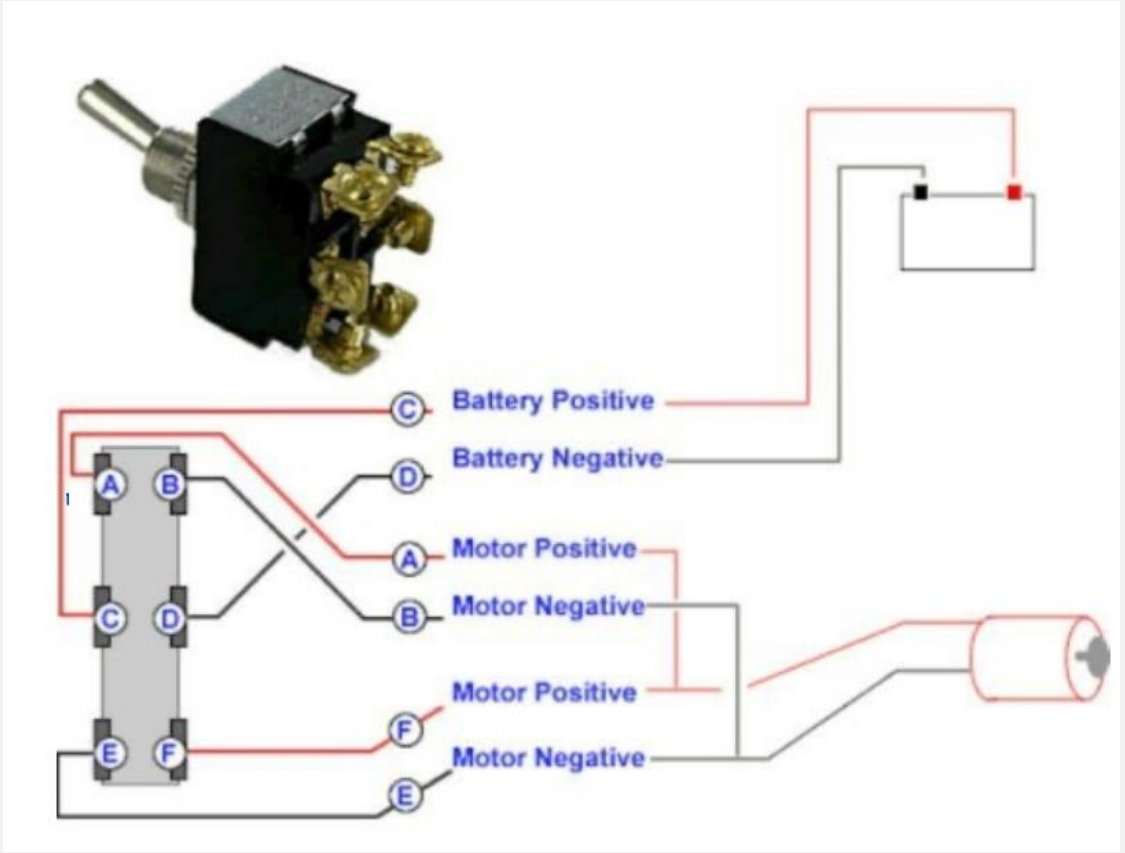

INITIALLY NULL:
BLOCK
JAM CANNOT WORK WITHOUT
PASSENGER/DRIVER
JAM TASK: PRIORITY: 5
TAKE SEMAPHORE
QUEUE
FIRST TASK
CREATED
MAIN:
1) CREATE QUEUE
2) CREATE MUTEX
3) INITIALISE COMPONENTS
4) ENABLE INTERRUPTS
5) CREATE BINARY SEMAPHORE
6) CREATE FOUR TASKS:
TAKE SEMAPHORE:
TAKE APPROPRIATE
ACTION WHILE
OPEN/CLOSE
WINDWS
TAKE SEMAPHORE:
TAKE APPROPRIATE
ACTION WHILE
OPEN/CLOSE
WINDWS
SECOND
TASK
CREATED
QUEUE IS EMPTY:
BLOCK
RECEIVE QUEUE: PRIORITY: 3
RECEIVE TO
CLOSE/OPEN/LOCK
WINDOW
BLOCK RECEIVER
AND GIVE MUTEX:
RESOURCE IS
AVAILABLE
PASSENGER : PRIORITY : 1
FOURTH
TASK
CREATED
OBSTACLE
TAKE MUTEX
GIVE MUTEX
TAKE MUTEX AND
ALLOW FOR BUTTON
PRESS
FILL QUEUE
THIRD TASK
CREATED
RECEIVE TO
CLOSE/OPEN/LOCK
WINDOW
BLOCK RECEIVER
AND GIVE MUTEX:
RESOURCE IS
AVAILABLE
DRIVER: PRIORITY: 1
TAKE MUTEX AND
ALLOW FOR BUTTON
PRESS
TAKE MUTEX
GIVE MUTEX
FILL QUEUE

STATE DIAGRAM.html

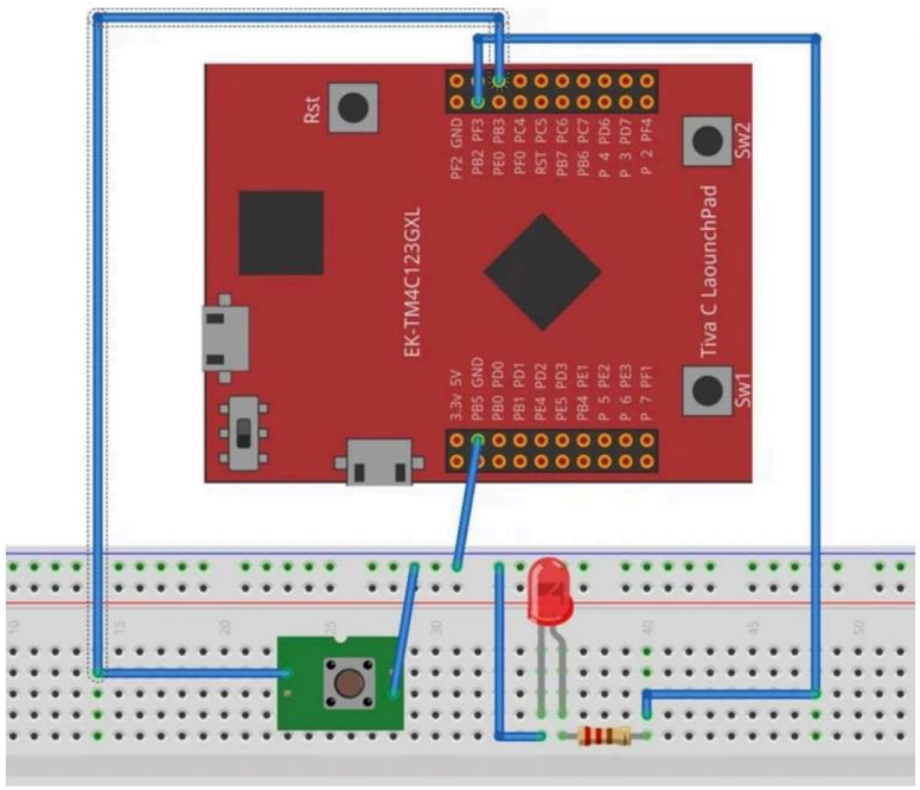
CONTENTS

CIRCUIT ASSEMBLY AND TOPOLOGY: MOTOR AND H-BRIDGE



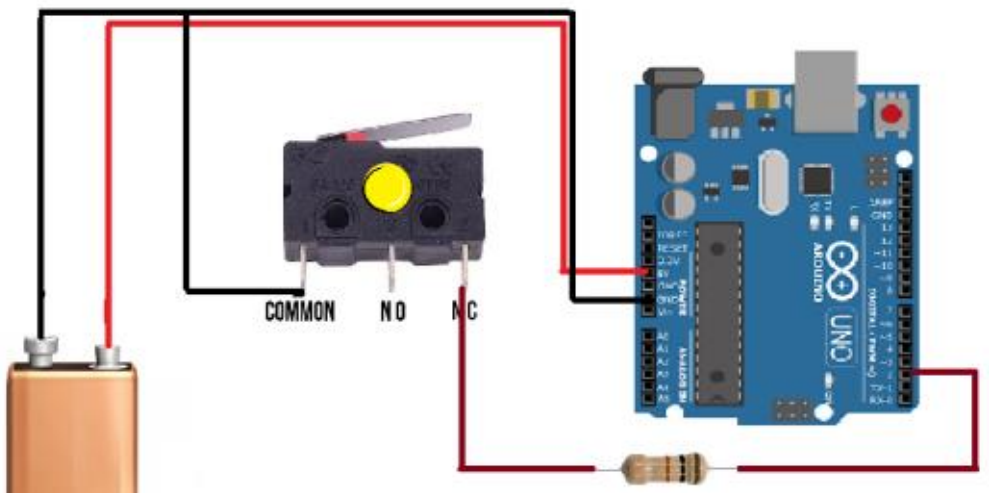


CIRCUIT ASSEMBLY AND TOPOLOGY: PUSH BUTTONS AND LIMIT SWITCHES



Push buttons

PD0
PD1
PD2
PD3



PDb → for limit switch 1;
PAT → for limit switch 2;

CONTENTS

TESTING VIDEO URL, GITHUB REPOSITORY URL, MATLAB URL

- TESTING VIDEO:
- GITHUB: https://github.com/PerfectionistAF/OS-CAD_FREERTOS
- MATLAB: https://drive.matlab.com/sharing/913fda3c-ad13-45e2-970e-817025c25ef4