Madonna Magdy Moussa

19p2671

RTOS Project

Video Link:

https://drive.google.com/file/d/1sb3JMAEtqwqW7euUf1AV_mxXcDTTKGsr/view?usp=share_link

Project Link:

https://github.com/madonnamagdymoussa/PowerWindowControlSystem_FreeRTOS_usingTivaC.git

Components used:

1- Infrared sensor (for obstacle detection)

2- Bluetooth module HC-05 (print the tasks which the processor currently executing).

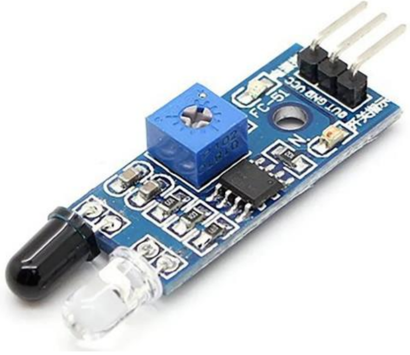
3- two motors (one for the passenger and one for the driver) and a motor driver

4- four manual switches

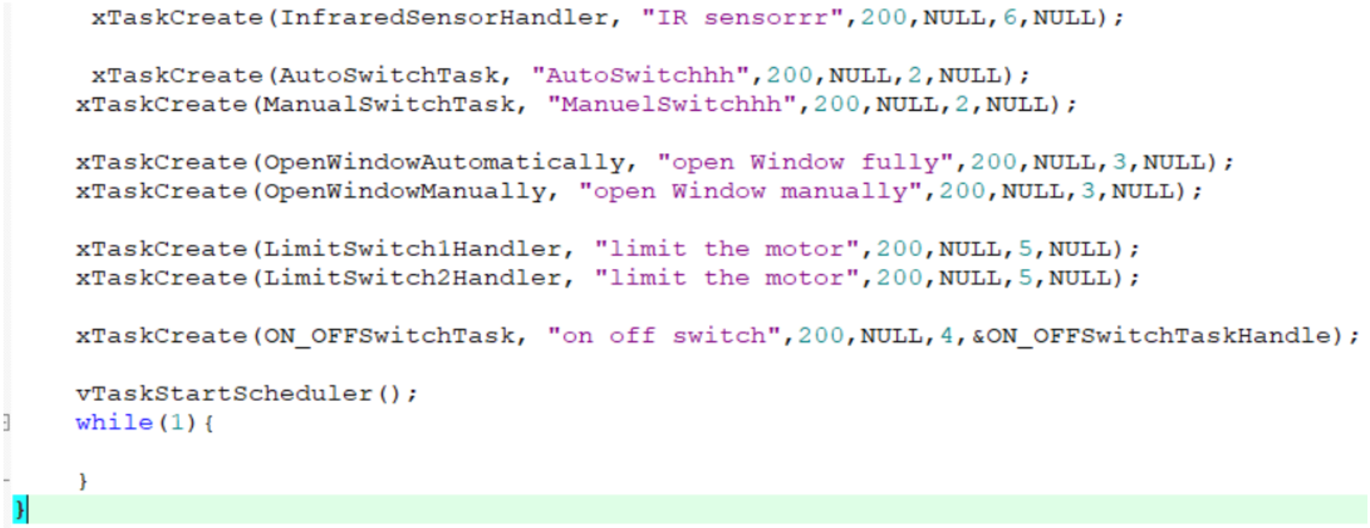
5- four automatic switches

6- two limit switches

7- one on/off switch



My system Design



```
     xTaskCreate(InfraredSensorHandler, "IR sensorrr",200,NULL,6,NULL);

     xTaskCreate(AutoSwitchTask, "AutoSwitchhh",200,NULL,2,NULL);
    xTaskCreate(ManualSwitchTask, "ManuelSwitchhh",200,NULL,2,NULL);

    xTaskCreate(OpenWindowAutomatically, "open Window fully",200,NULL,3,NULL);
    xTaskCreate(OpenWindowManually, "open Window manually",200,NULL,3,NULL);

    xTaskCreate(LimitSwitch1Handler, "limit the motor",200,NULL,5,NULL);
    xTaskCreate(LimitSwitch2Handler, "limit the motor",200,NULL,5,NULL);

    xTaskCreate(ON_OFFSwitchTask, "on off switch",200,NULL,4,&ON_OFFSwitchTaskHandle);

    vTaskStartScheduler();
    while(1){

    }
}
```

There are 8 tasks and five different priorities.

- There are four queues, each queue is related to certain automatic switch. The task autoSwitchTask() checks on each automatic button (there are four automatic buttons) if certain automatic button is pressed then it sends a value to the queue. The task openwindowAutomatically is blocked from reading on queue it will wait until the lower priority task AutoSwitchTask() writes on the queue.

- The task ManuelSwitch() uses one semaphore for the four manual switches and checks if on of the switches is pressed the it gives that semaphore. The other higher priority task which is OpenWindowManually will take that semaphore and checks which switch is pressed.

- The ON_OFFSwitch task allows the driver to use his buttons but does not allow the passenger to use his buttons. So, the ON_OFFSwitch task checks if the manual buttons of drivers is pressed or not and if it is pressed it gives the semaphore of the manual button **then we lower the priority of the task ON_OFFSwitch to allow the lower priority task like openWindowManually() to take the semaphore then in the openWindowManually() we raise the priority of the ON_OFFSwitch.**

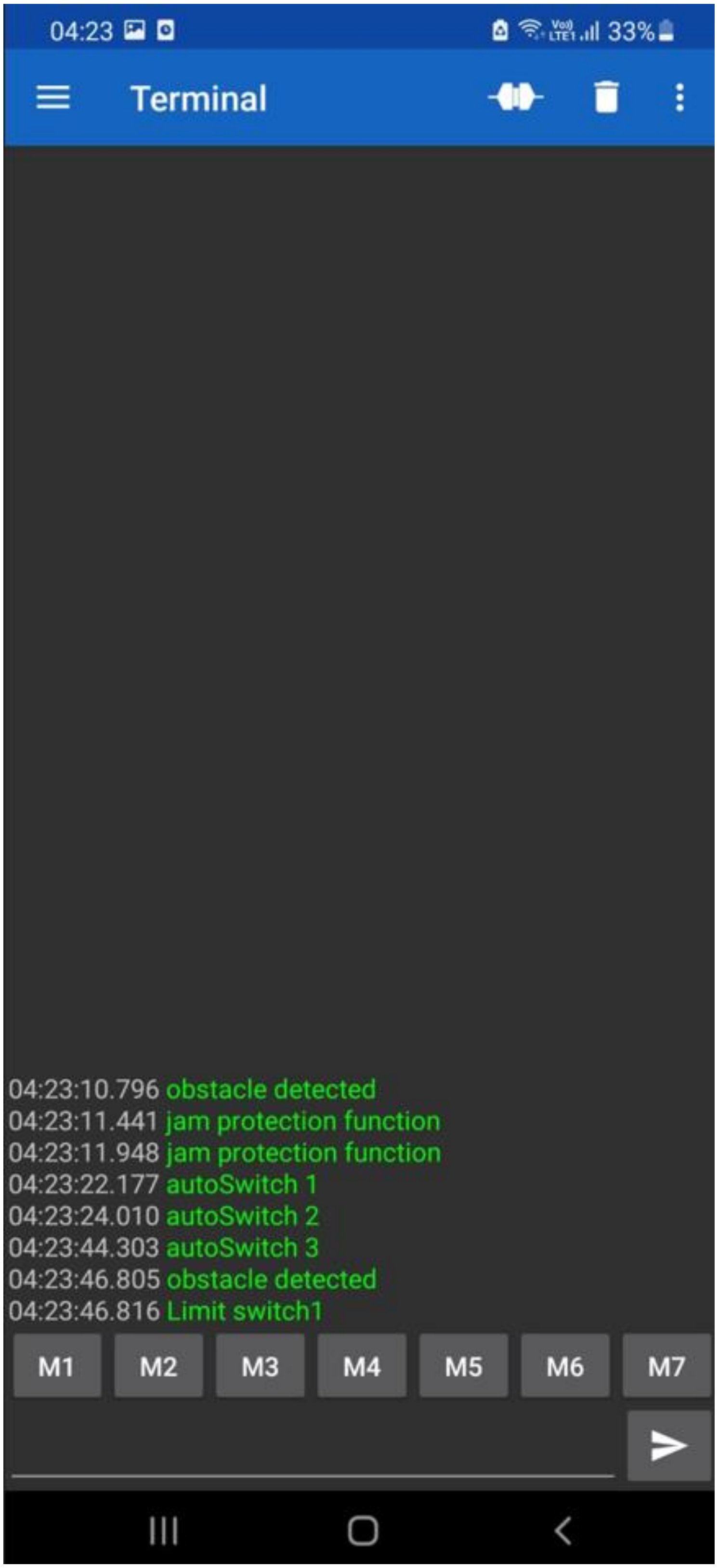
The same concept applies when the ON_OFFSwitch task checks on the manual switch and writes on the queue.

- There are two limit switch handlers the limit switches work with interrupts so there are handlers and inside each call back function it gives the semaphore from the ISR and the handler takes that semaphore.

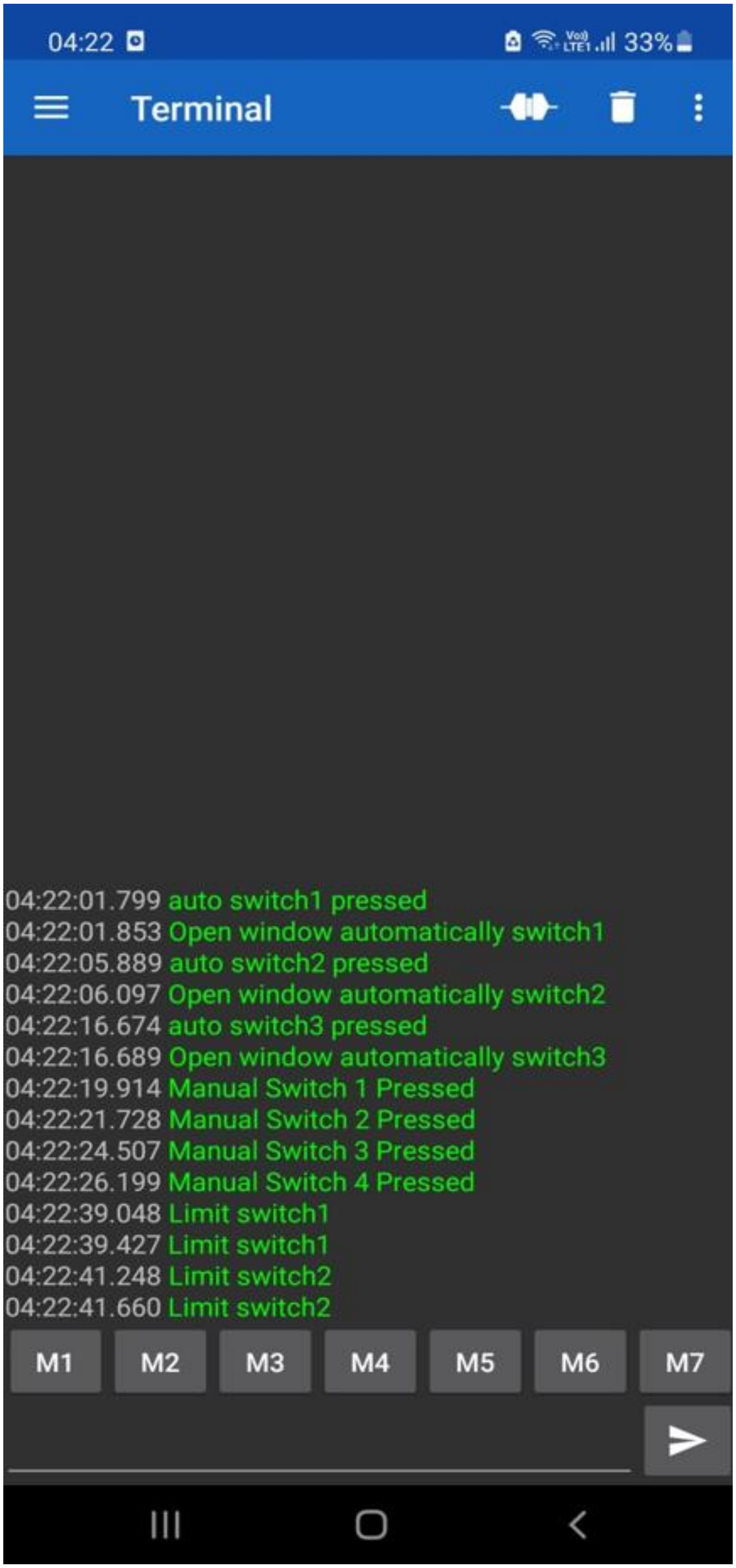
- The highest priority is the Infrared Sensor it also works with the interrupt and inside its call back function it gives the semaphore from the ISR and the handler takes that semaphore.

- There is a mutex inside the handler the mutex is on the manual switches we could not use the manual switch inside the Infrared Sensor handler but we can use automatic switch.

HC-05 Mobile Terminal

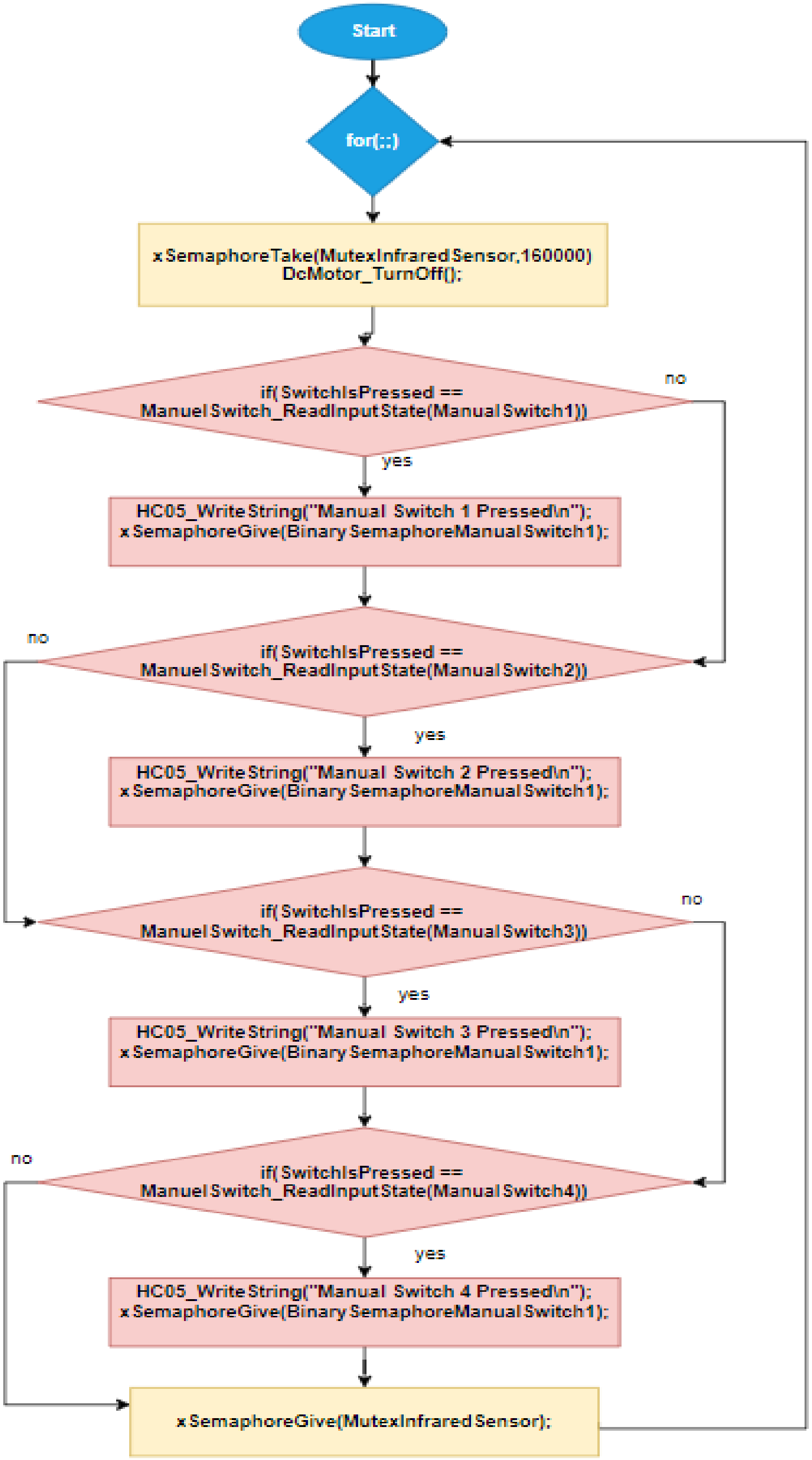


```
04:23:10.796 obstacle detected
04:23:11.441 jam protection function
04:23:11.948 jam protection function
04:23:22.177 autoSwitch 1
04:23:24.010 autoSwitch 2
04:23:44.303 autoSwitch 3
04:23:46.805 obstacle detected
04:23:46.816 Limit switch1
```

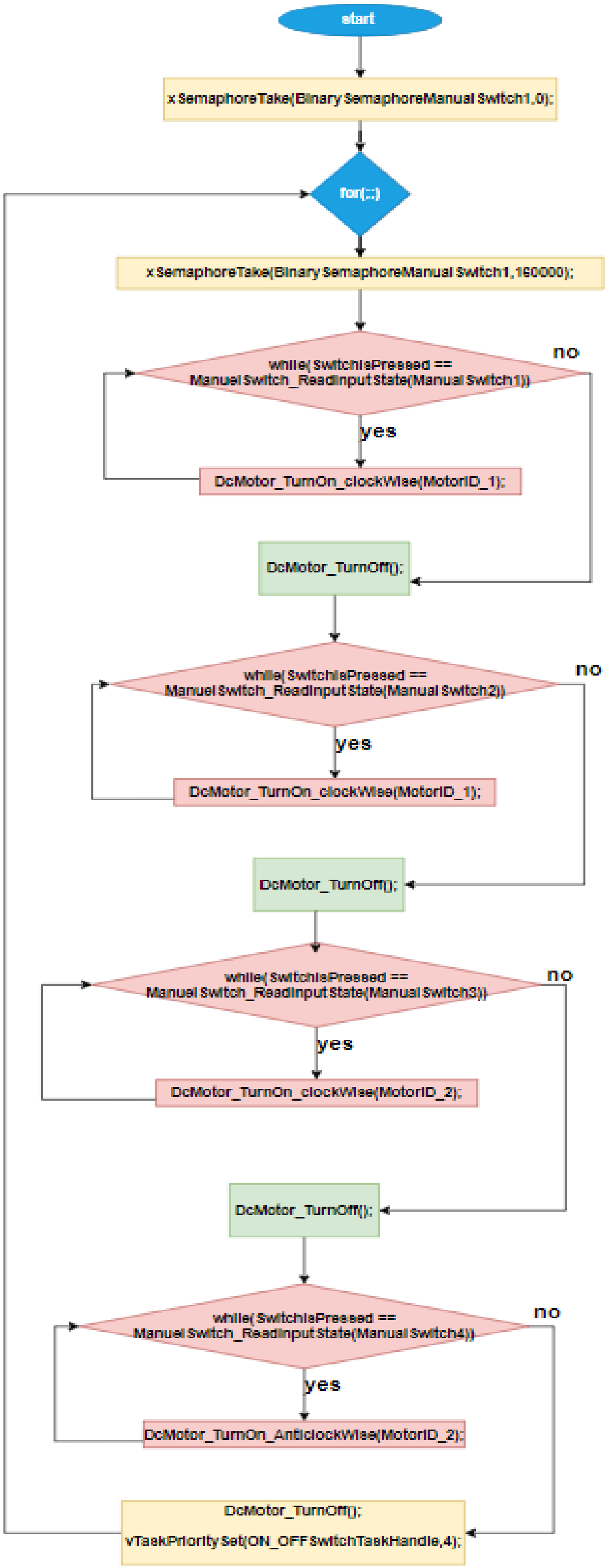


```
04:22:01.799 auto switch1 pressed
04:22:01.853 Open window automatically switch1
04:22:05.889 auto switch2 pressed
04:22:06.097 Open window automatically switch2
04:22:16.674 auto switch3 pressed
04:22:16.689 Open window automatically switch3
04:22:19.914 Manual Switch 1 Pressed
04:22:21.728 Manual Switch 2 Pressed
04:22:24.507 Manual Switch 3 Pressed
04:22:26.199 Manual Switch 4 Pressed
04:22:39.048 Limit switch1
04:22:39.427 Limit switch1
04:22:41.248 Limit switch2
04:22:41.660 Limit switch2
```

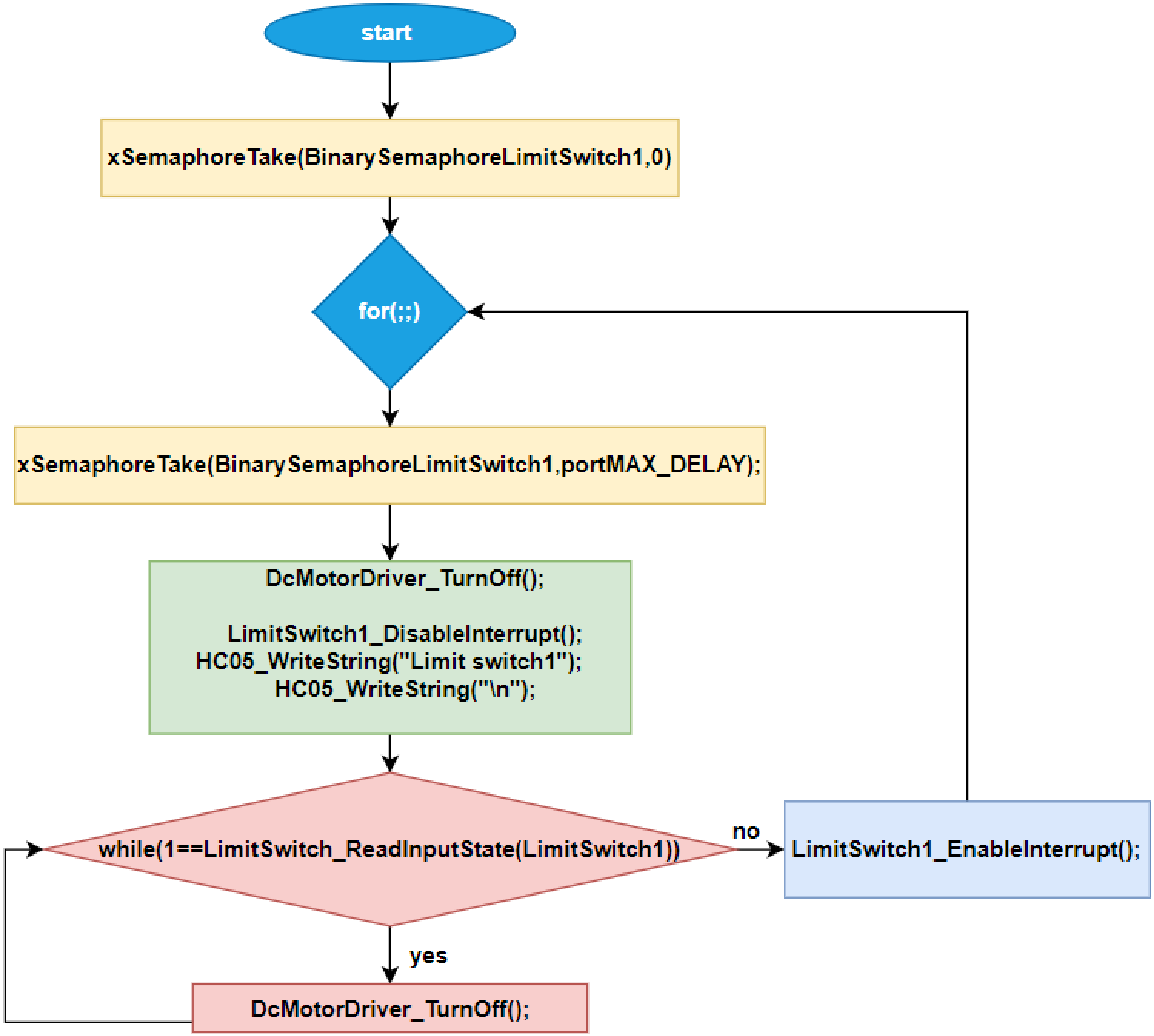
Manual Switch Task


Start
for(;;)
xSemaphoreTake(MutexInfraredSensor,160000)
DcMotor_TurnOff();
if(SwitchIsPressed ==
ManuelSwitch_ReadInputState(ManualSwitch1))
no
yes
HC05_WriteString("Manual Switch 1 Pressed\n");
xSemaphoreGive(BinarySemaphoreManualSwitch1);
no
if(SwitchIsPressed ==
ManuelSwitch_ReadInputState(ManualSwitch2))
yes
HC05_WriteString("Manual Switch 2 Pressed\n");
xSemaphoreGive(BinarySemaphoreManualSwitch1);
if(SwitchIsPressed ==
ManuelSwitch_ReadInputState(ManualSwitch3))
no
yes
HC05_WriteString("Manual Switch 3 Pressed\n");
xSemaphoreGive(BinarySemaphoreManualSwitch1);
no
if(SwitchIsPressed ==
ManuelSwitch_ReadInputState(ManualSwitch4))
yes
HC05_WriteString("Manual Switch 4 Pressed\n");
xSemaphoreGive(BinarySemaphoreManualSwitch1);
xSemaphoreGive(MutexInfraredSensor);

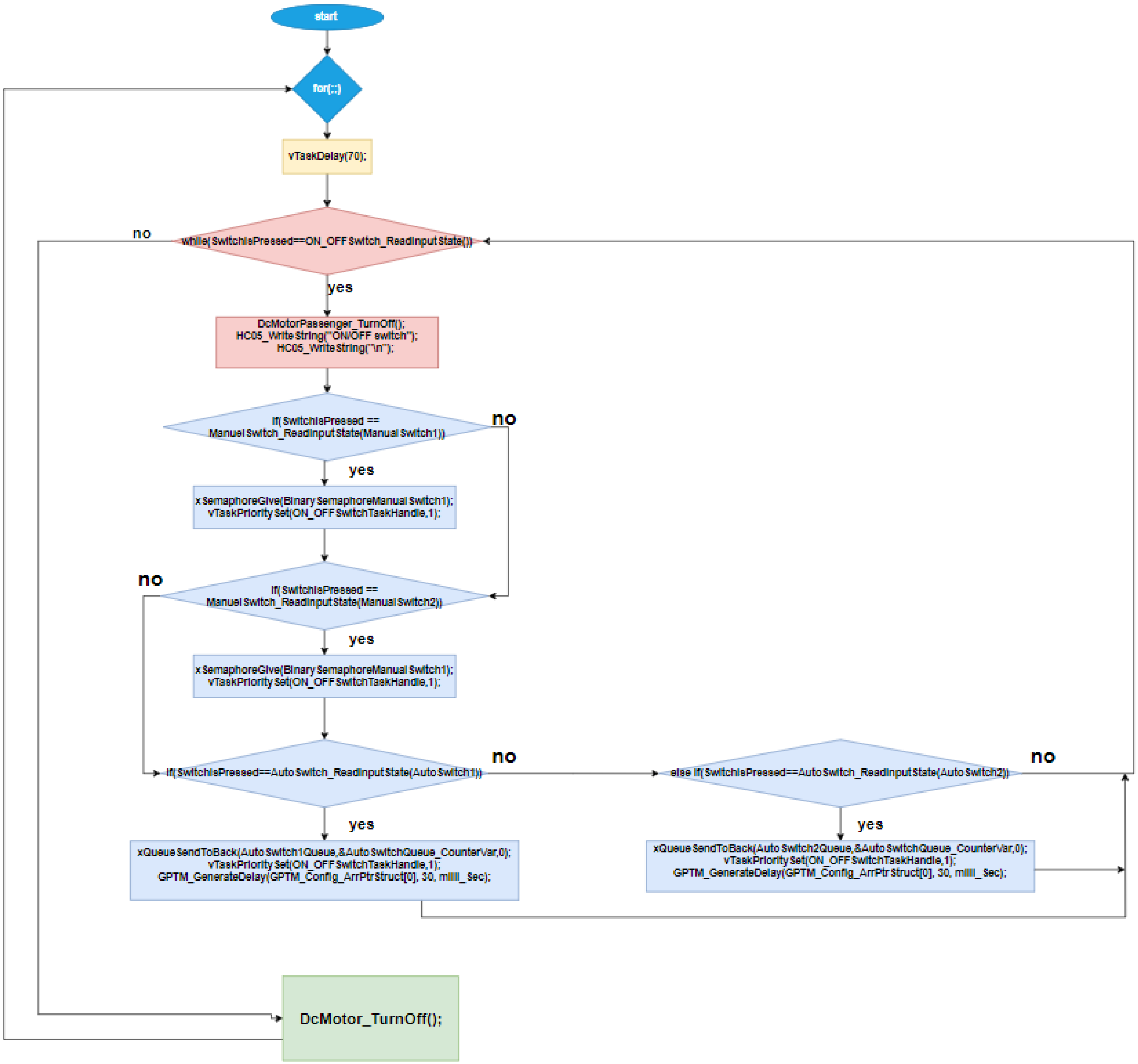
OpenWindowManually Task


start
x SemaphoreTake(Binary SemaphoreManual Switch1,0);
for(;;)
x SemaphoreTake(Binary SemaphoreManual Switch1,160000);
while(SwitchIsPressed == Manuel Switch_ReadInput State(Manual Switch1))
no
yes
DcMotor_TurnOn_clockWise(MotorID_1);
DcMotor_TurnOff();
while(SwitchIsPressed == Manuel Switch_ReadInput State(Manual Switch2))
no
yes
DcMotor_TurnOn_clockWise(MotorID_1);
DcMotor_TurnOff();
while(SwitchIsPressed == Manuel Switch_ReadInput State(Manual Switch3))
no
yes
DcMotor_TurnOn_clockWise(MotorID_2);
DcMotor_TurnOff();
while(SwitchIsPressed == Manuel Switch_ReadInput State(Manual Switch4))
no
yes
DcMotor_TurnOn_AnticlockWise(MotorID_2);
DcMotor_TurnOff();
vTaskPriority Set(ON_OFF SwitchTaskHandle,4);

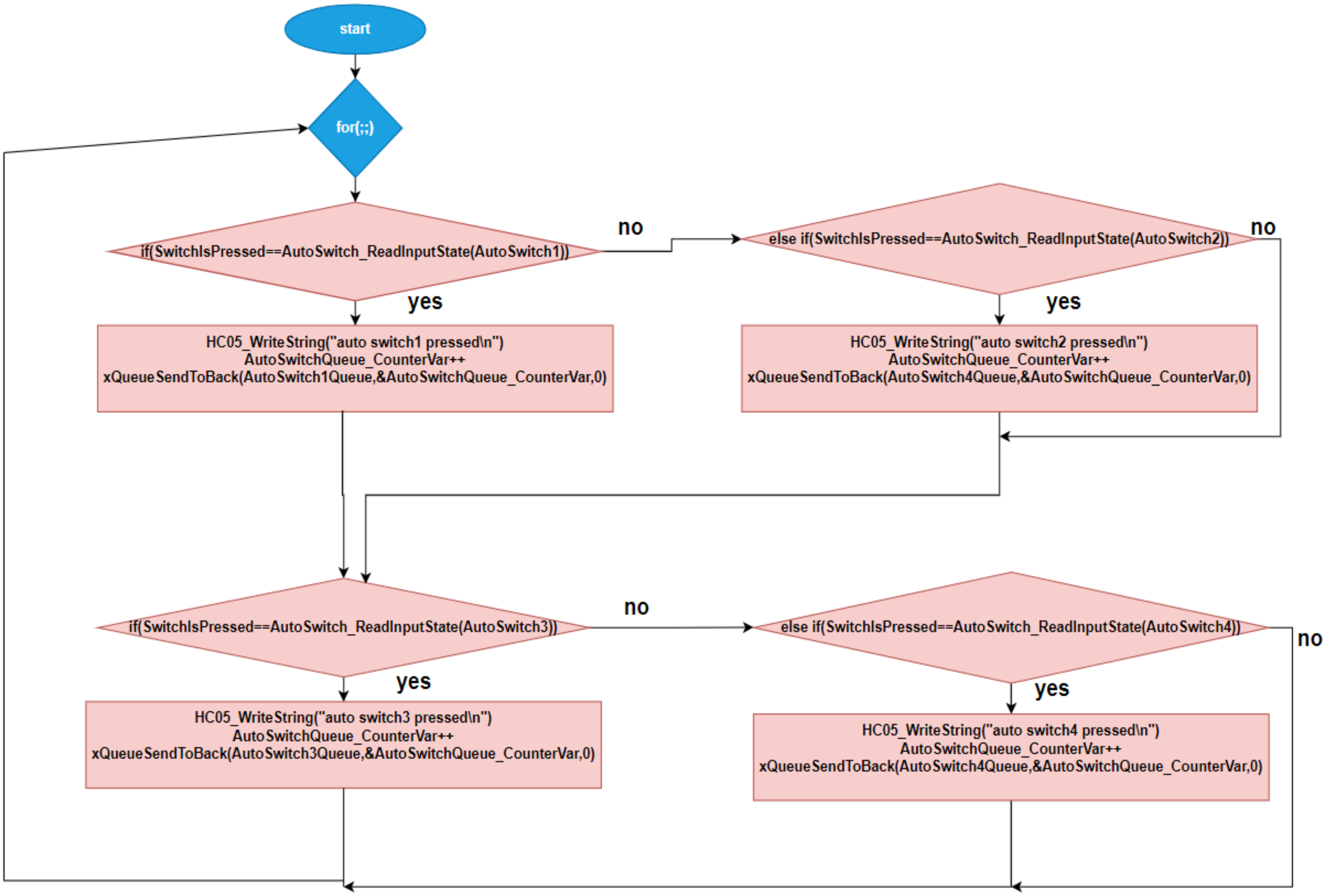
Limit Switch Task


start
xSemaphoreTake(BinarySemaphoreLimitSwitch1,0)
for(;;)
xSemaphoreTake(BinarySemaphoreLimitSwitch1,portMAX_DELAY);
DcMotorDriver_TurnOff();
LimitSwitch1_DisableInterrupt();
HC05_WriteString("Limit switch1");
HC05_WriteString("\n");
while(1==LimitSwitch_ReadInputState(LimitSwitch1))
no
LimitSwitch1_EnableInterrupt();
yes
DcMotorDriver_TurnOff();

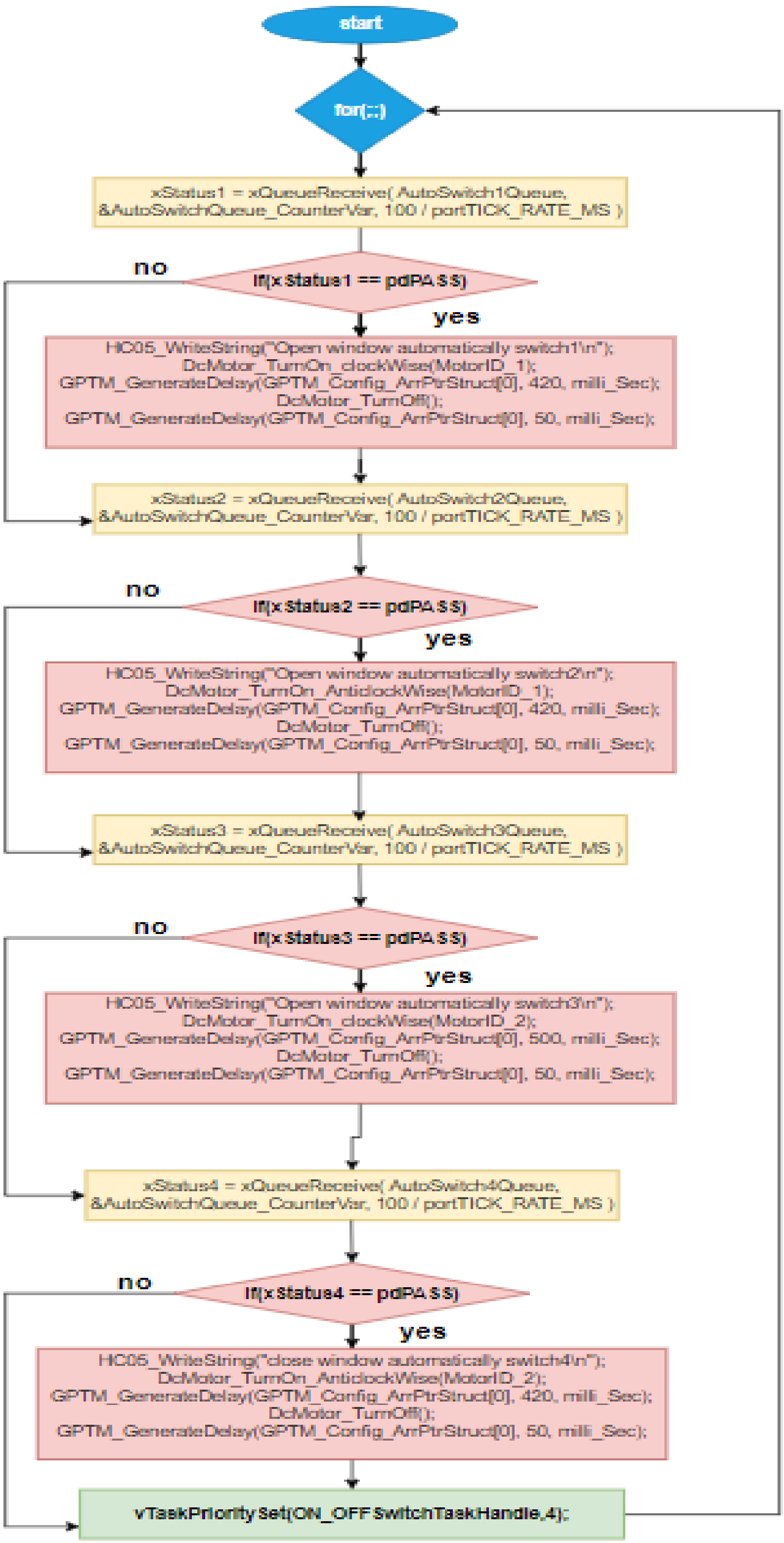
ON/OFF Switch Task


start
for(;;)
vTaskDelay(70);
no
while(SwitchIsPressed==ON_OFF Switch_ReadInput State())
yes
DcMotorPassenger_TurnOff();
HC05_Write String("ON/OFF switch");
HC05_Write String("\n");
If(SwitchIsPressed ==
Manuel Switch_ReadInput State(Manual Switch1))
no
yes
x SemaphoreGive(Binary SemaphoreManual Switch1);
vTaskPriority Set(ON_OFF SwitchTaskHandle,1);
no
If(SwitchIsPressed ==
Manuel Switch_ReadInput State(Manual Switch2))
yes
x SemaphoreGive(Binary SemaphoreManual Switch1);
vTaskPriority Set(ON_OFF SwitchTaskHandle,1);
If(SwitchIsPressed==Auto Switch_ReadInput State(Auto Switch1))
no
yes
xQueue SendToBack(Auto Switch1Queue,&Auto SwitchQueue_CounterVar,0);
vTaskPriority Set(ON_OFF SwitchTaskHandle,1);
GPTM_GenerateDelay(GPTM_Config_ArrPtr Struct[0], 30, milli_Sec);
else If(SwitchIsPressed==Auto Switch_ReadInput State(Auto Switch2))
no
yes
xQueue SendToBack(Auto Switch2Queue,&Auto SwitchQueue_CounterVar,0);
vTaskPriority Set(ON_OFF SwitchTaskHandle,1);
GPTM_GenerateDelay(GPTM_Config_ArrPtr Struct[0], 30, milli_Sec);
DcMotor_TurnOff();

Automatic Switch Task


start
for(;;)
if(SwitchIsPressed==AutoSwitch_ReadInputState(AutoSwitch1))
no
else if(SwitchIsPressed==AutoSwitch_ReadInputState(AutoSwitch2))
no
yes
HC05_WriteString("auto switch1 pressed\n")
AutoSwitchQueue_CounterVar++
xQueueSendToBack(AutoSwitch1Queue,&AutoSwitchQueue_CounterVar,0)
yes
HC05_WriteString("auto switch2 pressed\n")
AutoSwitchQueue_CounterVar++
xQueueSendToBack(AutoSwitch4Queue,&AutoSwitchQueue_CounterVar,0)
if(SwitchIsPressed==AutoSwitch_ReadInputState(AutoSwitch3))
no
else if(SwitchIsPressed==AutoSwitch_ReadInputState(AutoSwitch4))
no
yes
HC05_WriteString("auto switch3 pressed\n")
AutoSwitchQueue_CounterVar++
xQueueSendToBack(AutoSwitch3Queue,&AutoSwitchQueue_CounterVar,0)
yes
HC05_WriteString("auto switch4 pressed\n")
AutoSwitchQueue_CounterVar++
xQueueSendToBack(AutoSwitch4Queue,&AutoSwitchQueue_CounterVar,0)

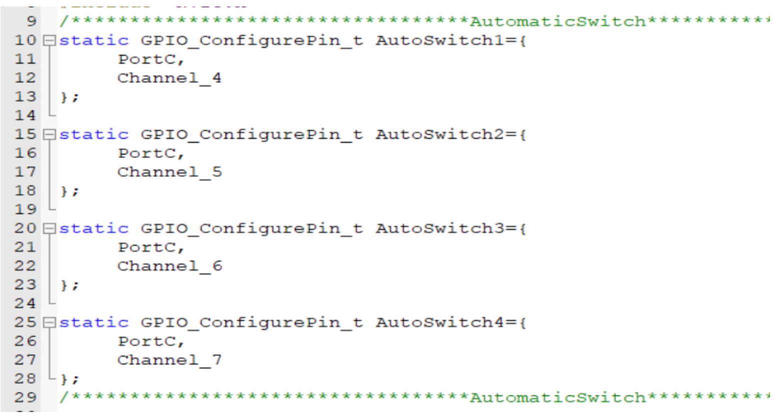
Open window Automatically



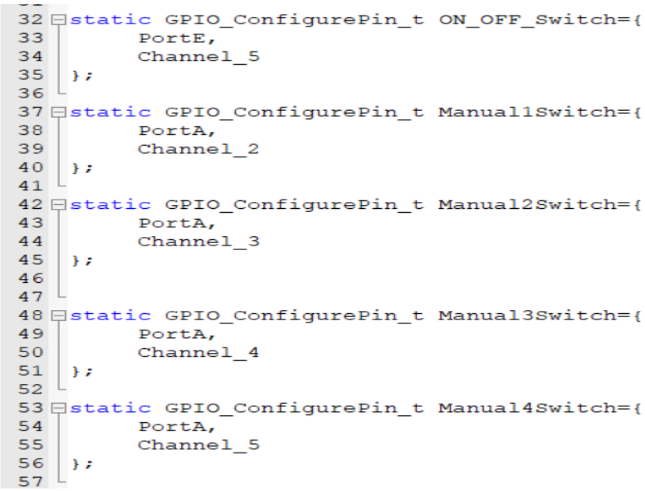
```
xStatus1 = xQueueReceive( AutoSwitch1Queue,
&AutoSwitchQueue_CounterVar, 100 / portTICK_RATE_MS )

If(xStatus1 == pdPASS)

HC05_WriteString("Open window automatically switch1\n");
DcMotor_TurnOn_clockWise(MotorID_1);
GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 420, milli_Sec);
DcMotor_TurnOff();
GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 50, milli_Sec);

xStatus2 = xQueueReceive( AutoSwitch2Queue,
&AutoSwitchQueue_CounterVar, 100 / portTICK_RATE_MS )

If(xStatus2 == pdPASS)

HC05_WriteString("Open window automatically switch2\n");
DcMotor_TurnOn_AnticlockWise(MotorID_1);
GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 420, milli_Sec);
DcMotor_TurnOff();
GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 50, milli_Sec);

xStatus3 = xQueueReceive( AutoSwitch3Queue,
&AutoSwitchQueue_CounterVar, 100 / portTICK_RATE_MS )

If(xStatus3 == pdPASS)

HC05_WriteString("Open window automatically switch3\n");
DcMotor_TurnOn_clockWise(MotorID_2);
GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 500, milli_Sec);
DcMotor_TurnOff();
GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 50, milli_Sec);

xStatus4 = xQueueReceive( AutoSwitch4Queue,
&AutoSwitchQueue_CounterVar, 100 / portTICK_RATE_MS )

If(xStatus4 == pdPASS)

HC05_WriteString("close window automatically switch4\n");
DcMotor_TurnOn_AnticlockWise(MotorID_2);
GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 420, milli_Sec);
DcMotor_TurnOff();
GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 50, milli_Sec);

vTaskPrioritySet(ON_OFFSwitchTaskHandle,4);
```

Code of switches

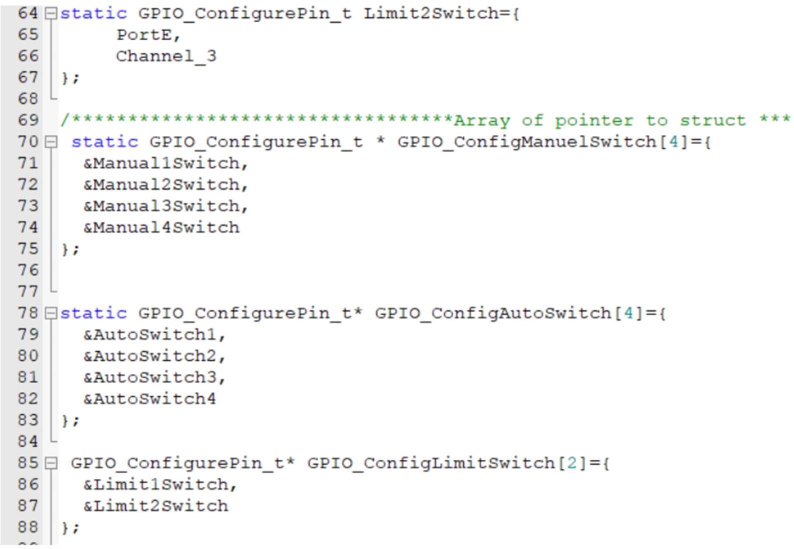
Structs:



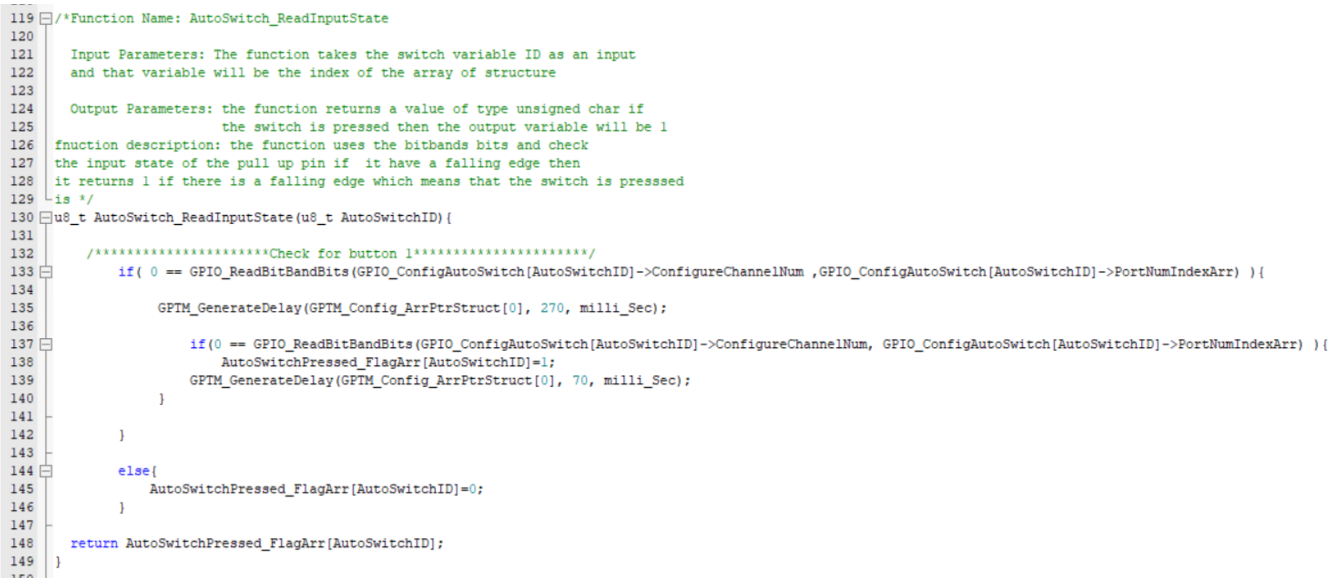
```
/**********************************AutomaticSwitch**********
static GPIO_ConfigurePin_t AutoSwitch1={
      PortC,
      Channel_4
};

static GPIO_ConfigurePin_t AutoSwitch2={
      PortC,
      Channel_5
};

static GPIO_ConfigurePin_t AutoSwitch3={
      PortC,
      Channel_6
};

static GPIO_ConfigurePin_t AutoSwitch4={
      PortC,
      Channel_7
};
/**********************************AutomaticSwitch**********
```



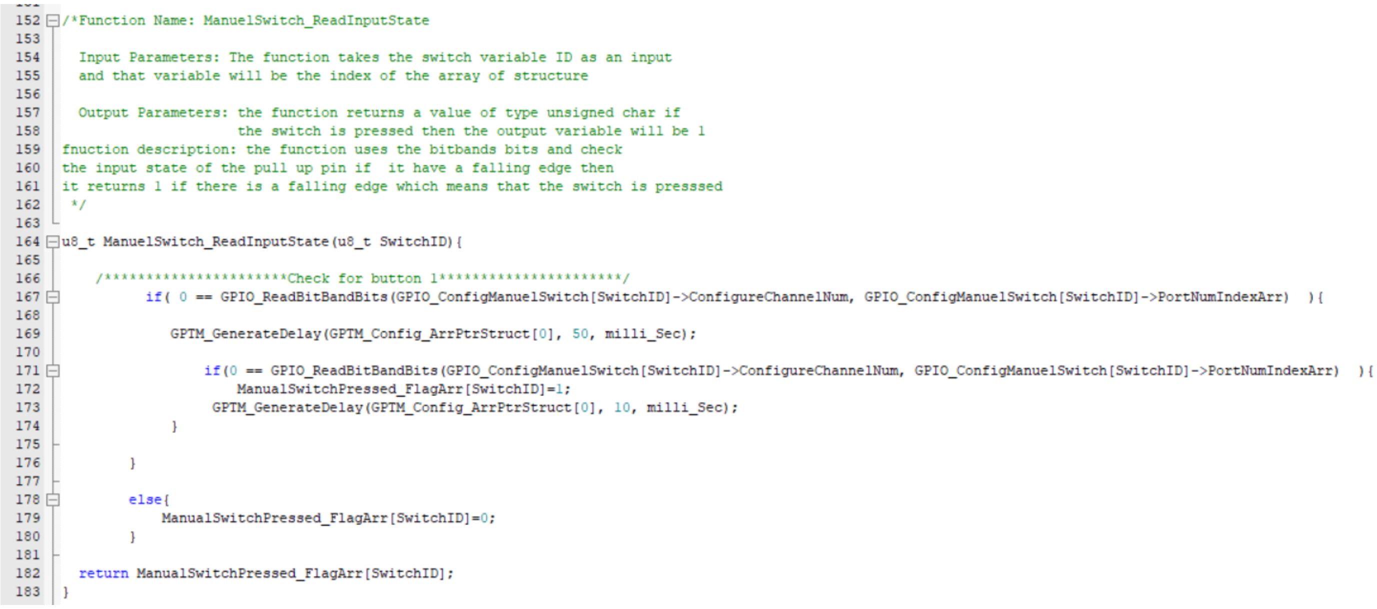
```
static GPIO_ConfigurePin_t ON_OFF_Switch={
      PortE,
      Channel_5
};

static GPIO_ConfigurePin_t Manual1Switch={
      PortA,
      Channel_2
};

static GPIO_ConfigurePin_t Manual2Switch={
      PortA,
      Channel_3
};


static GPIO_ConfigurePin_t Manual3Switch={
      PortA,
      Channel_4
};

static GPIO_ConfigurePin_t Manual4Switch={
      PortA,
      Channel_5
};

```



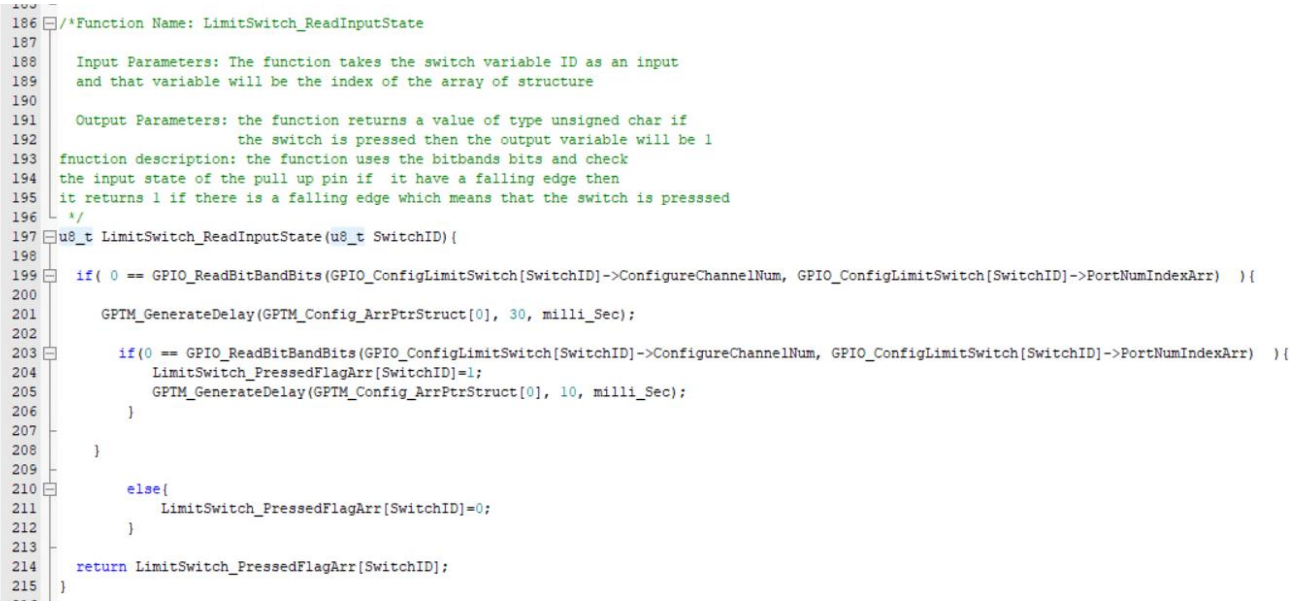
```
static GPIO_ConfigurePin_t Limit2Switch={
      PortE,
      Channel_3
};

/**********************************Array of pointer to struct ***
 static GPIO_ConfigurePin_t * GPIO_ConfigManuelSwitch[4]={
  &Manual1Switch,
  &Manual2Switch,
  &Manual3Switch,
  &Manual4Switch
};


static GPIO_ConfigurePin_t* GPIO_ConfigAutoSwitch[4]={
  &AutoSwitch1,
  &AutoSwitch2,
  &AutoSwitch3,
  &AutoSwitch4
};

 GPIO_ConfigurePin_t* GPIO_ConfigLimitSwitch[2]={
  &Limit1Switch,
  &Limit2Switch
};
```



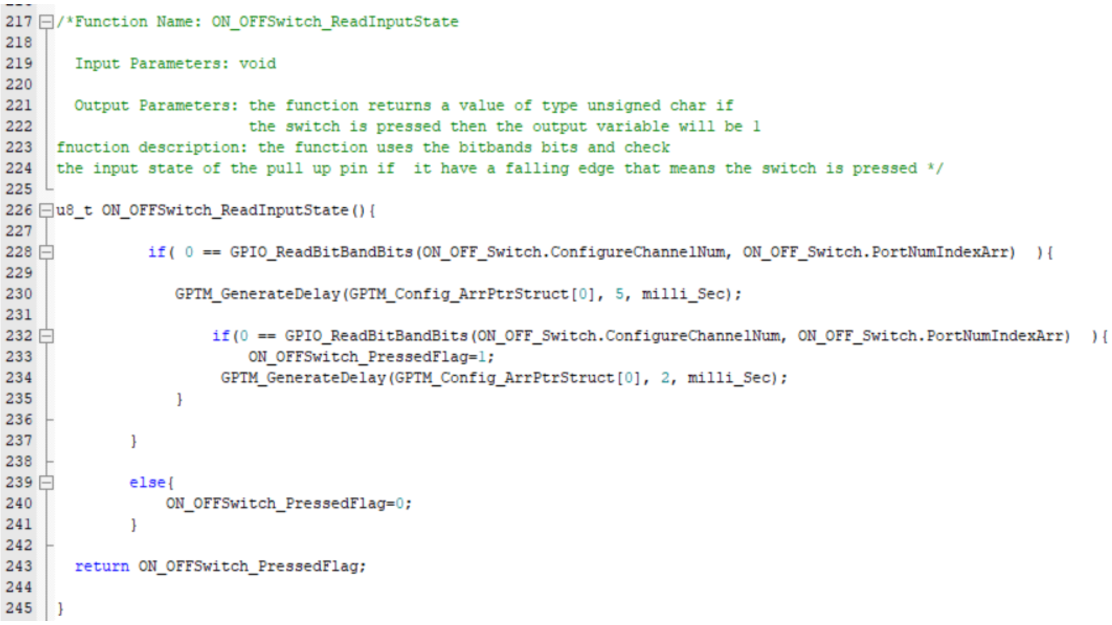
```c
/*Function Name: AutoSwitch_ReadInputState

  Input Parameters: The function takes the switch variable ID as an input
  and that variable will be the index of the array of structure

  Output Parameters: the function returns a value of type unsigned char if
                     the switch is pressed then the output variable will be 1
fnuction description: the function uses the bitbands bits and check
the input state of the pull up pin if  it have a falling edge then
it returns 1 if there is a falling edge which means that the switch is presssed
is */
u8_t AutoSwitch_ReadInputState(u8_t AutoSwitchID){

    /**********************Check for button 1**********************/
        if( 0 == GPIO_ReadBitBandBits(GPIO_ConfigAutoSwitch[AutoSwitchID]->ConfigureChannelNum ,GPIO_ConfigAutoSwitch[AutoSwitchID]->PortNumIndexArr) ){

             GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 270, milli_Sec);

                 if(0 == GPIO_ReadBitBandBits(GPIO_ConfigAutoSwitch[AutoSwitchID]->ConfigureChannelNum, GPIO_ConfigAutoSwitch[AutoSwitchID]->PortNumIndexArr) ){
                     AutoSwitchPressed_FlagArr[AutoSwitchID]=1;
                 GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 70, milli_Sec);
             }

        }

        else{
            AutoSwitchPressed_FlagArr[AutoSwitchID]=0;
        }

  return AutoSwitchPressed_FlagArr[AutoSwitchID];
}
```



```c
/*Function Name: ManuelSwitch_ReadInputState

  Input Parameters: The function takes the switch variable ID as an input
  and that variable will be the index of the array of structure

  Output Parameters: the function returns a value of type unsigned char if
                     the switch is pressed then the output variable will be 1
fnuction description: the function uses the bitbands bits and check
the input state of the pull up pin if  it have a falling edge then
it returns 1 if there is a falling edge which means that the switch is presssed
 */

u8_t ManuelSwitch_ReadInputState(u8_t SwitchID){

    /**********************Check for button 1**********************/
          if( 0 == GPIO_ReadBitBandBits(GPIO_ConfigManuelSwitch[SwitchID]->ConfigureChannelNum, GPIO_ConfigManuelSwitch[SwitchID]->PortNumIndexArr)  ){

             GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 50, milli_Sec);

                 if(0 == GPIO_ReadBitBandBits(GPIO_ConfigManuelSwitch[SwitchID]->ConfigureChannelNum, GPIO_ConfigManuelSwitch[SwitchID]->PortNumIndexArr)  ){
                     ManualSwitchPressed_FlagArr[SwitchID]=1;
                  GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 10, milli_Sec);
             }

        }

        else{
            ManualSwitchPressed_FlagArr[SwitchID]=0;
        }

  return ManualSwitchPressed_FlagArr[SwitchID];
}
```

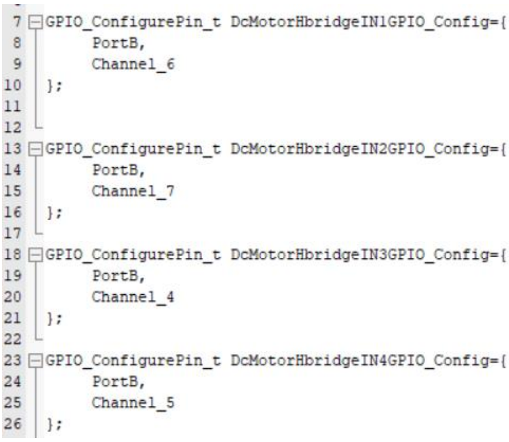


```
/*Function Name: LimitSwitch_ReadInputState

  Input Parameters: The function takes the switch variable ID as an input
  and that variable will be the index of the array of structure

  Output Parameters: the function returns a value of type unsigned char if
                     the switch is pressed then the output variable will be 1
fnuction description: the function uses the bitbands bits and check
the input state of the pull up pin if  it have a falling edge then
it returns 1 if there is a falling edge which means that the switch is presssed
 */
u8_t LimitSwitch_ReadInputState(u8_t SwitchID){

  if( 0 == GPIO_ReadBitBandBits(GPIO_ConfigLimitSwitch[SwitchID]->ConfigureChannelNum, GPIO_ConfigLimitSwitch[SwitchID]->PortNumIndexArr)  ){

     GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 30, milli_Sec);

       if(0 == GPIO_ReadBitBandBits(GPIO_ConfigLimitSwitch[SwitchID]->ConfigureChannelNum, GPIO_ConfigLimitSwitch[SwitchID]->PortNumIndexArr)  ){
           LimitSwitch_PressedFlagArr[SwitchID]=1;
           GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 10, milli_Sec);
        }

    }

        else{
            LimitSwitch_PressedFlagArr[SwitchID]=0;
        }

  return LimitSwitch_PressedFlagArr[SwitchID];
}
```

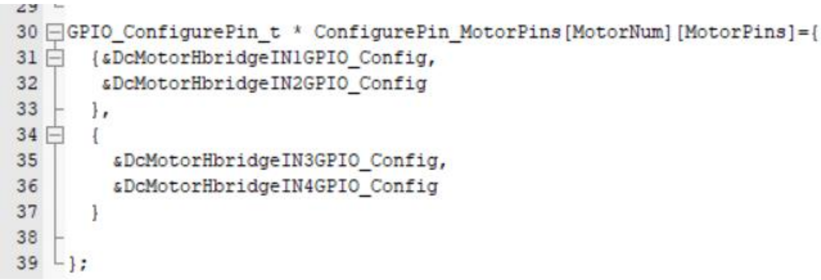


```
/*Function Name: ON_OFFSwitch_ReadInputState

  Input Parameters: void

  Output Parameters: the function returns a value of type unsigned char if
                     the switch is pressed then the output variable will be 1
fnuction description: the function uses the bitbands bits and check
the input state of the pull up pin if  it have a falling edge that means the switch is pressed */

u8_t ON_OFFSwitch_ReadInputState(){

          if( 0 == GPIO_ReadBitBandBits(ON_OFF_Switch.ConfigureChannelNum, ON_OFF_Switch.PortNumIndexArr)  ){

             GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 5, milli_Sec);

                if(0 == GPIO_ReadBitBandBits(ON_OFF_Switch.ConfigureChannelNum, ON_OFF_Switch.PortNumIndexArr)  ){
                    ON_OFFSwitch_PressedFlag=1;
                 GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 2, milli_Sec);
             }

        }

        else{
            ON_OFFSwitch_PressedFlag=0;
        }

  return ON_OFFSwitch_PressedFlag;

}
```

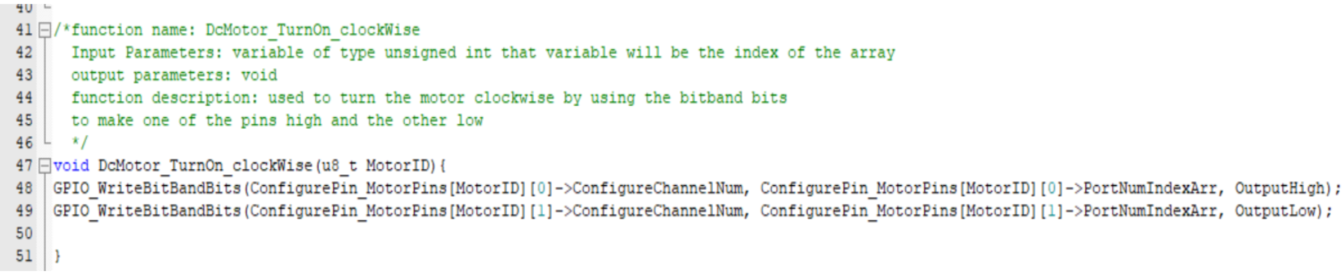
Code of dc motor



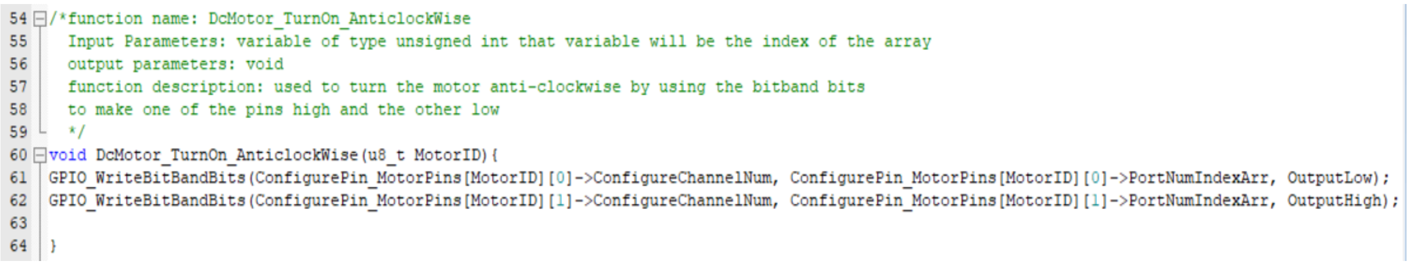
```c
GPIO_ConfigurePin_t DcMotorHbridgeIN1GPIO_Config={
      PortB,
      Channel_6
};


GPIO_ConfigurePin_t DcMotorHbridgeIN2GPIO_Config={
      PortB,
      Channel_7
};

GPIO_ConfigurePin_t DcMotorHbridgeIN3GPIO_Config={
      PortB,
      Channel_4
};

GPIO_ConfigurePin_t DcMotorHbridgeIN4GPIO_Config={
      PortB,
      Channel_5
};
```



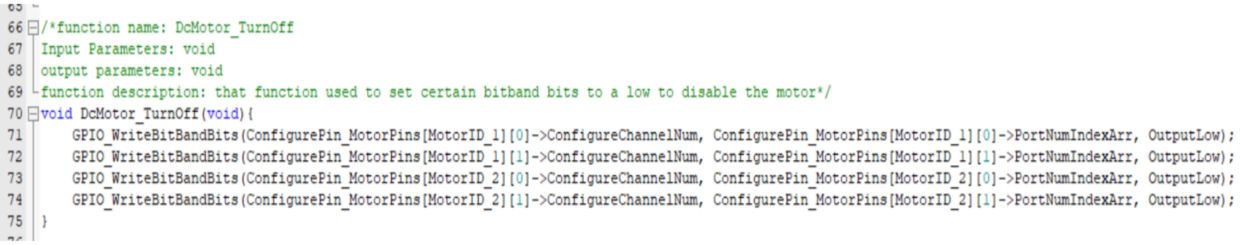
```c
GPIO_ConfigurePin_t * ConfigurePin_MotorPins[MotorNum][MotorPins]={
  {&DcMotorHbridgeIN1GPIO_Config,
   &DcMotorHbridgeIN2GPIO_Config
  },
  {
    &DcMotorHbridgeIN3GPIO_Config,
    &DcMotorHbridgeIN4GPIO_Config
  }

};
```



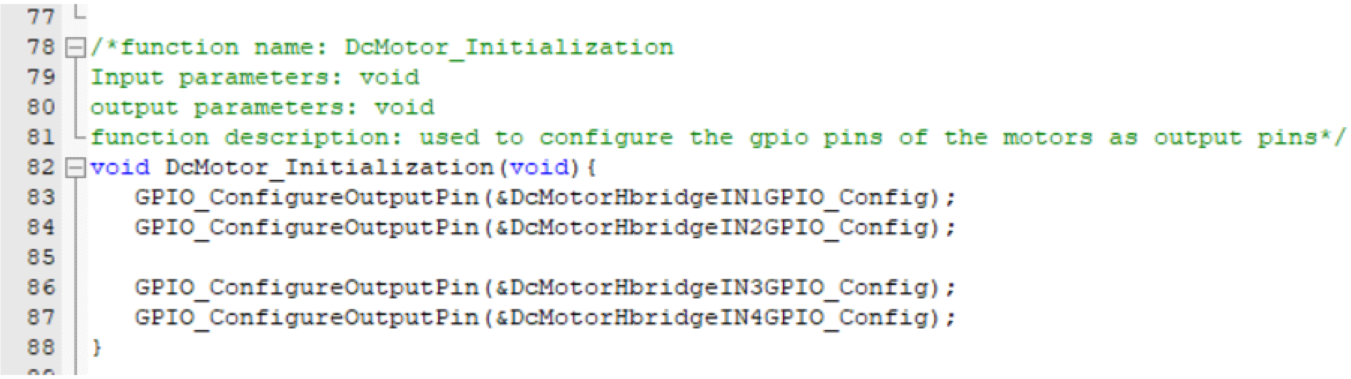
```c
/*function name: DcMotor_TurnOn_clockWise
  Input Parameters: variable of type unsigned int that variable will be the index of the array
  output parameters: void
  function description: used to turn the motor clockwise by using the bitband bits
  to make one of the pins high and the other low
  */
void DcMotor_TurnOn_clockWise(u8_t MotorID){
GPIO_WriteBitBandBits(ConfigurePin_MotorPins[MotorID][0]->ConfigureChannelNum, ConfigurePin_MotorPins[MotorID][0]->PortNumIndexArr, OutputHigh);
GPIO_WriteBitBandBits(ConfigurePin_MotorPins[MotorID][1]->ConfigureChannelNum, ConfigurePin_MotorPins[MotorID][1]->PortNumIndexArr, OutputLow);

}
```



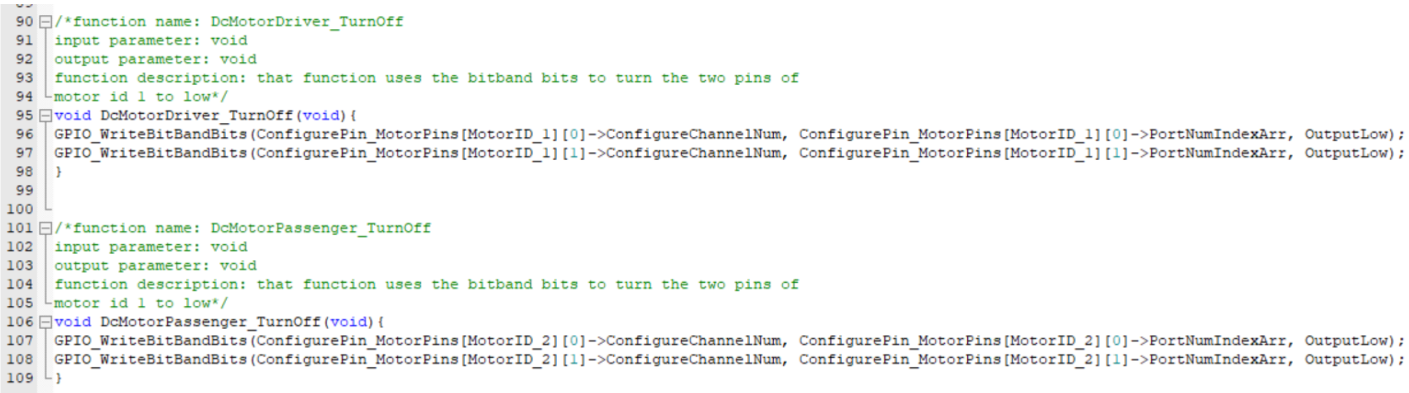
```c
/*function name: DcMotor_TurnOn_AnticlockWise
  Input Parameters: variable of type unsigned int that variable will be the index of the array
  output parameters: void
  function description: used to turn the motor anti-clockwise by using the bitband bits
  to make one of the pins high and the other low
  */
void DcMotor_TurnOn_AnticlockWise(u8_t MotorID){
GPIO_WriteBitBandBits(ConfigurePin_MotorPins[MotorID][0]->ConfigureChannelNum, ConfigurePin_MotorPins[MotorID][0]->PortNumIndexArr, OutputLow);
GPIO_WriteBitBandBits(ConfigurePin_MotorPins[MotorID][1]->ConfigureChannelNum, ConfigurePin_MotorPins[MotorID][1]->PortNumIndexArr, OutputHigh);

}
```



```
/*function name: DcMotor_TurnOff
Input Parameters: void
output parameters: void
function description: that function used to set certain bitband bits to a low to disable the motor*/
void DcMotor_TurnOff(void){
    GPIO_WriteBitBandBits(ConfigurePin_MotorPins[MotorID_1][0]->ConfigureChannelNum, ConfigurePin_MotorPins[MotorID_1][0]->PortNumIndexArr, OutputLow);
    GPIO_WriteBitBandBits(ConfigurePin_MotorPins[MotorID_1][1]->ConfigureChannelNum, ConfigurePin_MotorPins[MotorID_1][1]->PortNumIndexArr, OutputLow);
    GPIO_WriteBitBandBits(ConfigurePin_MotorPins[MotorID_2][0]->ConfigureChannelNum, ConfigurePin_MotorPins[MotorID_2][0]->PortNumIndexArr, OutputLow);
    GPIO_WriteBitBandBits(ConfigurePin_MotorPins[MotorID_2][1]->ConfigureChannelNum, ConfigurePin_MotorPins[MotorID_2][1]->PortNumIndexArr, OutputLow);
}
```

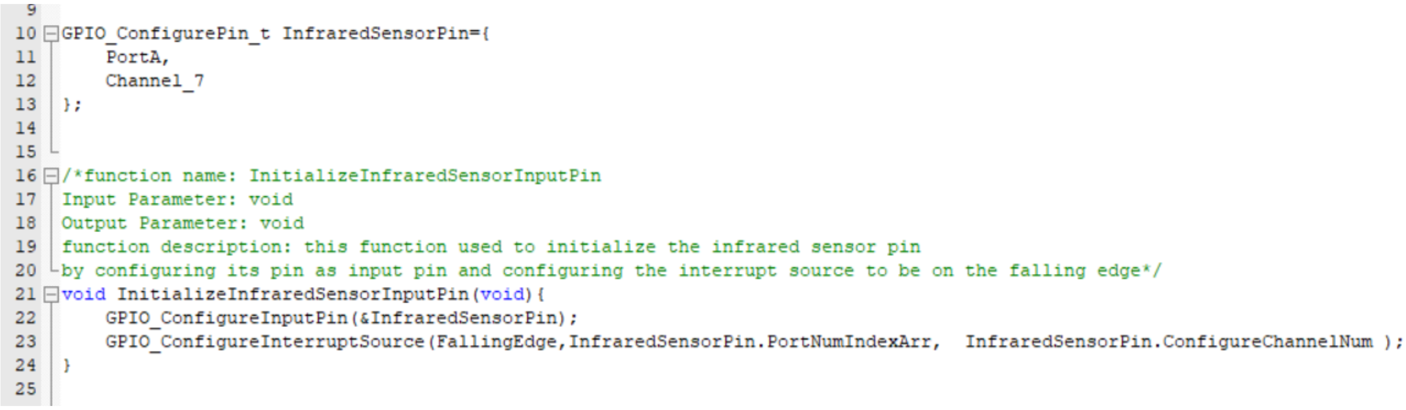


```
/*function name: DcMotor_Initialization
Input parameters: void
output parameters: void
function description: used to configure the gpio pins of the motors as output pins*/
void DcMotor_Initialization(void){
    GPIO_ConfigureOutputPin(&DcMotorHbridgeIN1GPIO_Config);
    GPIO_ConfigureOutputPin(&DcMotorHbridgeIN2GPIO_Config);

    GPIO_ConfigureOutputPin(&DcMotorHbridgeIN3GPIO_Config);
    GPIO_ConfigureOutputPin(&DcMotorHbridgeIN4GPIO_Config);
}
```

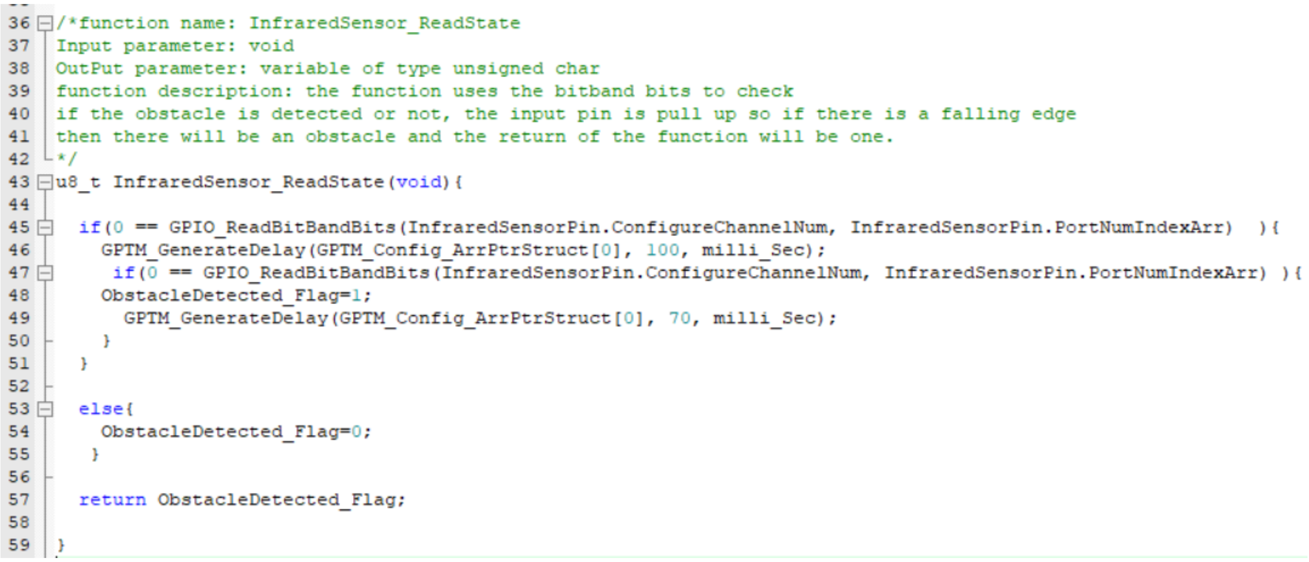


```
/*function name: DcMotorDriver_TurnOff
input parameter: void
output parameter: void
function description: that function uses the bitband bits to turn the two pins of
motor id 1 to low*/
void DcMotorDriver_TurnOff(void){
GPIO_WriteBitBandBits(ConfigurePin_MotorPins[MotorID_1][0]->ConfigureChannelNum, ConfigurePin_MotorPins[MotorID_1][0]->PortNumIndexArr, OutputLow);
GPIO_WriteBitBandBits(ConfigurePin_MotorPins[MotorID_1][1]->ConfigureChannelNum, ConfigurePin_MotorPins[MotorID_1][1]->PortNumIndexArr, OutputLow);
}


/*function name: DcMotorPassenger_TurnOff
input parameter: void
output parameter: void
function description: that function uses the bitband bits to turn the two pins of
motor id 1 to low*/
void DcMotorPassenger_TurnOff(void){
GPIO_WriteBitBandBits(ConfigurePin_MotorPins[MotorID_2][0]->ConfigureChannelNum, ConfigurePin_MotorPins[MotorID_2][0]->PortNumIndexArr, OutputLow);
GPIO_WriteBitBandBits(ConfigurePin_MotorPins[MotorID_2][1]->ConfigureChannelNum, ConfigurePin_MotorPins[MotorID_2][1]->PortNumIndexArr, OutputLow);
}
```

Code of infrared sensor

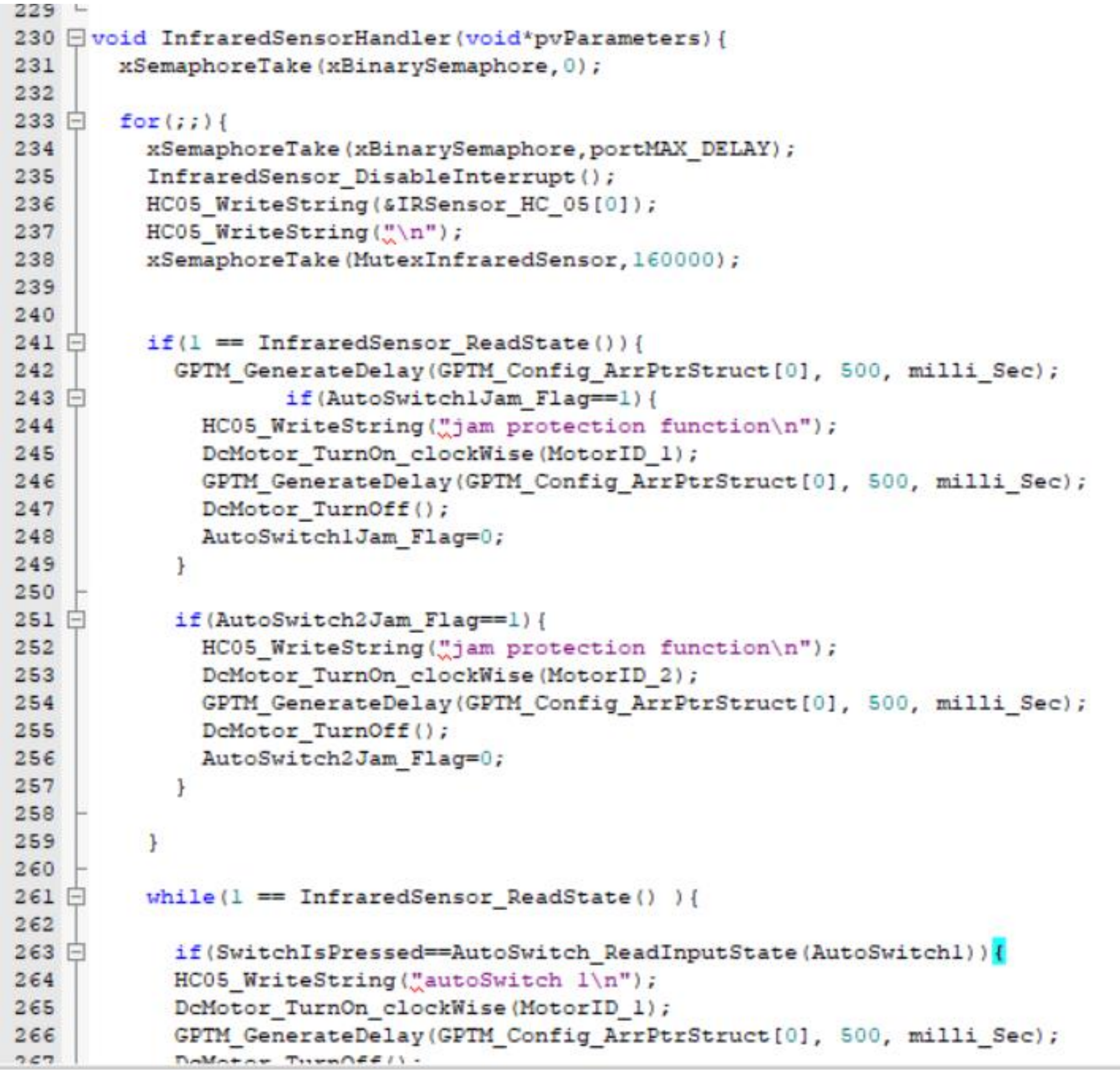


```

GPIO_ConfigurePin_t InfraredSensorPin={
    PortA,
    Channel_7
};


/*function name: InitializeInfraredSensorInputPin
Input Parameter: void
Output Parameter: void
function description: this function used to initialize the infrared sensor pin
by configuring its pin as input pin and configuring the interrupt source to be on the falling edge*/
void InitializeInfraredSensorInputPin(void){
    GPIO_ConfigureInputPin(&InfraredSensorPin);
    GPIO_ConfigureInterruptSource(FallingEdge,InfraredSensorPin.PortNumIndexArr,  InfraredSensorPin.ConfigureChannelNum );
}

```



```
/*function name: InfraredSensor_ReadState
Input parameter: void
OutPut parameter: variable of type unsigned char
function description: the function uses the bitband bits to check
if the obstacle is detected or not, the input pin is pull up so if there is a falling edge
then there will be an obstacle and the return of the function will be one.
*/
u8_t InfraredSensor_ReadState(void){

  if(0 == GPIO_ReadBitBandBits(InfraredSensorPin.ConfigureChannelNum, InfraredSensorPin.PortNumIndexArr)  ){
    GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 100, milli_Sec);
     if(0 == GPIO_ReadBitBandBits(InfraredSensorPin.ConfigureChannelNum, InfraredSensorPin.PortNumIndexArr) ){
    ObstacleDetected_Flag=1;
      GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 70, milli_Sec);
    }
  }

  else{
    ObstacleDetected_Flag=0;
   }

  return ObstacleDetected_Flag;

}
```

Corner cases

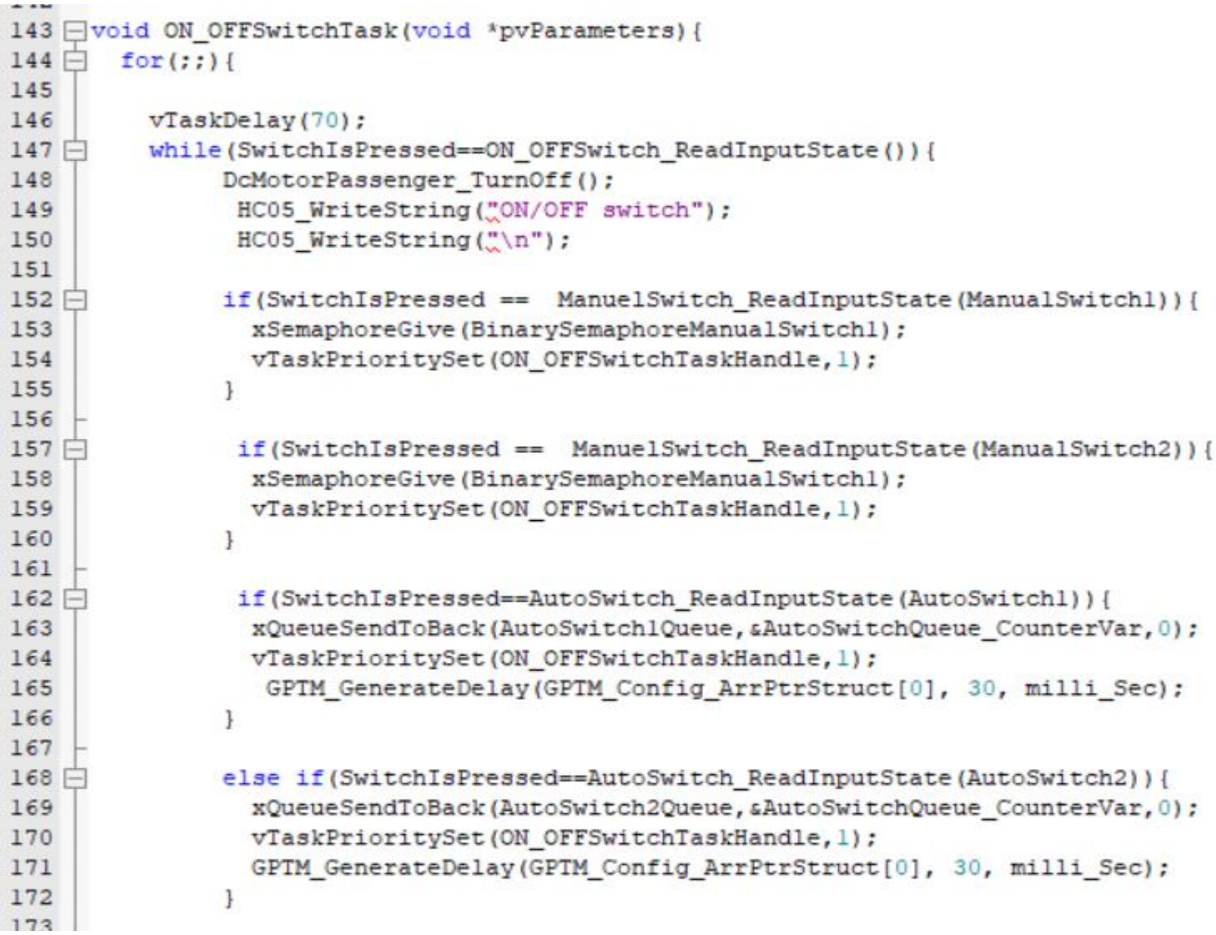


```
void InfraredSensorHandler(void*pvParameters){
 xSemaphoreTake(xBinarySemaphore,0);

 for(;;){
  xSemaphoreTake(xBinarySemaphore,portMAX_DELAY);
  InfraredSensor_DisableInterrupt();
  HC05_WriteString(&IRSensor_HC_05[0]);
  HC05_WriteString("\n");
  xSemaphoreTake(MutexInfraredSensor,160000);


  if(1 == InfraredSensor_ReadState()){
   GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 500, milli_Sec);
       if(AutoSwitch1Jam_Flag==1){
    HC05_WriteString("jam protection function\n");
    DcMotor_TurnOn_clockWise(MotorID_1);
    GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 500, milli_Sec);
    DcMotor_TurnOff();
    AutoSwitch1Jam_Flag=0;
   }

   if(AutoSwitch2Jam_Flag==1){
    HC05_WriteString("jam protection function\n");
    DcMotor_TurnOn_clockWise(MotorID_2);
    GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 500, milli_Sec);
    DcMotor_TurnOff();
    AutoSwitch2Jam_Flag=0;
   }

  }

  while(1 == InfraredSensor_ReadState() ){

   if(SwitchIsPressed==AutoSwitch_ReadInputState(AutoSwitch1)){
   HC05_WriteString("autoSwitch 1\n");
   DcMotor_TurnOn_clockWise(MotorID_1);
   GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 500, milli_Sec);
   DcMotor_TurnOff();
```

- The infrared sensor already works with interrupts and has its callback function and handler so when the interrupt occurs the CPU executes the call back function and gives the semaphore so the scheduler assigns the handler task to the CPU. **The corner case is:** at the start of the handler the interrupt related to the infrared is disabled and at the end of the handler that interrupt is enabled.

The reason for doing that is when there is an obstacle that remains a long time the CPU keeps receiving many interrupt requests because there will be falling levels as long as there is an obstacle so the motor will keep moving in certain direction because of jamming. That behavior is not needed, the needed behavior, is when there is an obstacle in front of the infrared sensor that remains a long time the jamming protection (moving the motor in clock wise direction for 500 milli second) will occur once.

- The same concept applies to limit switches as it works also with interrupts.

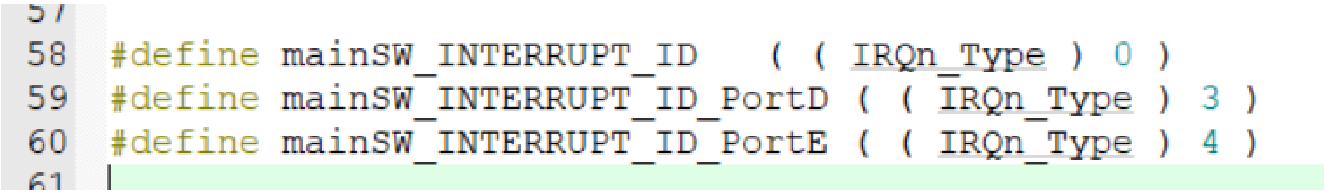


```c
void ON_OFFSwitchTask(void *pvParameters){
  for(;;){

    vTaskDelay(70);
    while(SwitchIsPressed==ON_OFFSwitch_ReadInputState()){
          DcMotorPassenger_TurnOff();
           HC05_WriteString("ON/OFF switch");
           HC05_WriteString("\n");

          if(SwitchIsPressed ==  ManuelSwitch_ReadInputState(ManualSwitch1)){
            xSemaphoreGive(BinarySemaphoreManualSwitch1);
            vTaskPrioritySet(ON_OFFSwitchTaskHandle,1);
          }

           if(SwitchIsPressed ==  ManuelSwitch_ReadInputState(ManualSwitch2)){
            xSemaphoreGive(BinarySemaphoreManualSwitch1);
            vTaskPrioritySet(ON_OFFSwitchTaskHandle,1);
          }

           if(SwitchIsPressed==AutoSwitch_ReadInputState(AutoSwitch1)){
            xQueueSendToBack(AutoSwitch1Queue,&AutoSwitchQueue_CounterVar,0);
            vTaskPrioritySet(ON_OFFSwitchTaskHandle,1);
             GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 30, milli_Sec);
          }

          else if(SwitchIsPressed==AutoSwitch_ReadInputState(AutoSwitch2)){
            xQueueSendToBack(AutoSwitch2Queue,&AutoSwitchQueue_CounterVar,0);
            vTaskPrioritySet(ON_OFFSwitchTaskHandle,1);
            GPTM_GenerateDelay(GPTM_Config_ArrPtrStruct[0], 30, milli_Sec);
          }
```

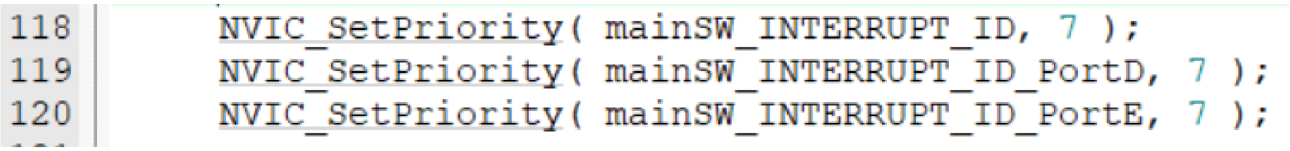
- The on off switch task is a higher priority than open window manually task and open window automatically task so if the driver pressed manual or automatic switch, after giving the semaphore the priority of the on off switch task will be decreased to allow the lower priority task which is open window manually and open window automatically to execute and in these tasks the priority of the on off switch will be increased again.

Challenges

- At the beginning, there was a problem in portEND_SWITCHING_ISR() which is called in the callback function (it did not make switch to the handler)
- Solution: I added these macros, each macro is for a specific port and the function NVIC_SetPriority() the second parameter of this function is the same number of the configMAX_Priorities in the free artos config file



```
#define mainSW_INTERRUPT_ID   ( ( IRQn_Type ) 0 )
#define mainSW_INTERRUPT_ID_PortD ( ( IRQn_Type ) 3 )
#define mainSW_INTERRUPT_ID_PortE ( ( IRQn_Type ) 4 )
```



```
    NVIC_SetPriority( mainSW_INTERRUPT_ID, 7 );
    NVIC_SetPriority( mainSW_INTERRUPT_ID_PortD, 7 );
    NVIC_SetPriority( mainSW_INTERRUPT_ID_PortE, 7 );
```