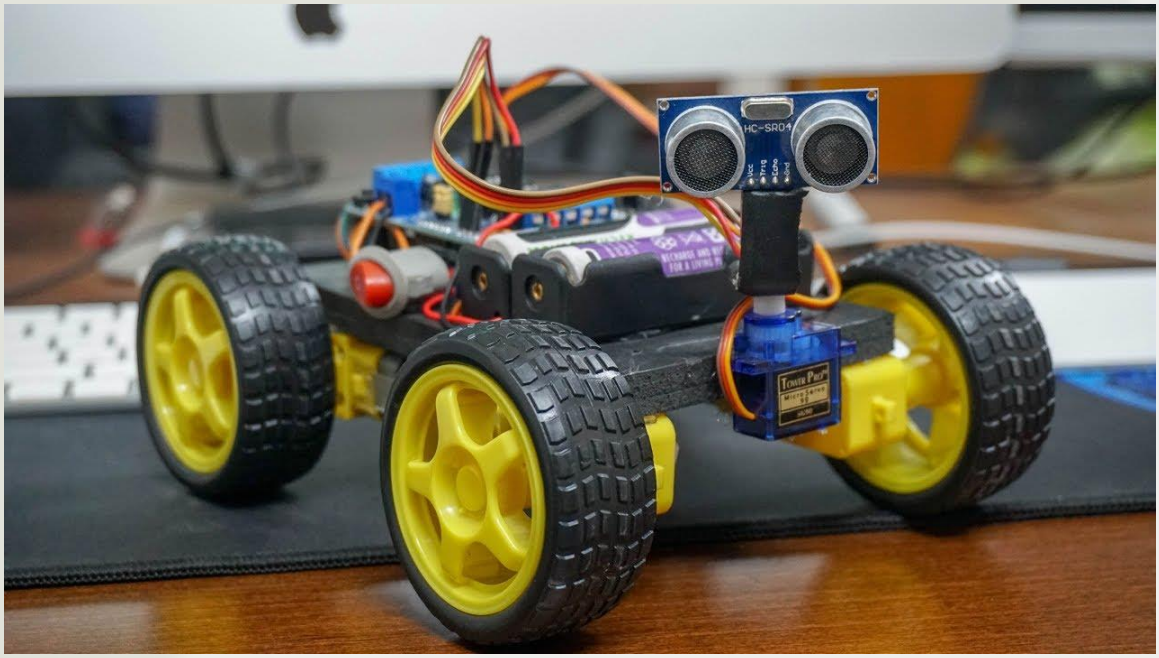
Sprints

Obstacle Avoidance Robot V1.0



By: Team 7

-Kareem Magdy -Moustafa Abdelrahim -Mohamed Sayed

-Mohamed Abdulwhab -Nada Abdelazim

Sprints

Table of contents

Project Introduction

This is a document to design a four-wheeled robot that avoids any obstacle in front of it.

After setting the default direction of rotation and waiting for five seconds, the robot moves with varying speeds based on the distance from any detected obstacle. The LCD displays information about the direction of movement, distance, and status of the robot.

Hardware Components:

The components are: ATmega32 microcontroller, four motors, one ultrasonic sensor, push-button, a keypad with two buttons and LCD.

Ultrasonic Sensor Connections:

The sensory component of the robot features Trig connected to Port B pin3 and Echo connected to Port B pin2.

System Requirements:

The robot starts at 0 speed.

The default rotation direction is to the right.

Pressing the first button on the keypad starts the robot, and the second button stops it.

After pressing start, the LCD displays a message for selecting the default rotation direction of the robot.

The robot waits for five seconds to choose between left and right.

The LCD displays speed and moving direction, as well as the object distance. Movement directions include forward, backwards, rotating, or stopped.

Default rotation direction changes based on pressing PBUTTON0 once or twice.

After five seconds, the robot starts moving at 30% speed.

Robot’s movement:

Robot Start up movement:

The robot initiates movement two seconds after the default direction of rotation is set. The LCD displays a welcome message of “Set Def Rot” when the robot starts.

Robot Movement with no Obstacles or obstacles at distance more 70 cm:

The robot moves forward at 30% speed for the first five seconds. After that, it moves with 50% of the speed unless there is an obstacle at a distance less than 70 centimeters.

Robot Movement with Obstacles at 70 to 30 cm distance:

If there is an obstacle at a distance between 30 and 70 centimeters, the robot decreases its speed to 30%, and the LCD displays this change.

Robot Movement with Obstacles at 20 to 30 cm distance:

When the robot detects an obstacle between 20 and 30 centimeters, it stops and rotates 90 degrees either to the right or left, based on the chosen configuration. This change in direction is displayed on the LCD.

Robot Movement with Obstacles at less than 20 cm distance:

If there is an obstacle at a distance between 0 and 20 centimeters, the robot stops and moves backwards with 30% speed until it is beyond 20 cm to 30 cm distance.

Obstacle detection:

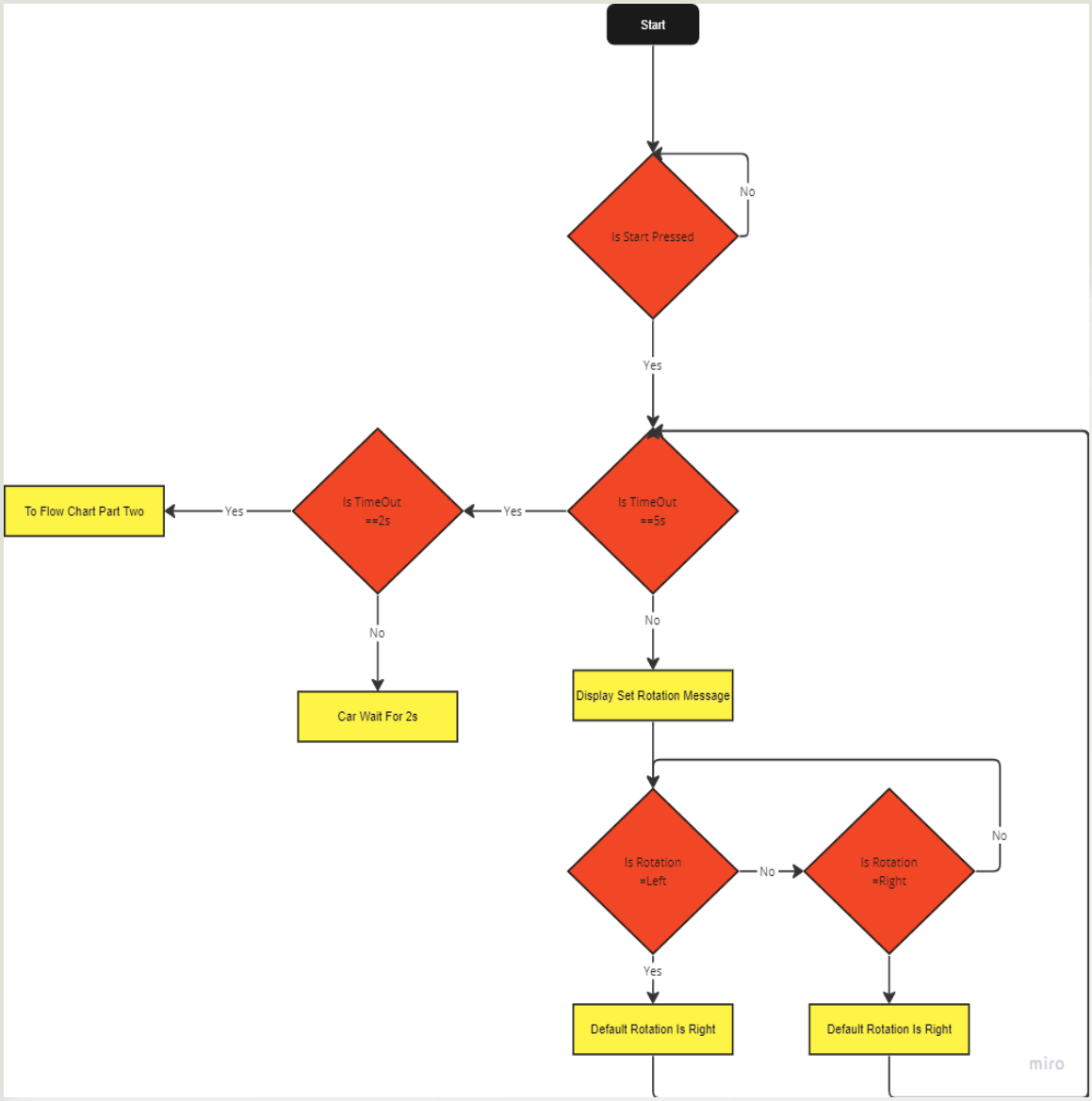
If the robot rotates 360 degrees without detecting distances greater than 20 centimeters, it stops. The LCD displays information that the robot has stopped. The robot checks if an obstacle was removed every three seconds and moves in the direction of the furthest object.

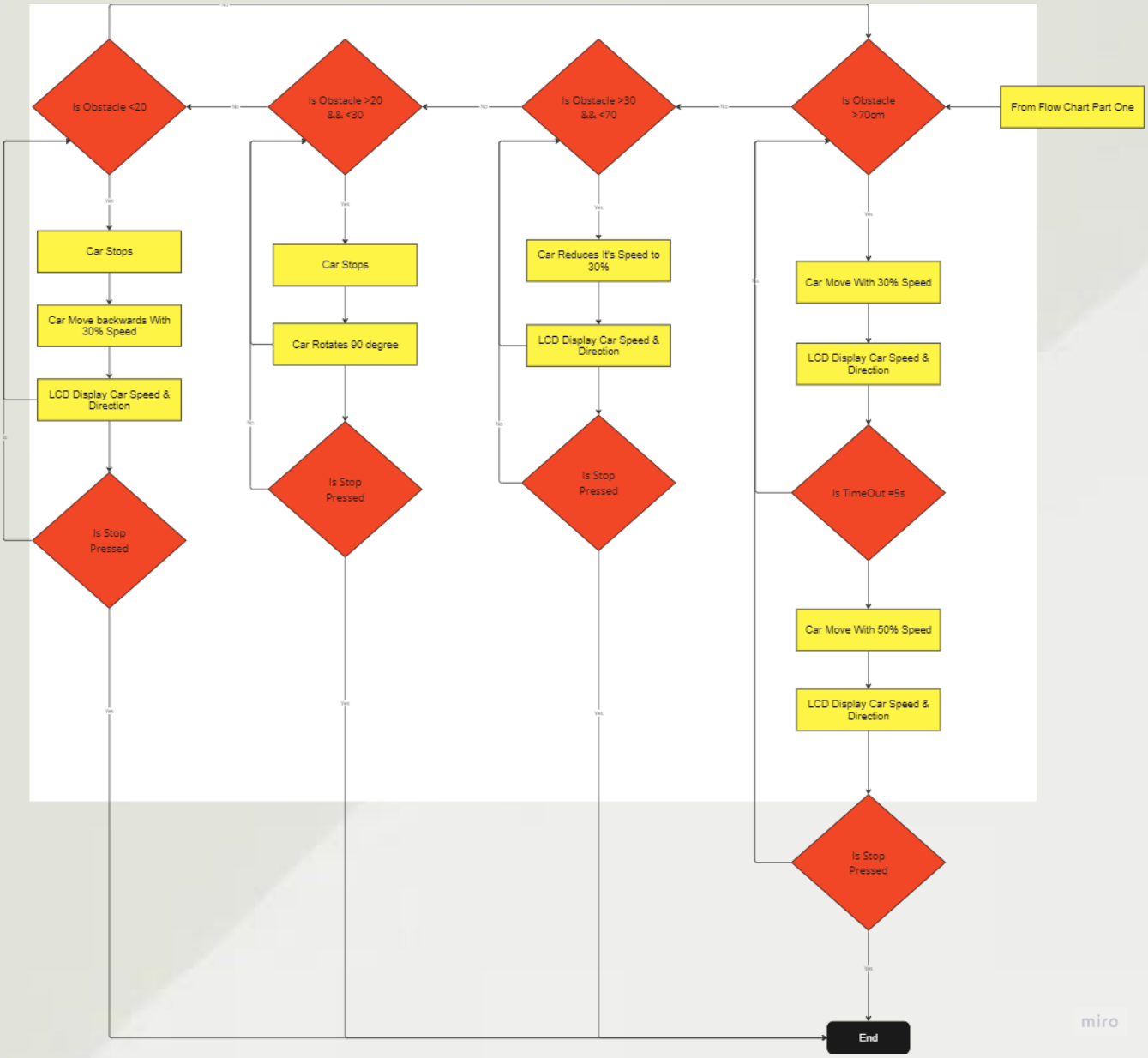
Sprints

Project

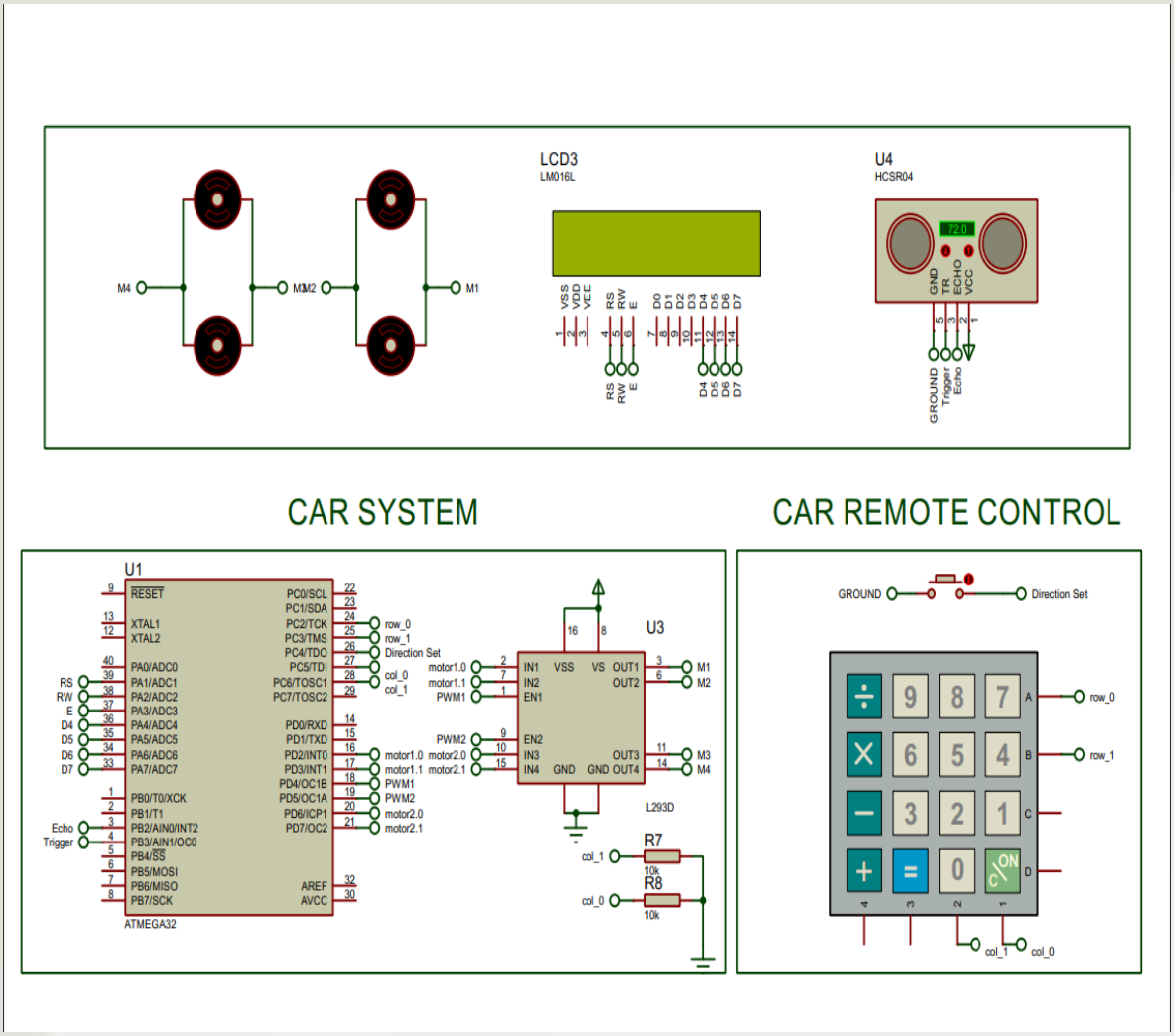
Sprints

Main project flow chart



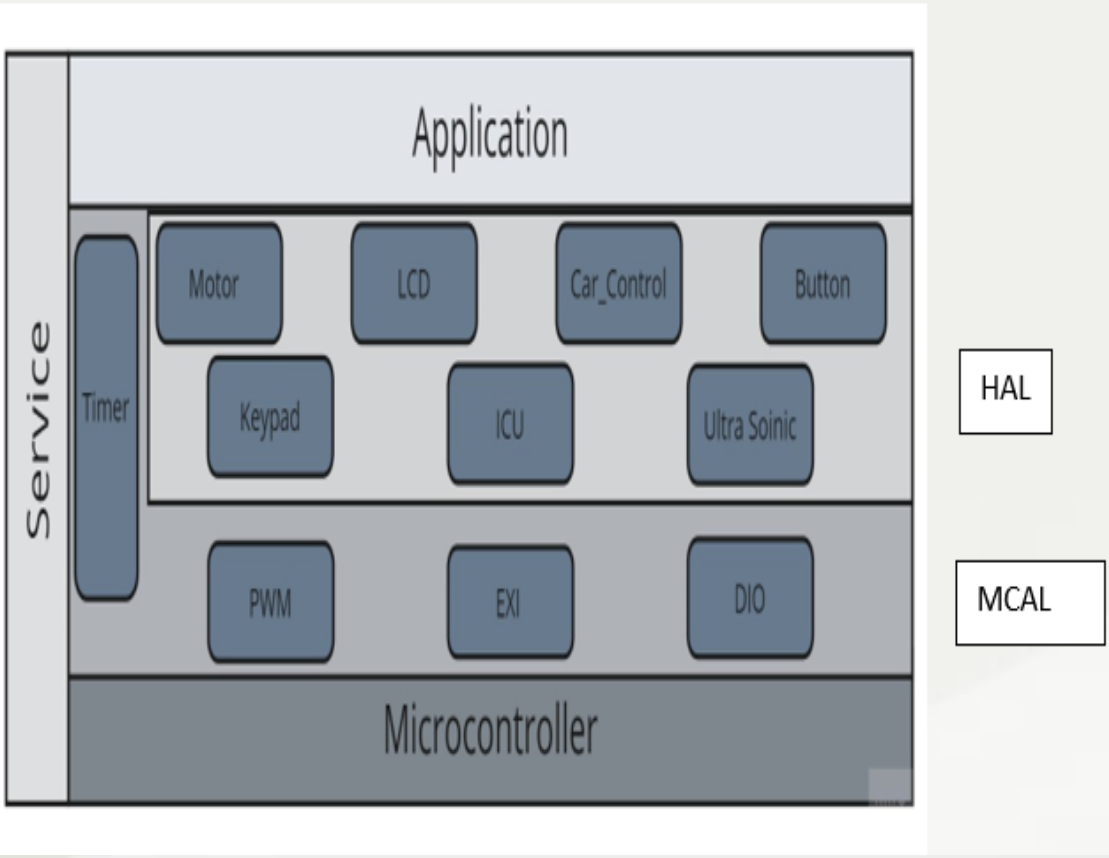

From Flow Chart Part One
Is Obstacle >70cm
Is Obstacle >30 && <70
Is Obstacle >20 && <30
Is Obstacle <20
Car Move With 30% Speed
LCD Display Car Speed & Direction
Is TimeOut =5s
Car Move With 50% Speed
LCD Display Car Speed & Direction
Is Stop Pressed
Car Reduces It's Speed to 30%
LCD Display Car Speed & Direction
Is Stop Pressed
Car Stops
Car Rotates 90 degree
Is Stop Pressed
Car Stops
Car Move backwards With 30% Speed
LCD Display Car Speed & Direction
Is Stop Pressed
End
miro

Proteus design



HIGH LEVEL DESIGN

Layered architecture



Layers description:

(1)- Application layer:

Contains functions calls to implement the main project.

(2)- HAL:

Contains Drivers of the external electronic devices which will be connected to the microcontroller and the system overall.

(3)-MCAL: “Microcontroller Abstraction Layer”

Contains interfaces of the microcontroller’s peripherals.

Modules descriptions

Modules descriptions:

HALL LAYER:

1. motor module: Motor module is used to control the speed and direction of a DC motor.

2. Car control module:

3. ICU module: Input Capture Unit module is used to measure the time between two events or pulses.

4. Ultrasonic sensor module: Uses an ultrasonic sensor to detect the location and distance of objects within the robot's surroundings.

5. LCD module: Liquid Crystal Display module is used to display text and graphics on a screen.

6. Keypad module: Keypad is used to read input from a matrix of buttons arranged in rows and columns.

7. Button: Button module is used to detect when a button has been pressed or released.

MCAL LAYER:

8. PWM: Pulse Width Modulation module is used to generate analog signals by varying the duty cycle of a digital signal. In this project the PWM module is used to control the speed of the motors.

9. External Interrupt module: Interrupt module is used to handle external events or interruptions that occur during program execution.

10. DIO module: Digital Input/Output module is used to control and read digital signals from external devices.

11. Timer module: Timer module is used to generate precise time delays or periodic interruptions.

Driver's documentation

1. motor module:

```
void Car_Moving_BWD(void);
void Car_Motors_init(void);
void Car_Moving_FWD(void);
void Car_Rotating(void);
void Car_Stop(void);
```

2. Car control module:

```
void rotate_90degree_Left (void);
void rotate_90degree_calculation (void);
void pwm(float a_speed);
void rotate_90degree_Right (void);
```

3. ICU module:

```
void Icu_Enable(EN_int_t EXInt );
void Icu_Disable(EN_int_t EXInt);
void Icu_Trigger(EN_int_t EXInt,EN_trig trigger);
void Icu_SetCallback(EN_int_t EXInt,void(*fptr)(void));
```

4. Ultrasonic sensor module:

```
void Ultrasonic_init_SW(void);

void Ultrasonic_GetDistance(uint32_t*dis);
```

5. LCD module:

```
void LCD_WRITE_COMMAND(uint8_t a_COMMAND);
void LCD_WRITE_DATA(uint8_t a_DATA);

void LCD_INIT(void);
void LCD_Write_String(uint8_t*a_String);
void LCD_Write_Number(uint32_t a_number);
void LCD_Clear(void);
void LCD_GoTo(uint8_t a_line,uint8_t a_cell);
void LCD_Write_Charecter(uint8_t a_char);
void LCD_Create_Charecter(uint8_t*a_Pattern,uint8_t a_Adress);
```

6. Keypad module:

```
void keypad_init (void);
void keypad_get_value  (uint8_t *value)
```

7. Button module:

```
BUTTON_ERROR_TYPE Button_INIT(DIO_PIN_TYPE PIN);
BUTTON_ERROR_TYPE Button_read(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE*VOLT);
```

8. PWM: Pulse

```
void PWM_duty(uint8_t duty);
```

9. External Interrupt module:

```
EN_int__error_t EXI_Enable (EN_int_t Interrupt);
EN_int__error_t EXI_Disable (EN_int_t Interrupt);
EN_int__error_t EXI_Trigger(EN_int_t Interrupt,EN_trig trigger);
void EXI_SetCallBack(EN_int_t Interrupt,void(*ptrf)(void));
```

10. DIO module:

```
DIO_ERROR_TYPE DIO_INITPIN(DIO_PIN_TYPE PIN,DIO_PINSTATUS_TYPE STATUS);
DIO_ERROR_TYPE DIO_WRITEPIN(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE VOLTAGE);
DIO_ERROR_TYPE DIO_READPIN(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE* VOLT);
void DIO_WritePort(DIO_PORT_TYPE port ,uint8_t data);
void DIO_TogglePin(DIO_PIN_TYPE pin);
void DIO_Init_MCU(void);
void DIO_Init(void);
```

11. Timer module:

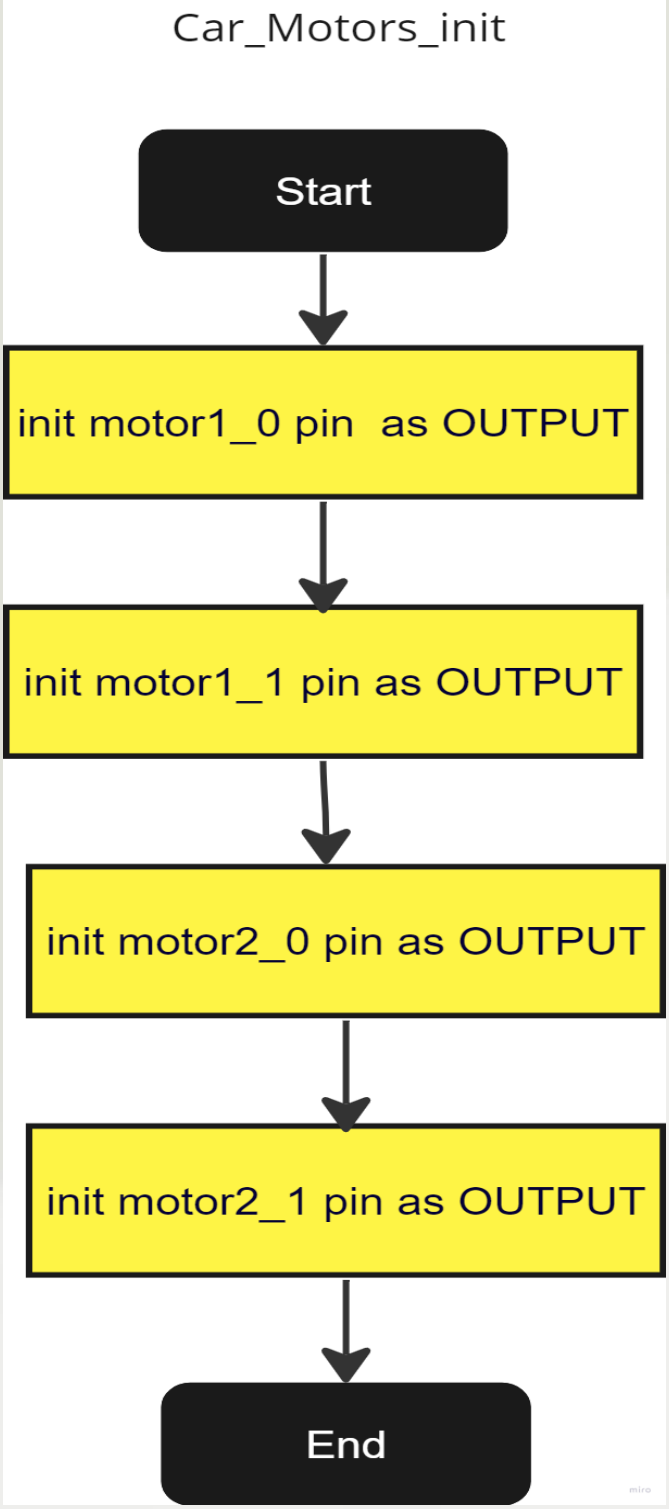
```
Timer_ErrorStatus TIMER_2_init( );
Timer_ErrorStatus TIMER_2_start();
void TIMER_2_stop(void);
Timer_ErrorStatus TIMER_2_setIntialValue(uint8_t a_value);
Timer_ErrorStatus TIMER_2_OvfNum(double overflow);
void TIMER_2_DELAY_MS(double _delay);
void TIMER_2_INT();
void TIMER_2_SetCallBack(void(*ptrf)(void));
```

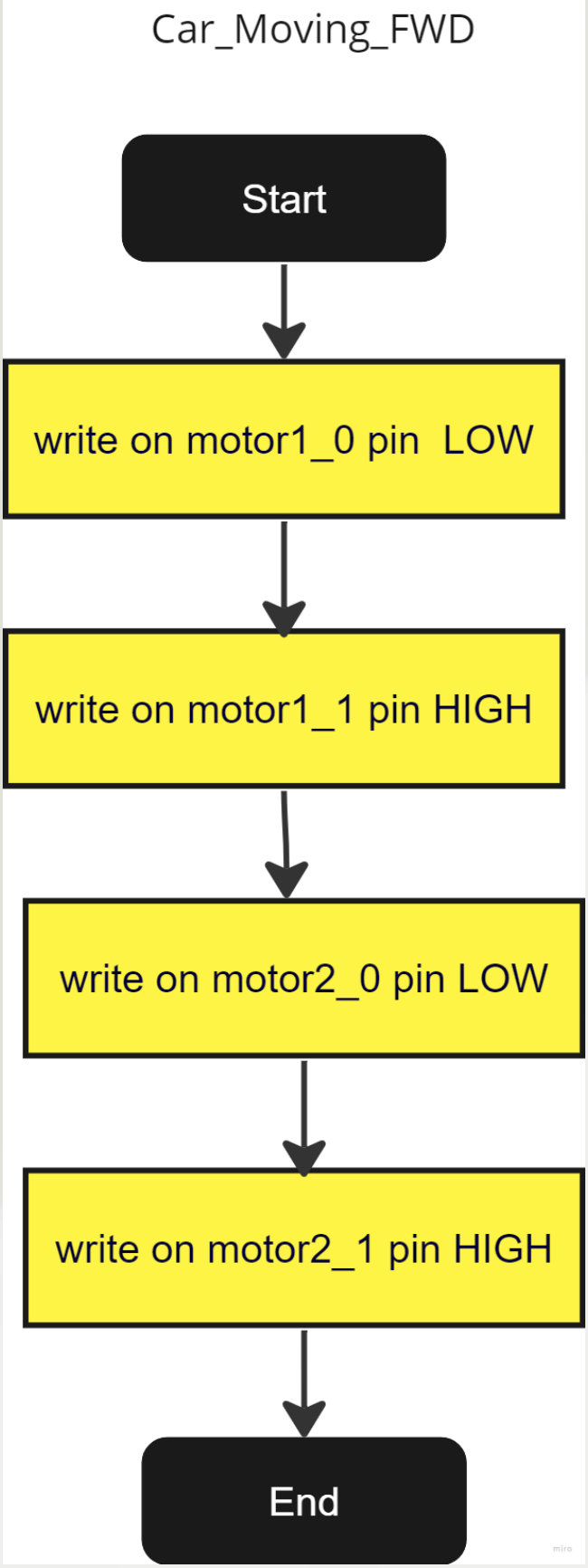
Sprints

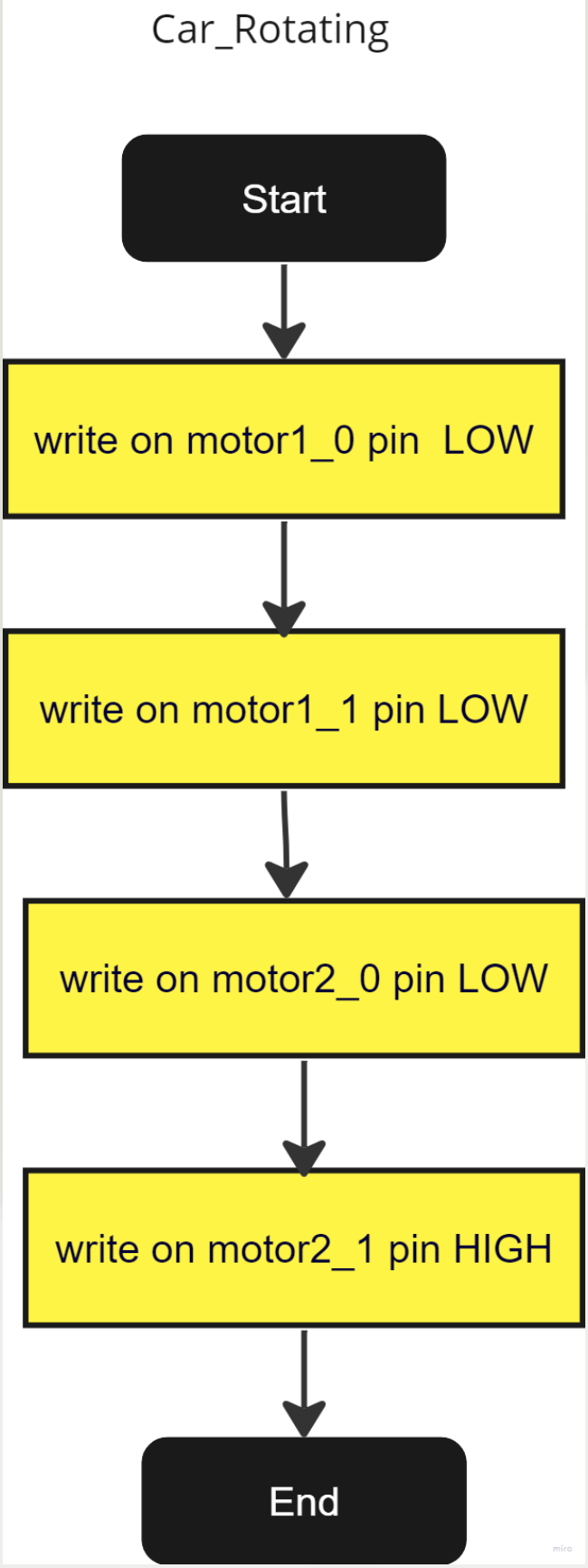
LOW LEVEL DESIGN

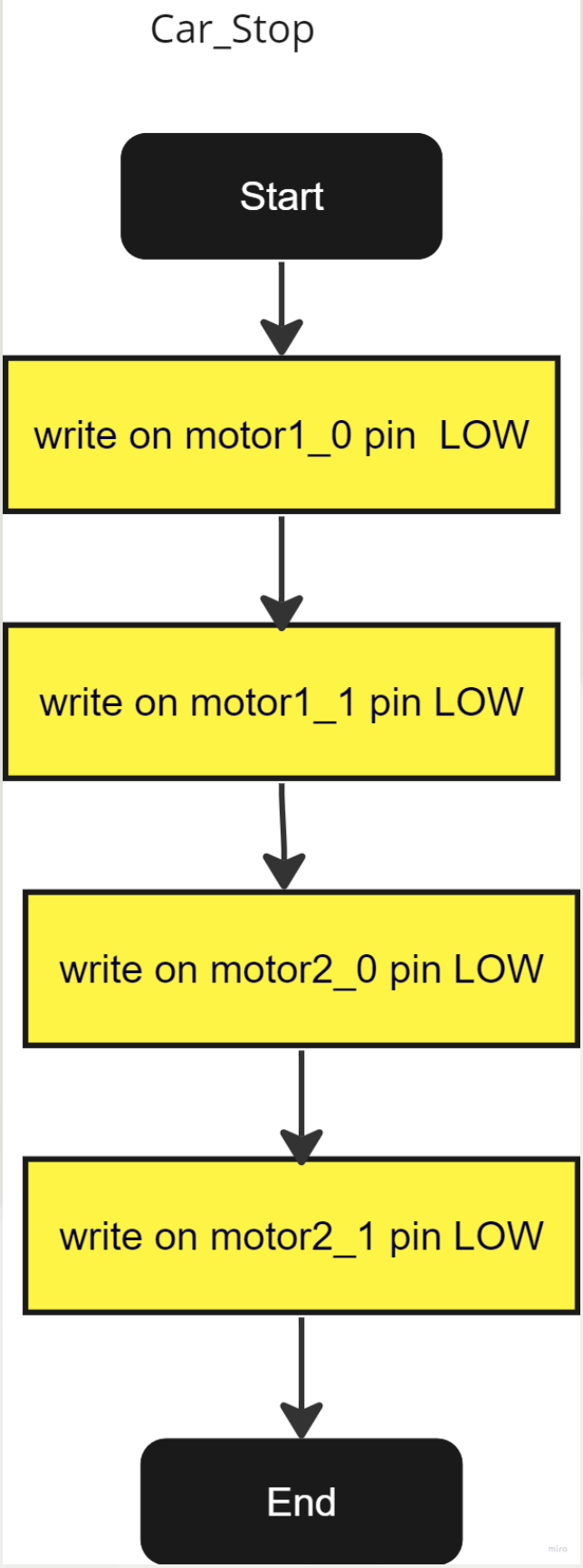
Flow charts for module's functions

1. motor module:



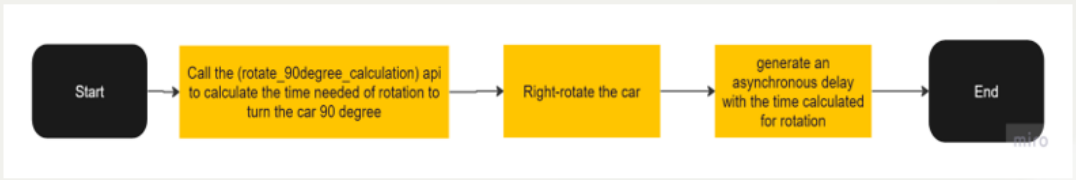

Car_Moving_FWD
Start
write on motor1_0 pin LOW
write on motor1_1 pin HIGH
write on motor2_0 pin LOW
write on motor2_1 pin HIGH
End


Car_Rotating
Start
write on motor1_0 pin LOW
write on motor1_1 pin LOW
write on motor2_0 pin LOW
write on motor2_1 pin HIGH
End

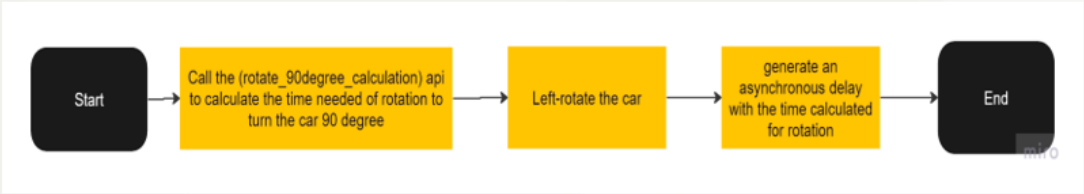

Car_Stop
Start
write on motor1_0 pin LOW
write on motor1_1 pin LOW
write on motor2_0 pin LOW
write on motor2_1 pin LOW
End

2. Car control module:

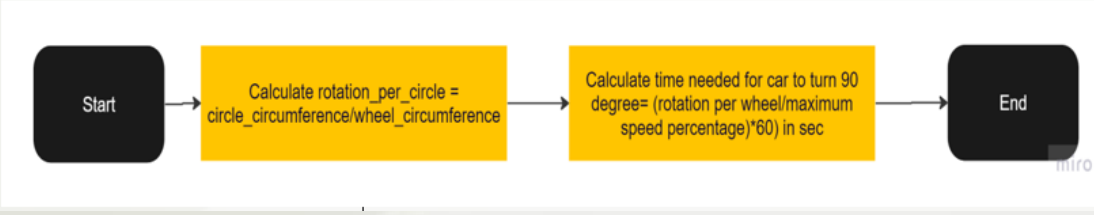
```
void rotate_90degree_Right (void);
```




```
void rotate_90degree_Left (void);
```



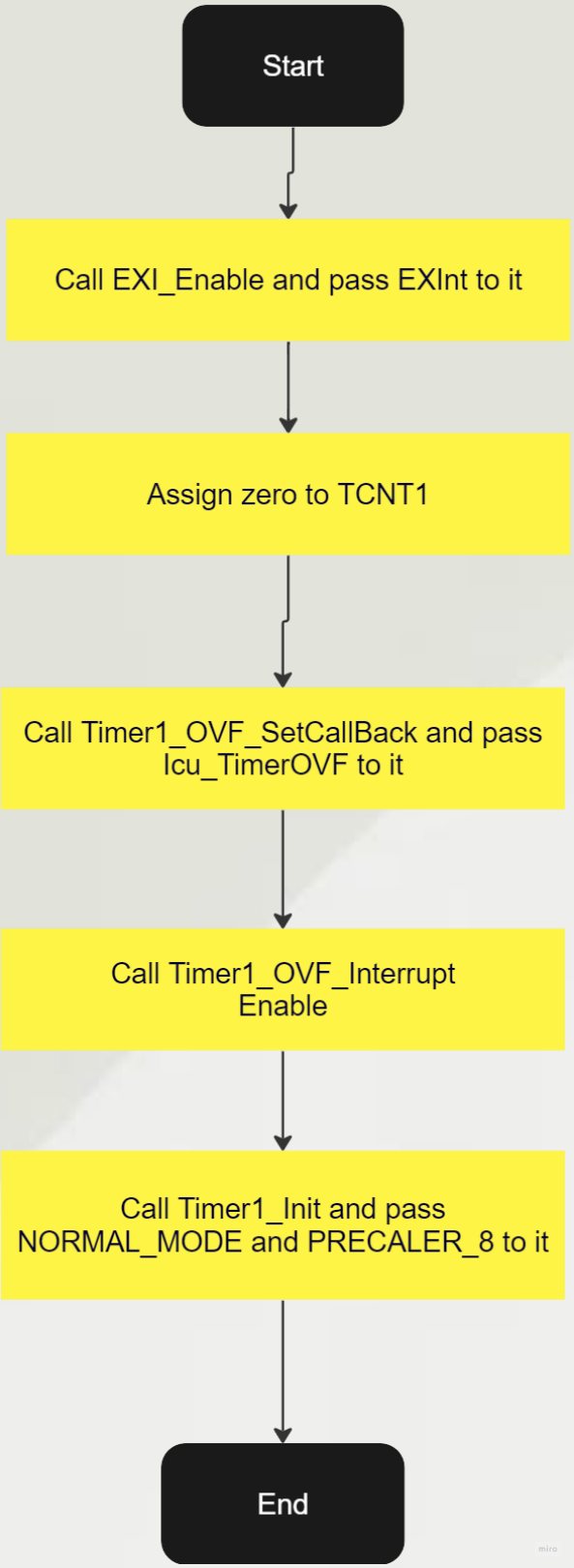

```
static void rotate_90degree_calculation (void);
```



3. ICU module:

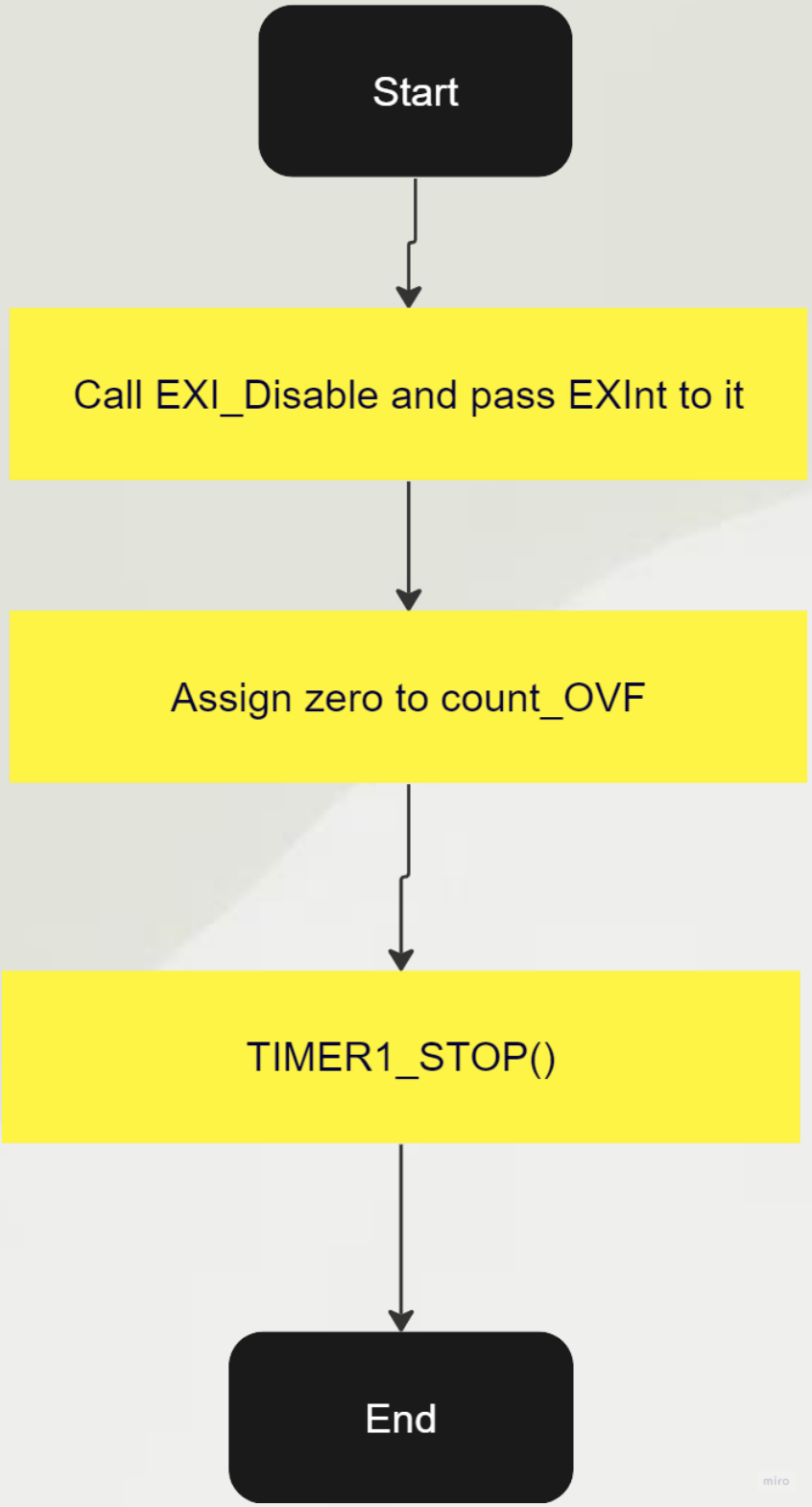
Sprints

Icu_Enable(EXTnT)

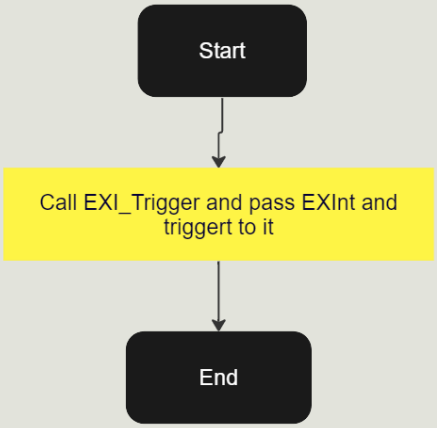


Sprints

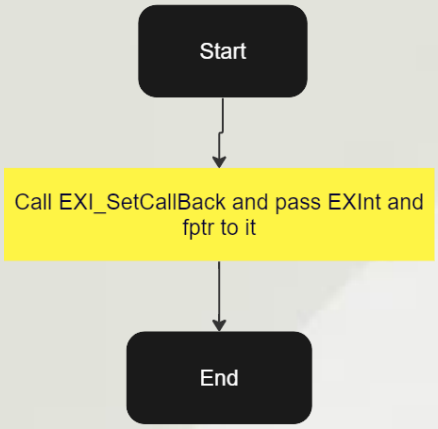
Icu_Disable(EXInt)



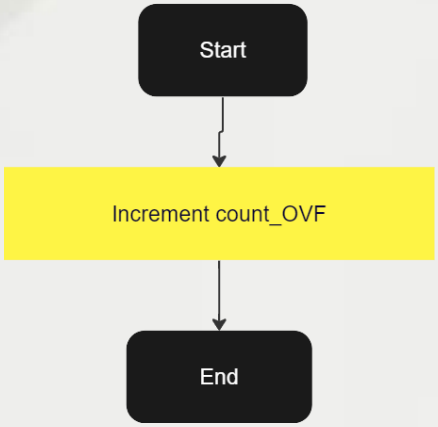
Icu_Trigger(EXInt,trig trigger)



Icu_SetCallback(EXInt,(*fptr)(void)

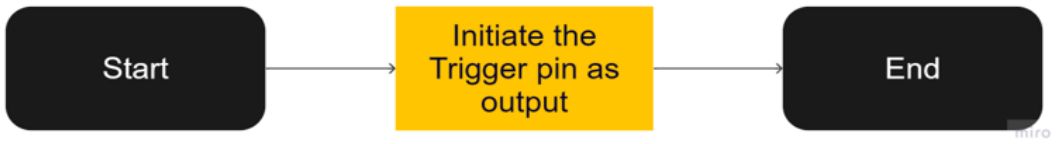


Icu_TimerOVF

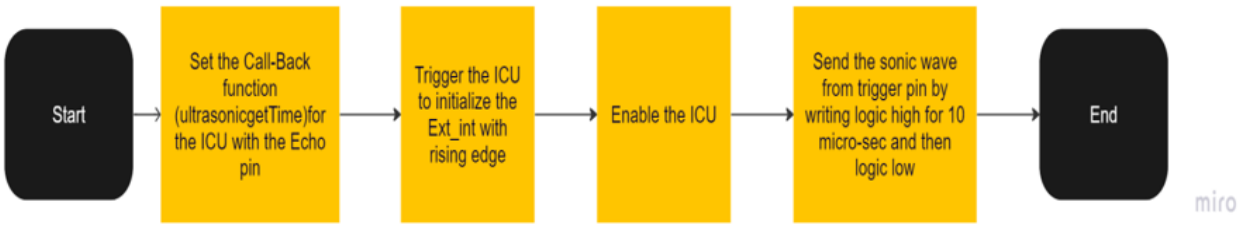


4. Ultrasonic sensor module:

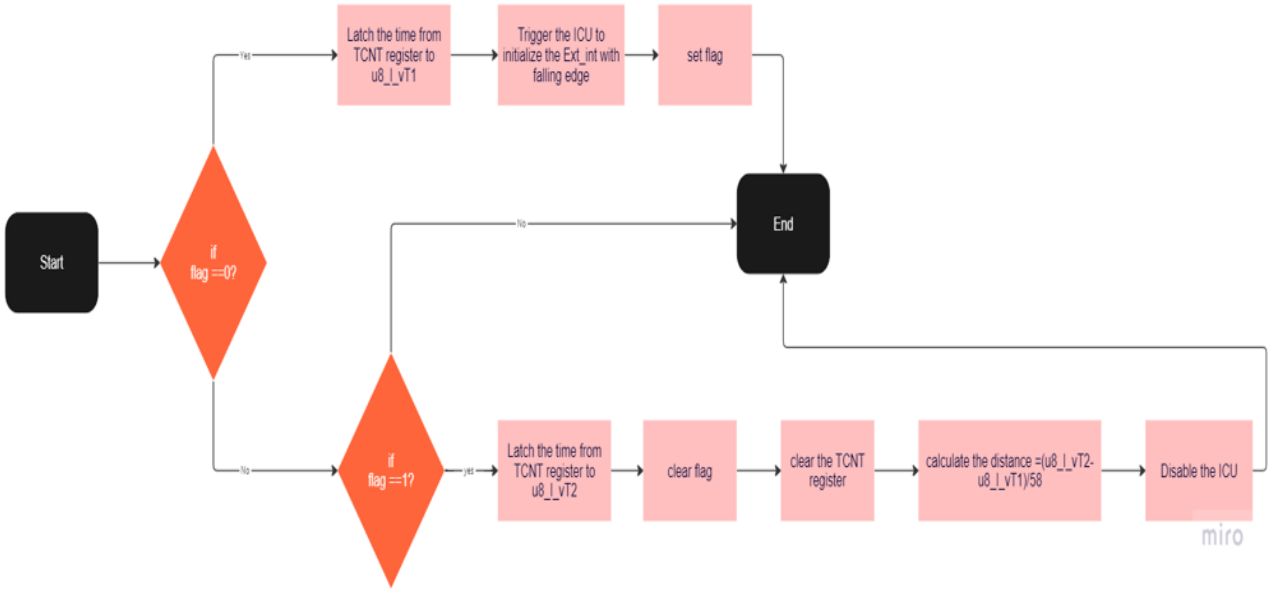
```
void Ultrasonic_init_SW(void);
```



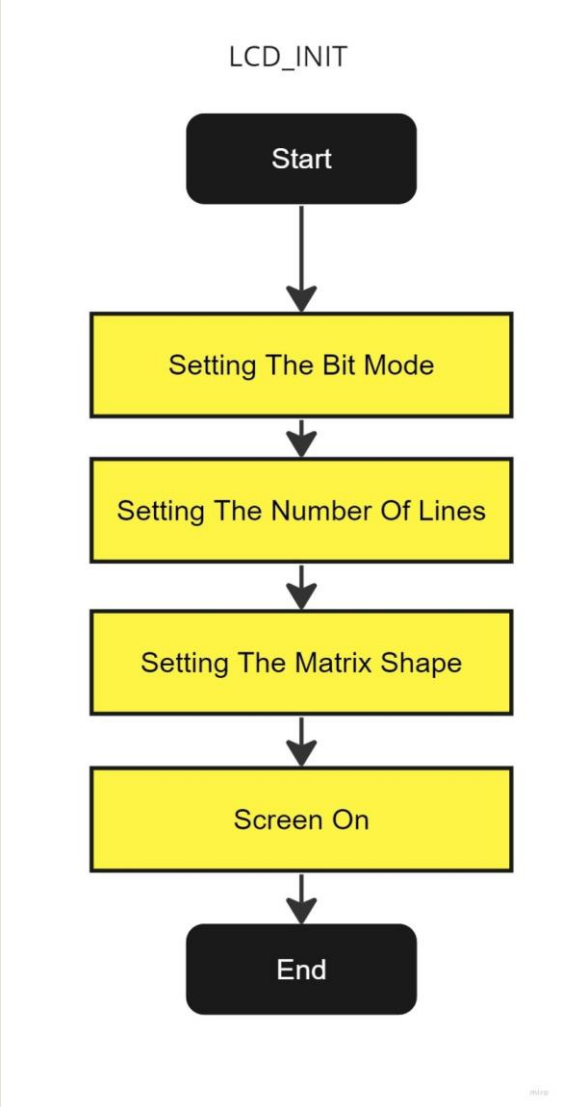

```
void Ultrasonic_GetDistance(uint32_t*dis);
```

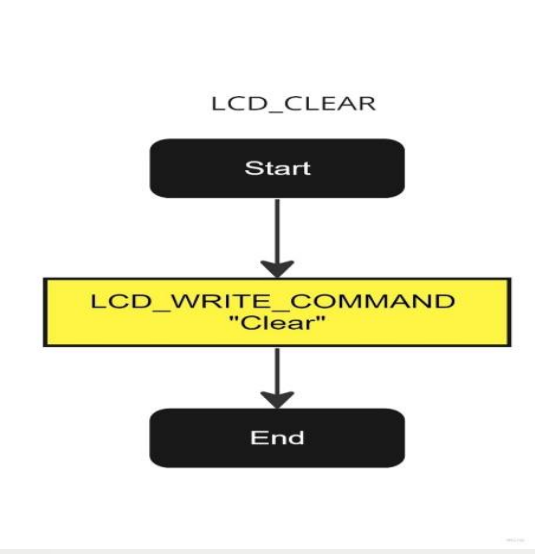


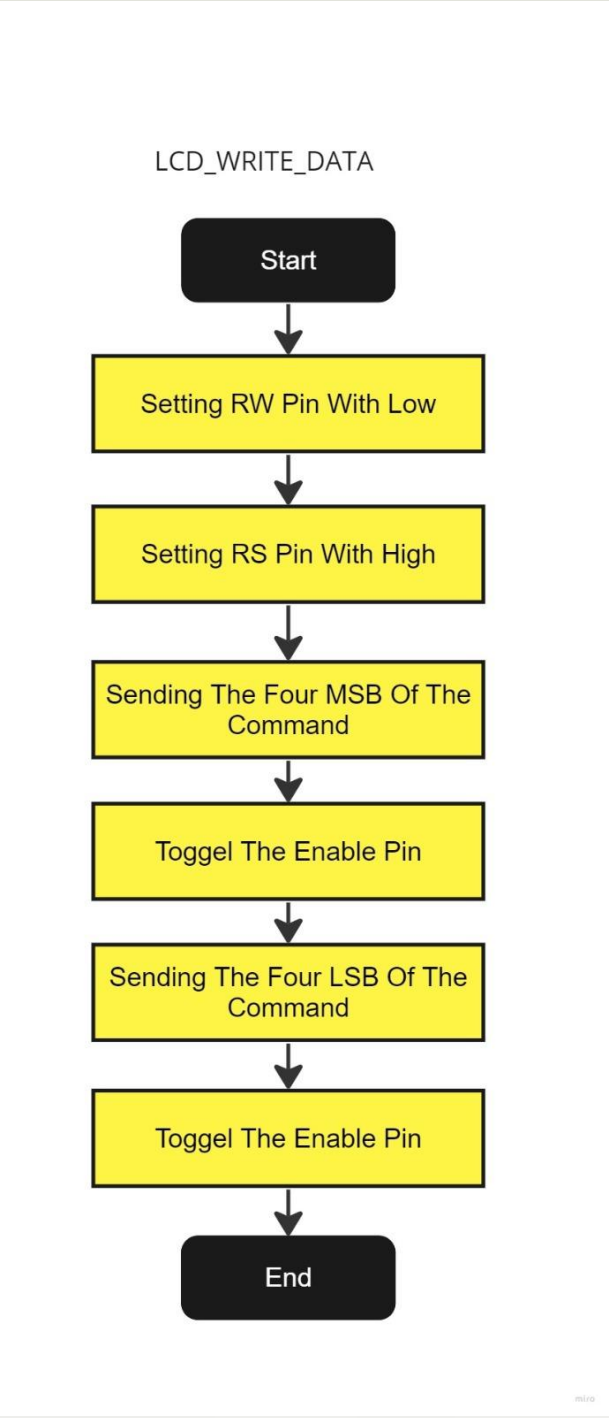

```
static void ultrasonicgetTime (void);
```

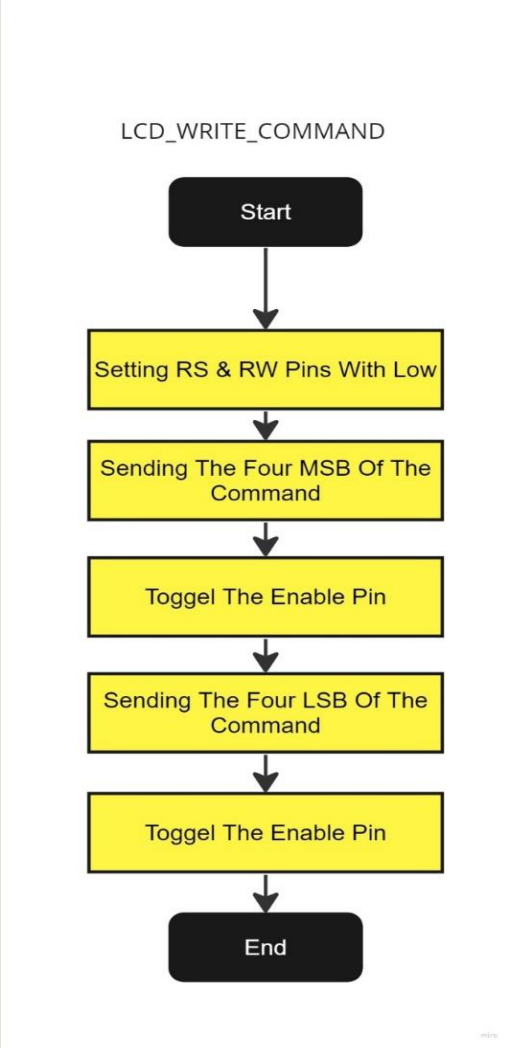


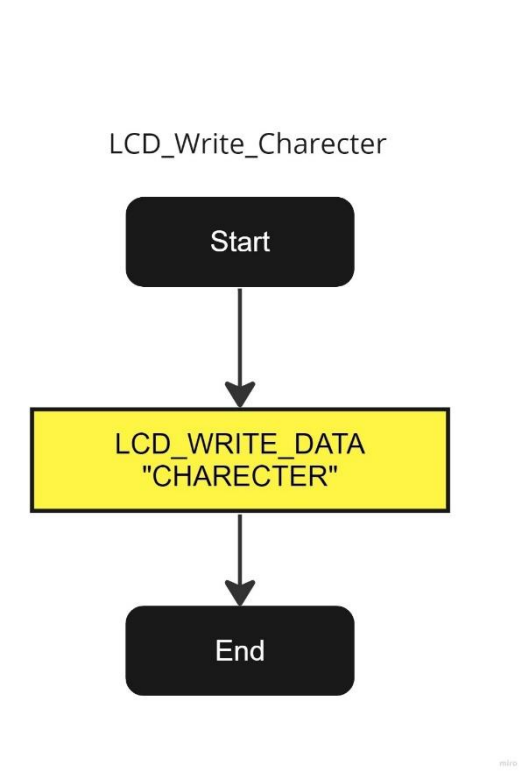
5. LCD module:

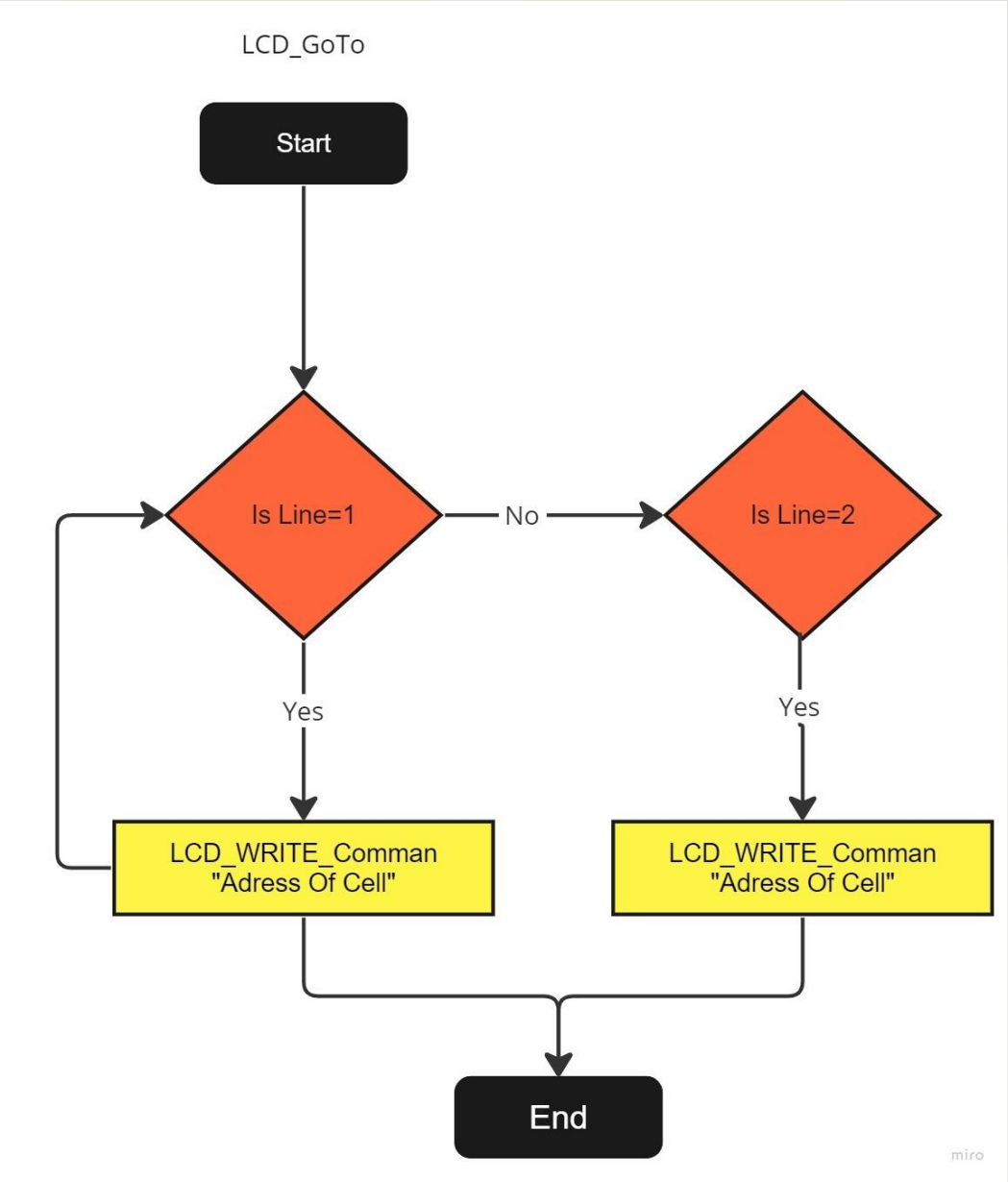

LCD_INIT
Start
Setting The Bit Mode
Setting The Number Of Lines
Setting The Matrix Shape
Screen On
End

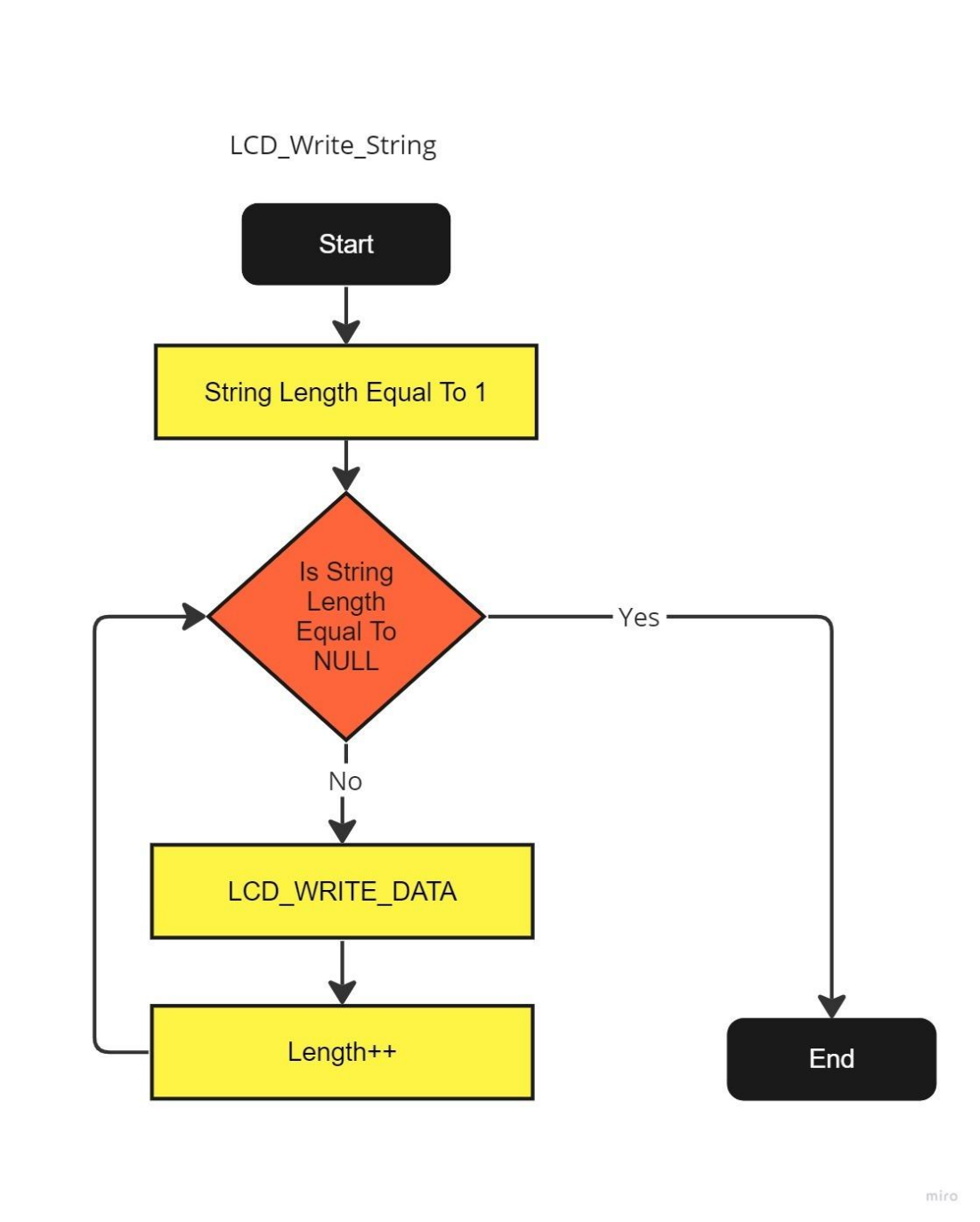

LCD_CLEAR
Start
LCD_WRITE_COMMAND
"Clear"
End

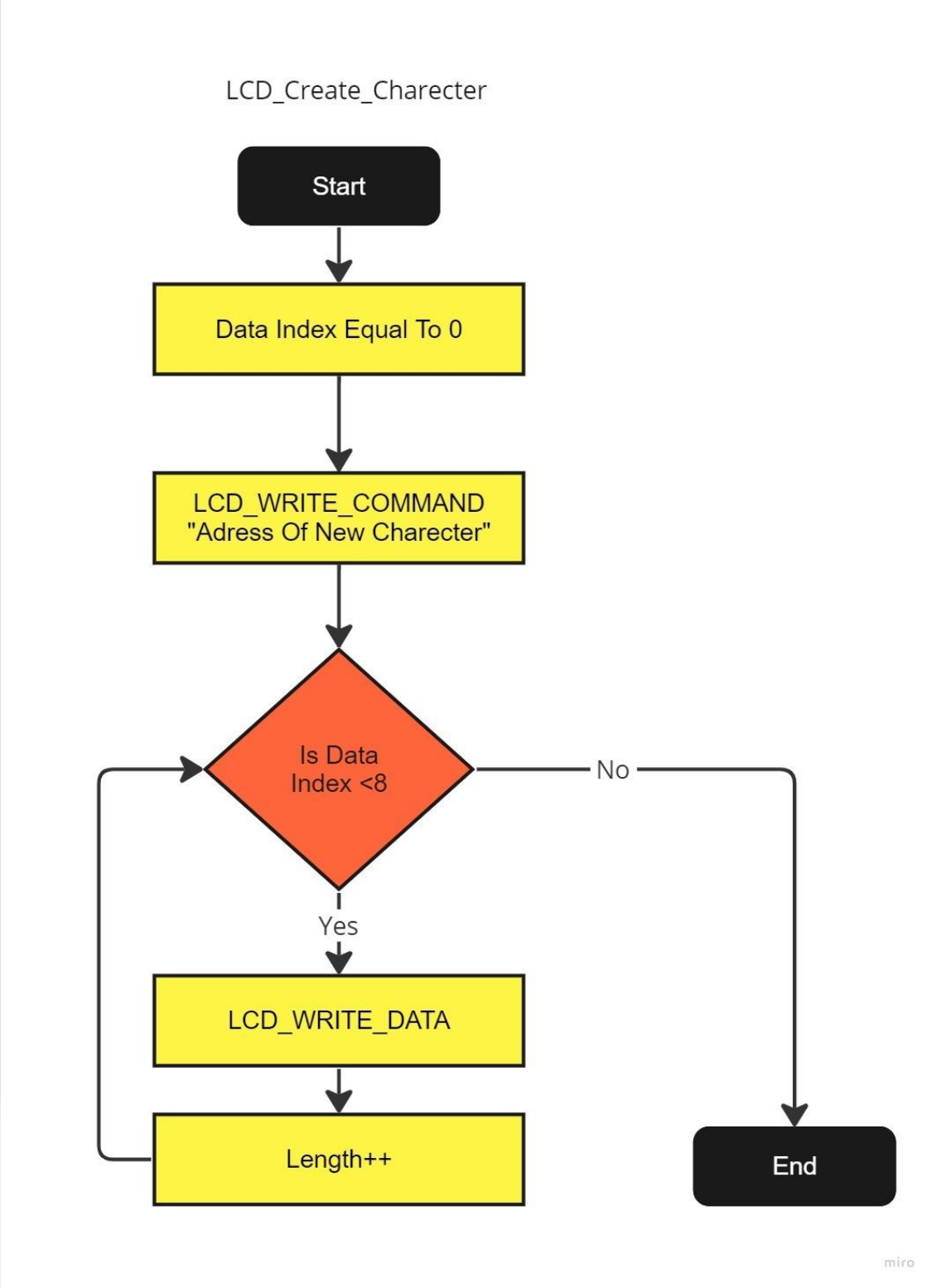

LCD_WRITE_DATA
Start
Setting RW Pin With Low
Setting RS Pin With High
Sending The Four MSB Of The Command
Toggel The Enable Pin
Sending The Four LSB Of The Command
Toggel The Enable Pin
End


LCD_WRITE_COMMAND
Start
Setting RS & RW Pins With Low
Sending The Four MSB Of The Command
Toggel The Enable Pin
Sending The Four LSB Of The Command
Toggel The Enable Pin
End

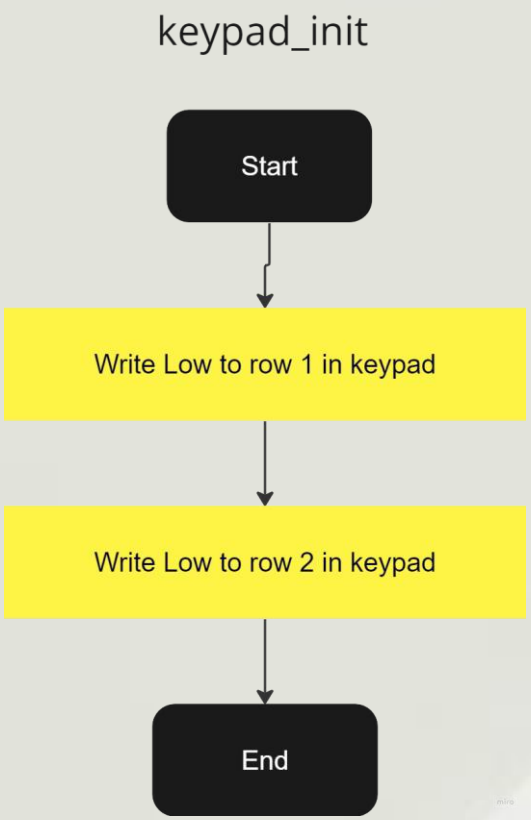

LCD_Write_Charecter
Start
LCD_WRITE_DATA
"CHARECTER"
End

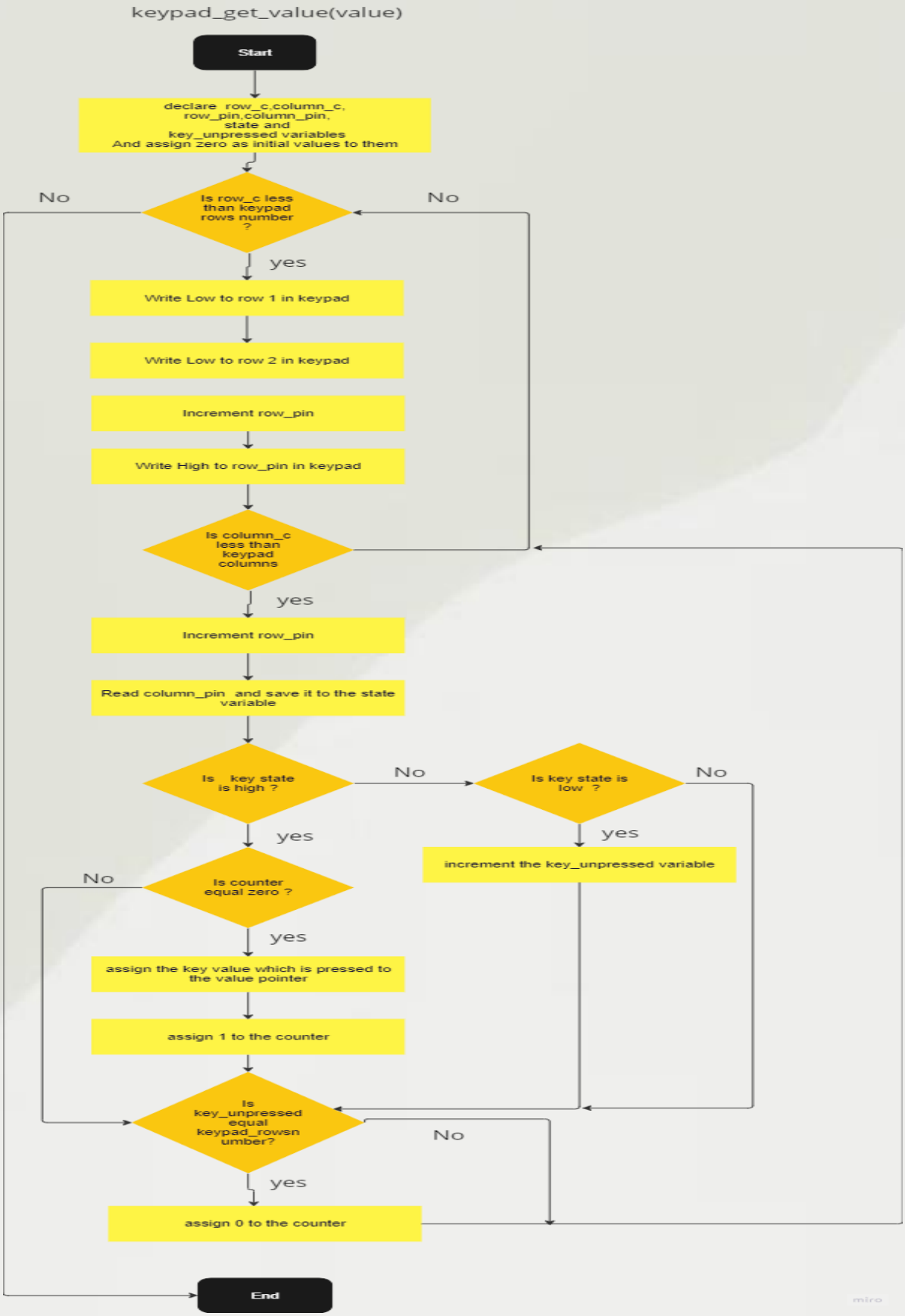

LCD_GoTo
Start
Is Line=1
No
Is Line=2
Yes
Yes
LCD_WRITE_Comman
"Adress Of Cell"
LCD_WRITE_Comman
"Adress Of Cell"
End
miro


LCD_Write_String
Start
String Length Equal To 1
Is String Length Equal To NULL
Yes
No
LCD_WRITE_DATA
Length++
End
miro

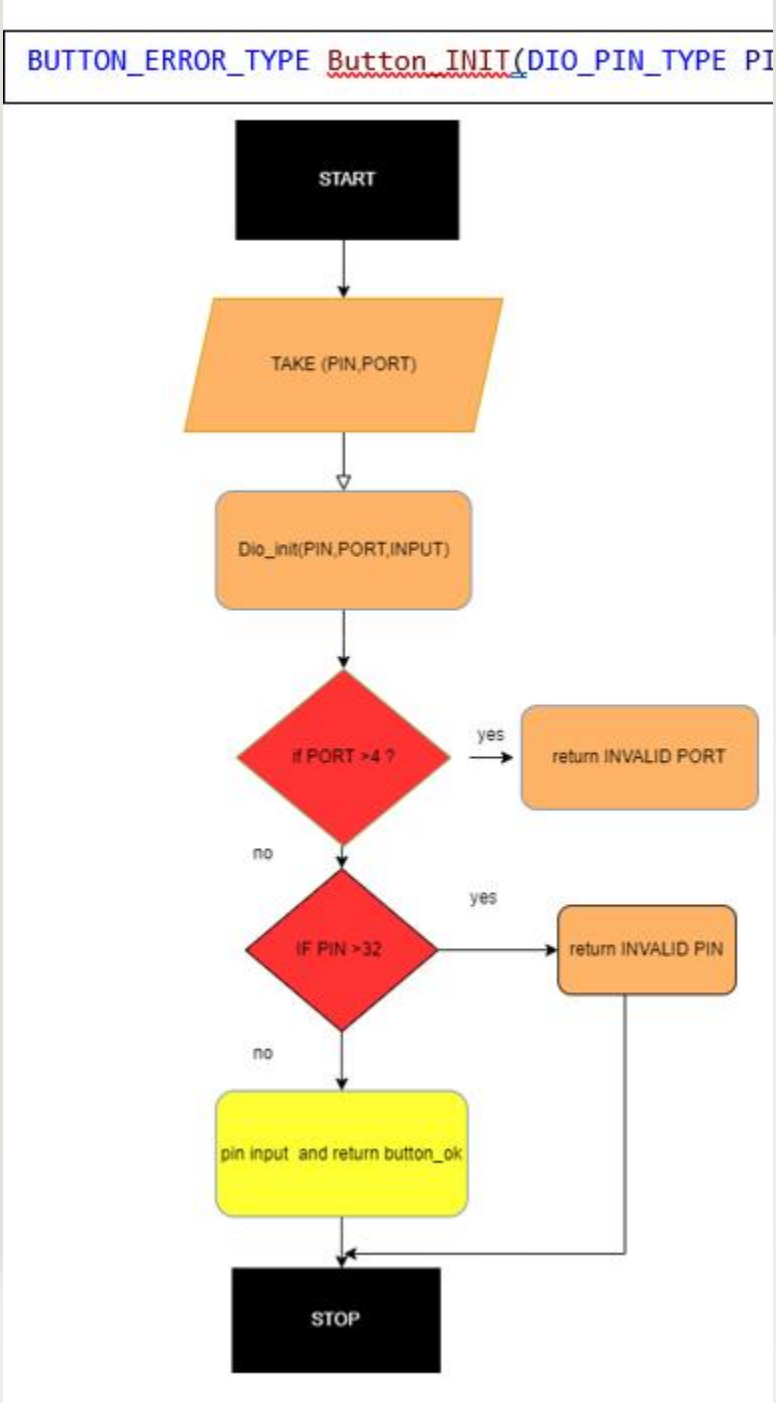

LCD_Create_Charecter
Start
Data Index Equal To 0
LCD_WRITE_COMMAND
"Adress Of New Charecter"
Is Data
Index <8
No
Yes
LCD_WRITE_DATA
Length++
End
miro

6. Keypad module:

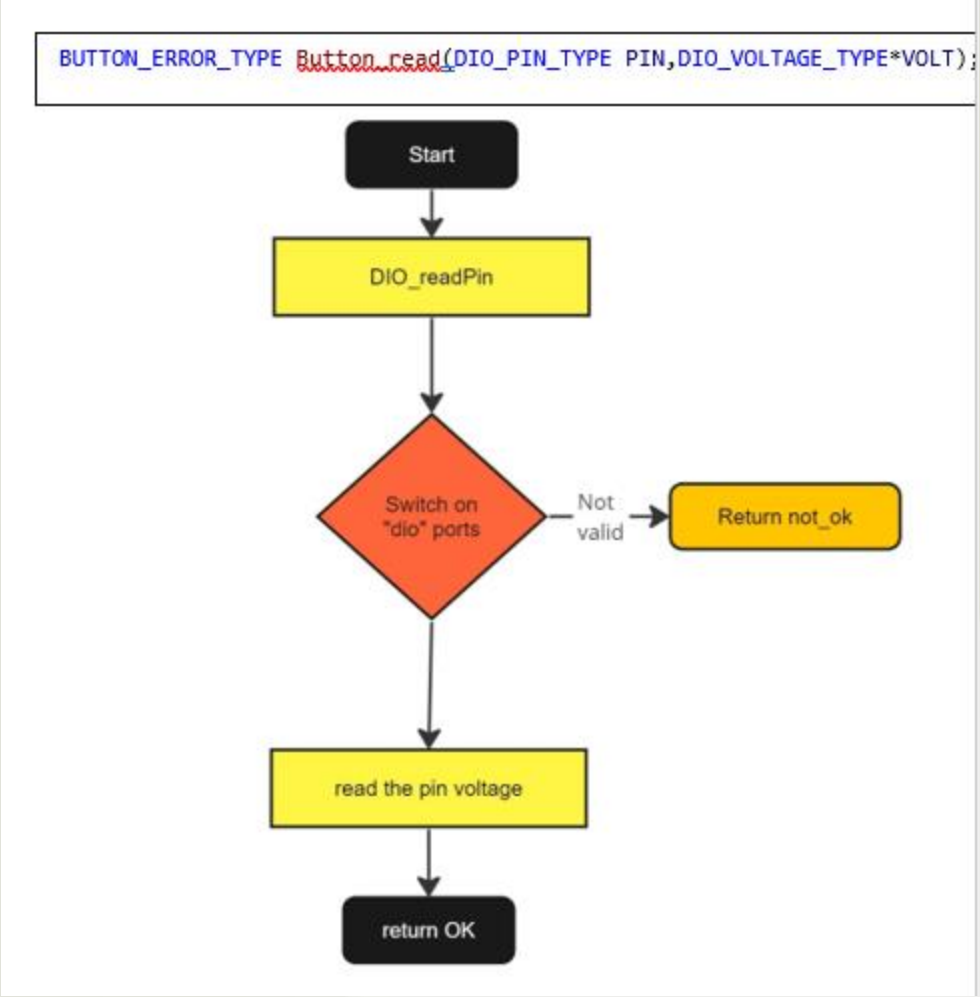



keypad_get_value(value)
Start
declare row_c,column_c, row_pin,column_pin, state and key_unpressed variables And assign zero as initial values to them
Is row_c less than keypad rows number ?
No
No
yes
Write Low to row 1 in keypad
Write Low to row 2 in keypad
Increment row_pin
Write High to row_pin in keypad
Is column_c less than keypad columns
yes
Increment row_pin
Read column_pin and save it to the state variable
Is key state is high ?
No
Is key state is low ?
No
yes
yes
increment the key_unpressed variable
Is counter equal zero ?
No
yes
assign the key value which is pressed to the value pointer
assign 1 to the counter
Is key_unpressed equal keypad_rowsn umber?
No
yes
assign 0 to the counter
End

7. Button:

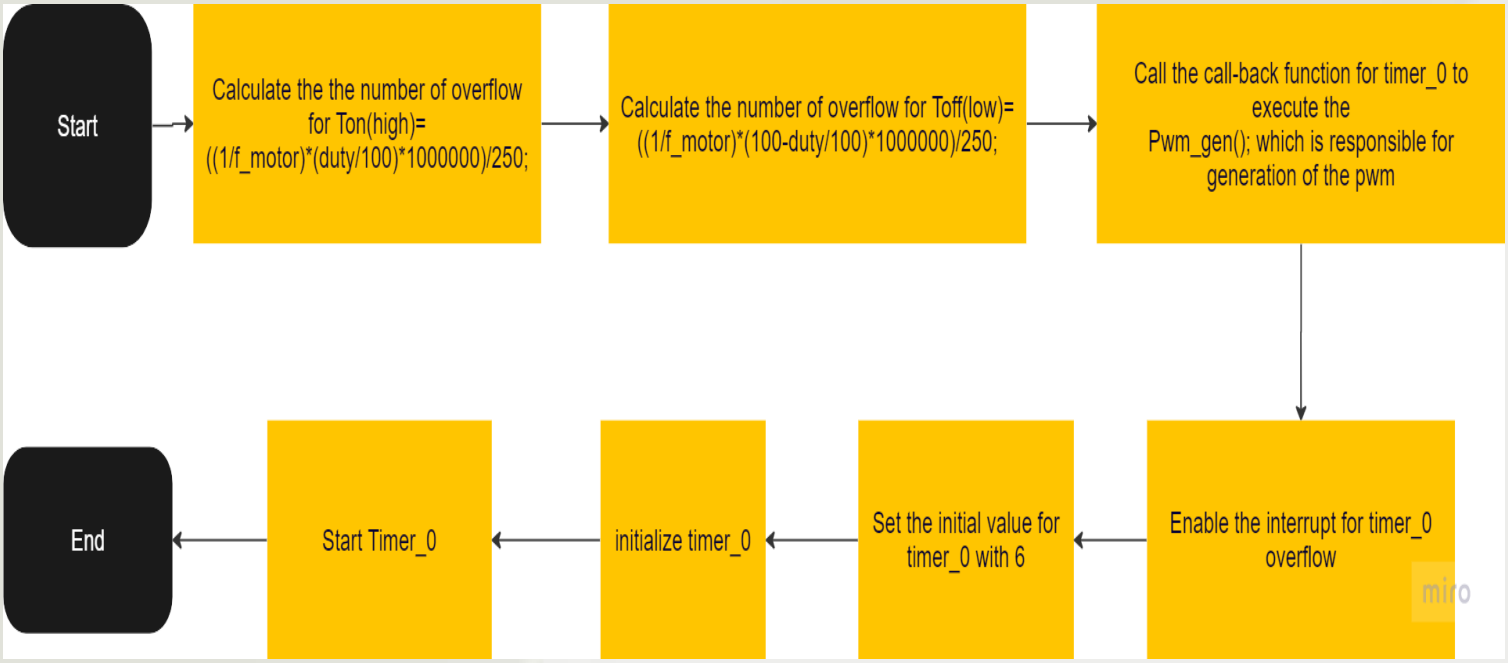

BUTTON_ERROR_TYPE Button_INIT(DIO_PIN_TYPE PI
START
TAKE (PIN,PORT)
Dio_init(PIN,PORT,INPUT)
if PORT >4 ?
yes
return INVALID PORT
no
IF PIN >32
yes
return INVALID PIN
no
pin input and return button_ok
STOP

```
BUTTON_ERROR_TYPE Button_read(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE*VOLT);
```

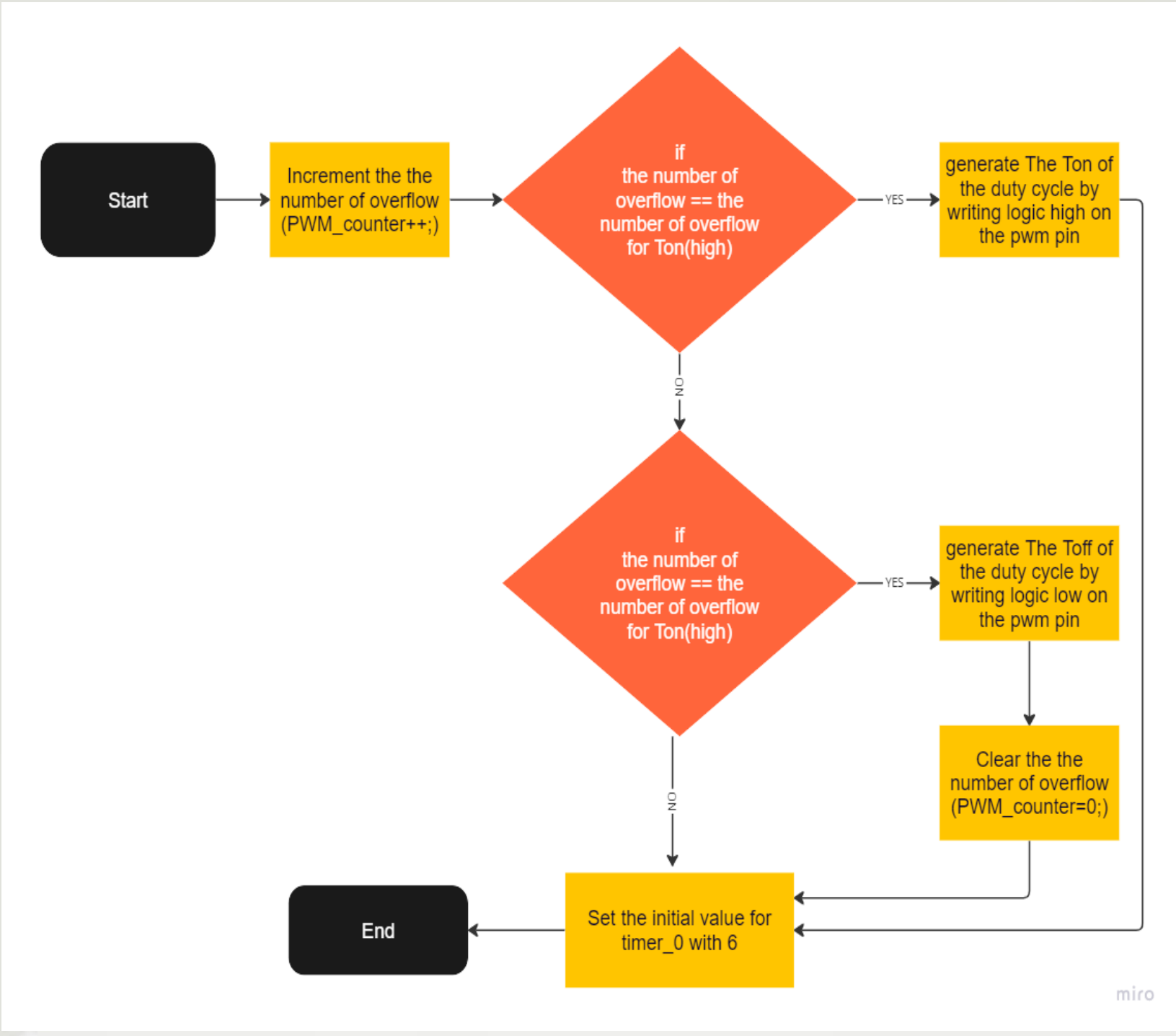


8. PWM:

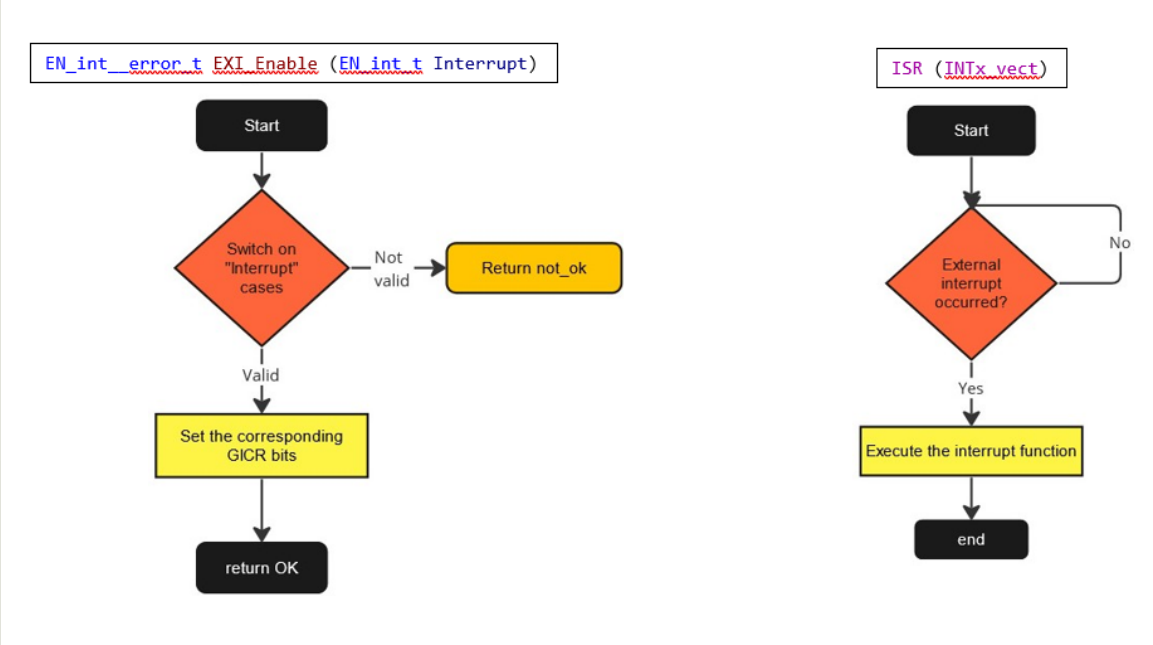
```
void PWM_duty(uint8_t duty);
```

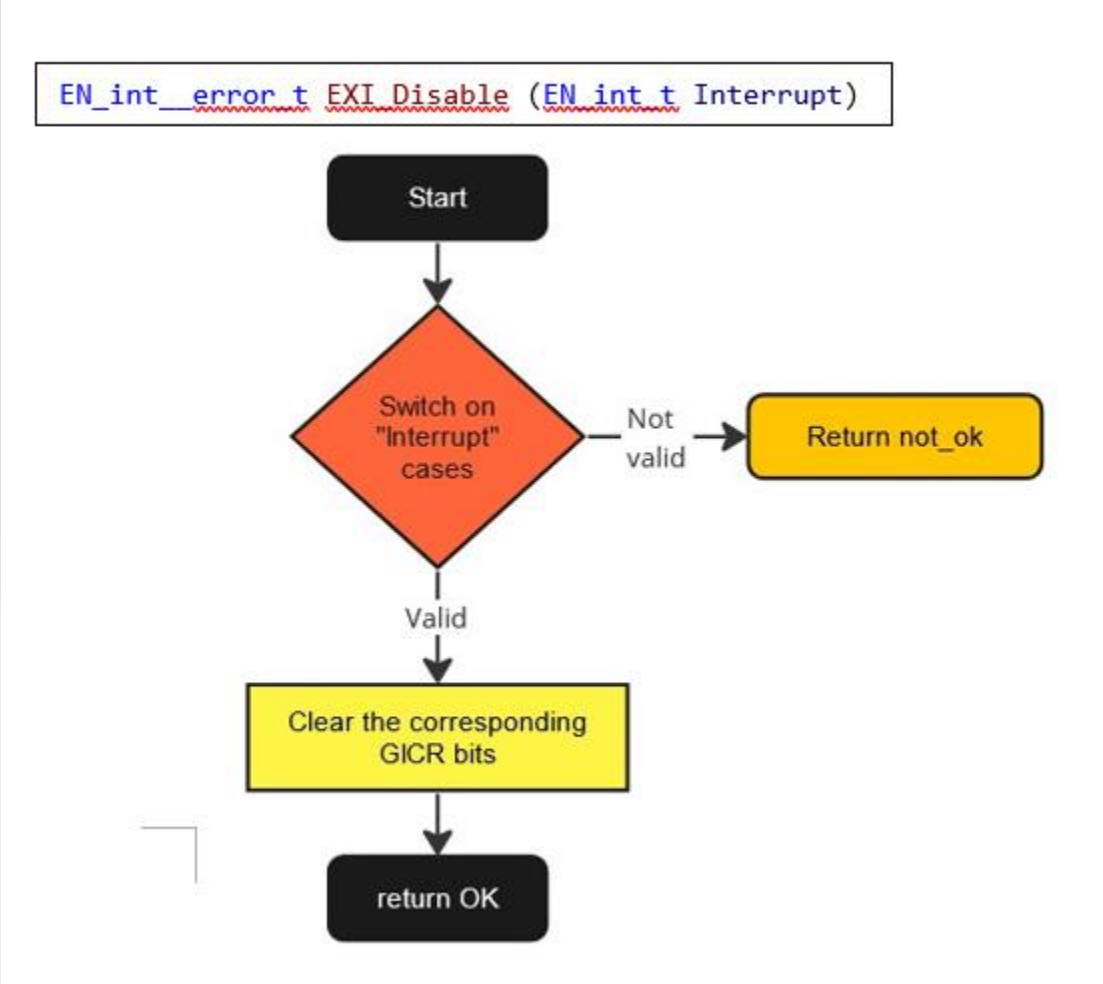


```
static void PWM_gen();
```

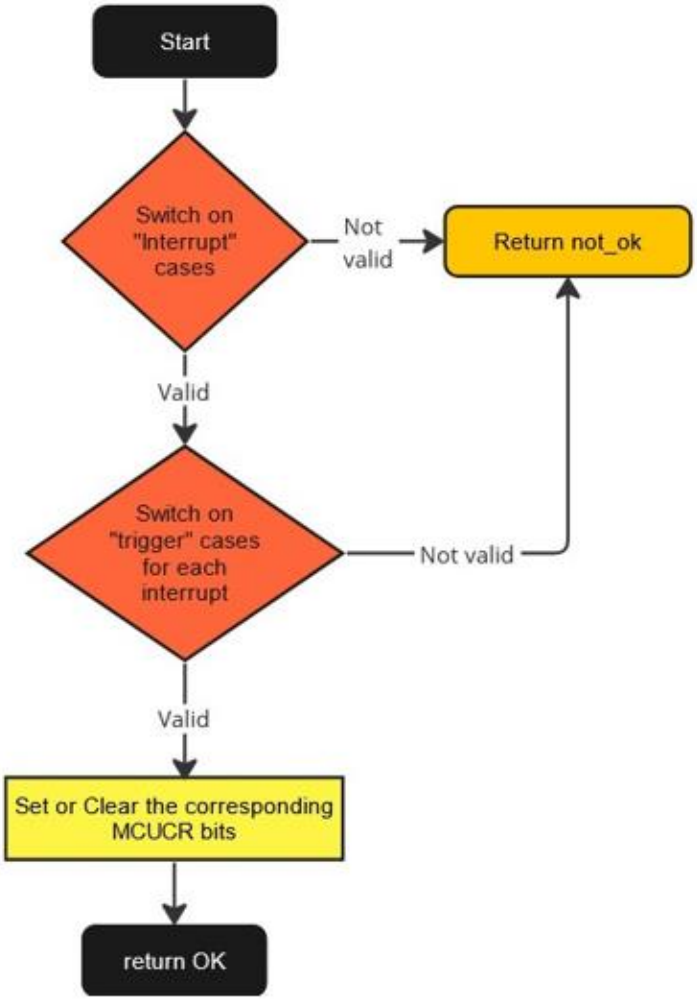


9. External Interrupt module:

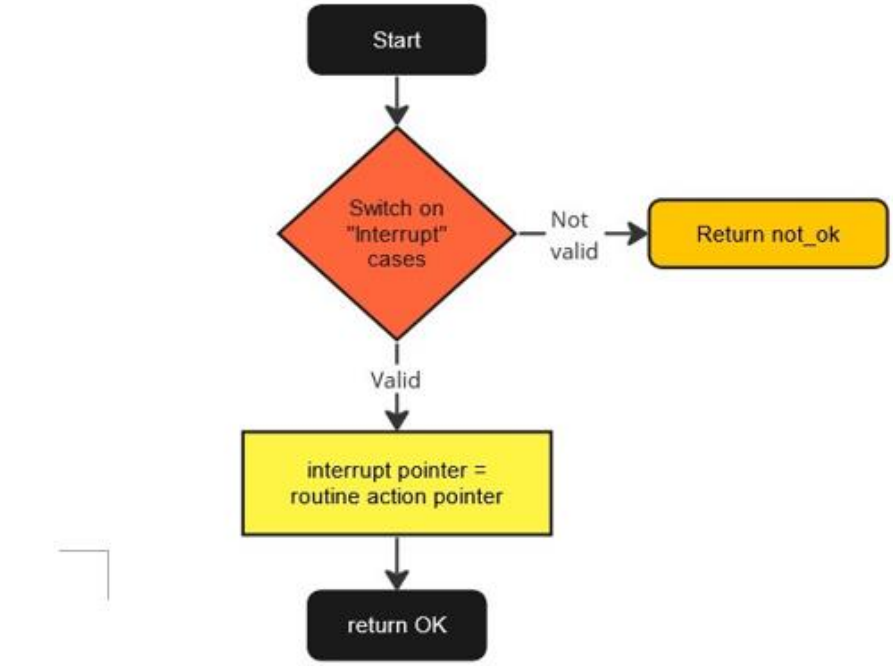




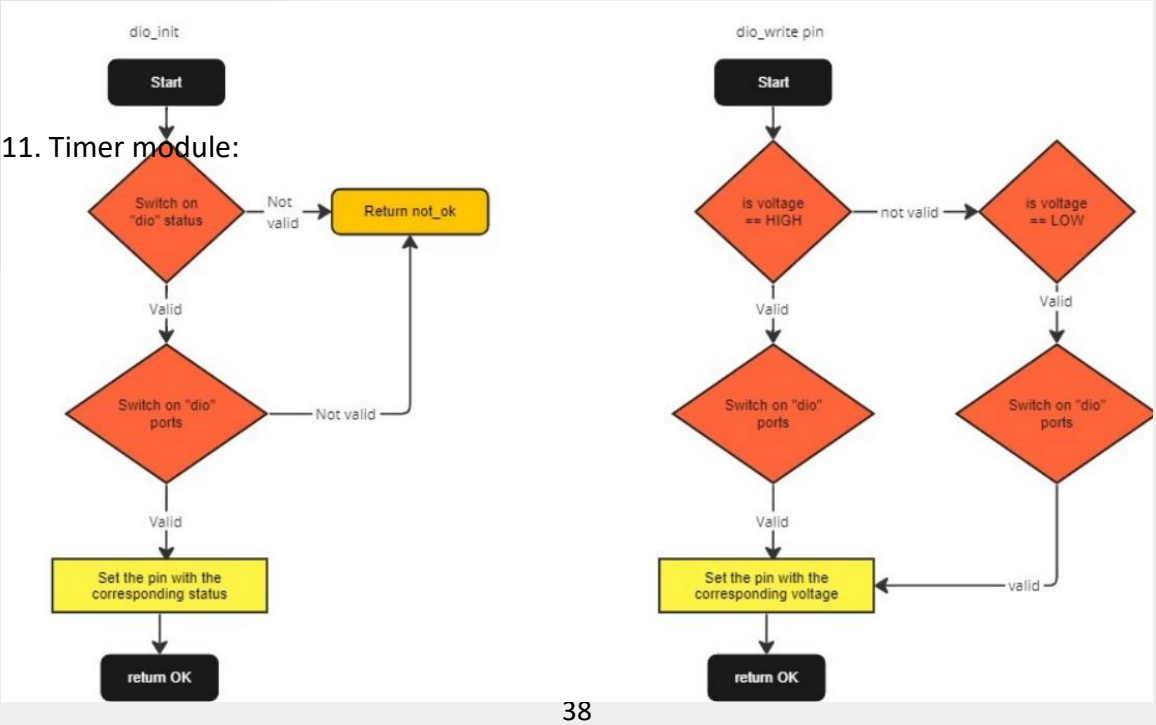
```
EN_int__error_t EXI_Trigger(EN_int_t Interrupt,EN_trig trigger)
```



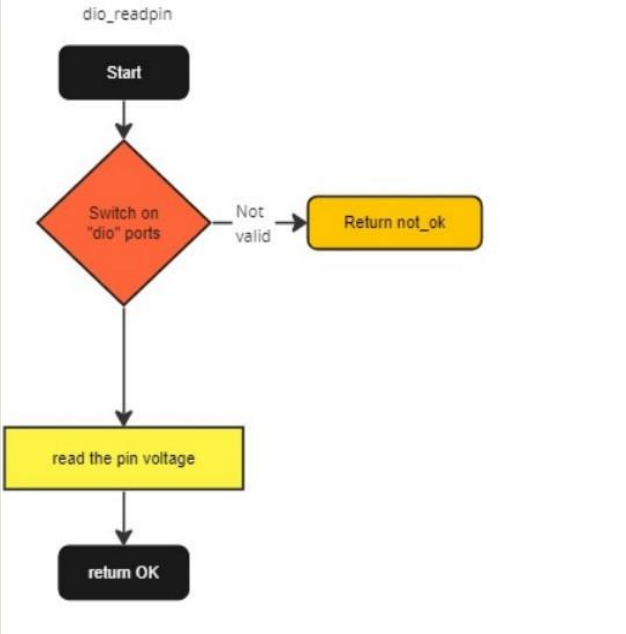
```
EN_int__error_t EXI_SetCallBack(EN_int_t Interrupt,void(*ptrf)(void))
```



10. DIO module:

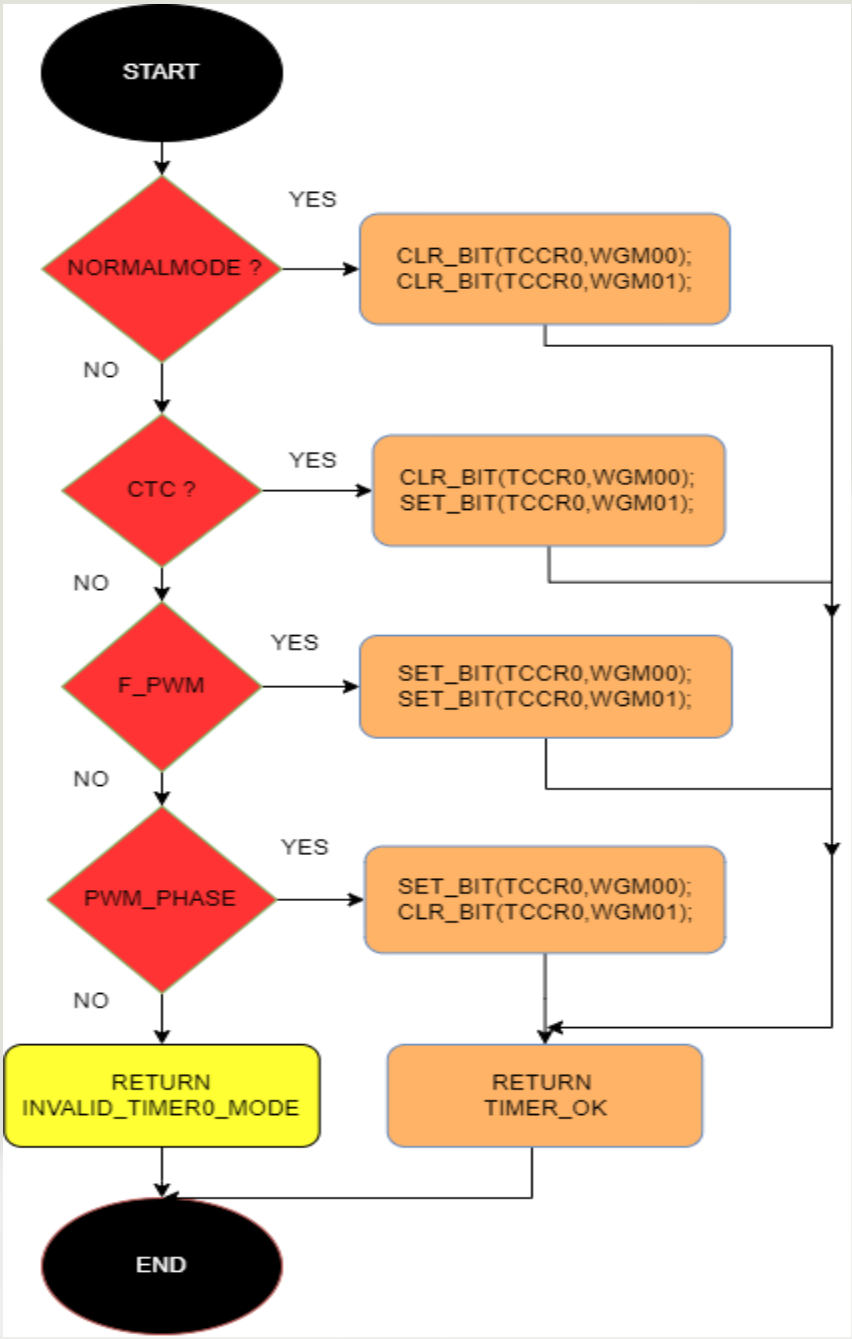


11. Timer module:

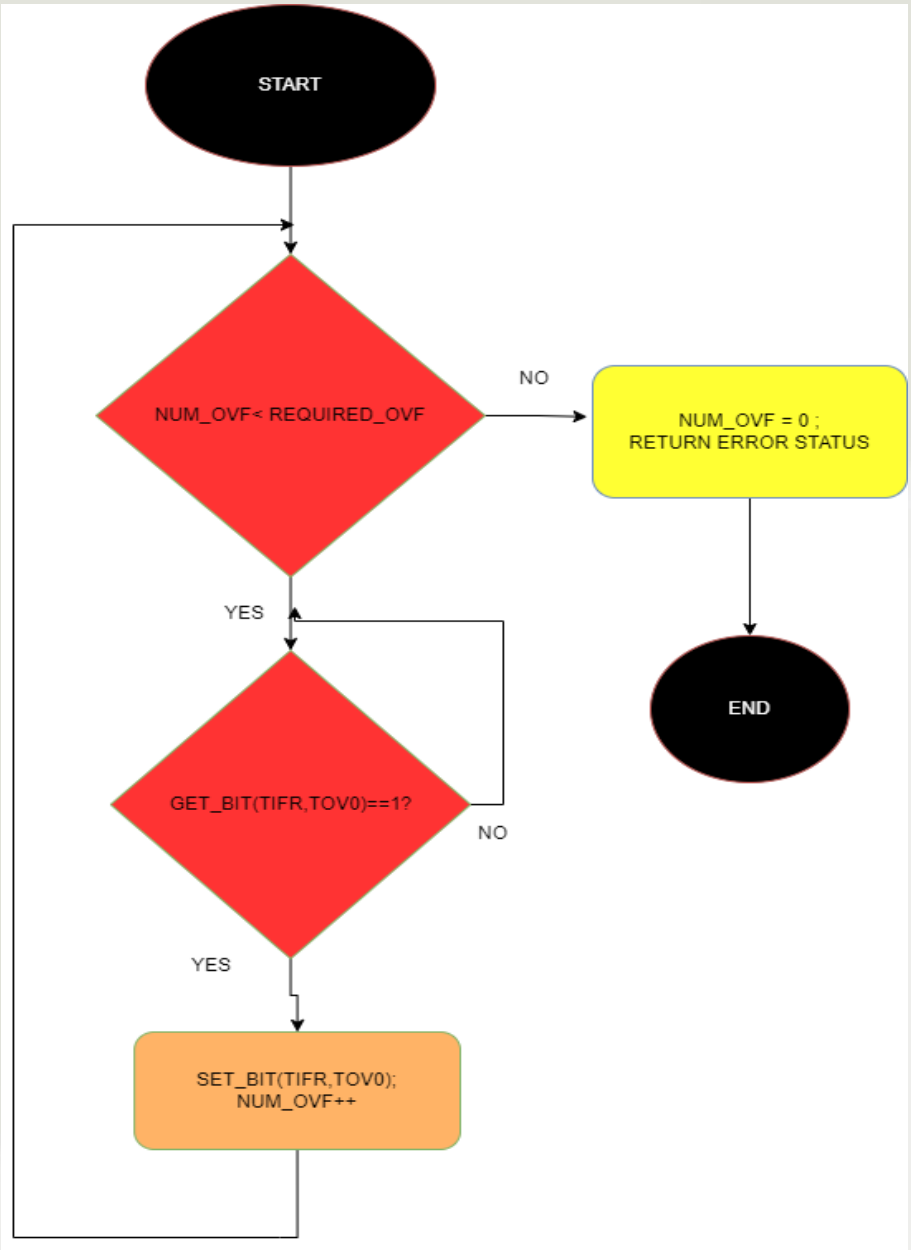

dio_readpin
Start
Switch on "dio" ports
Not valid
Return not_ok
read the pin voltage
return OK

11. Timer module:

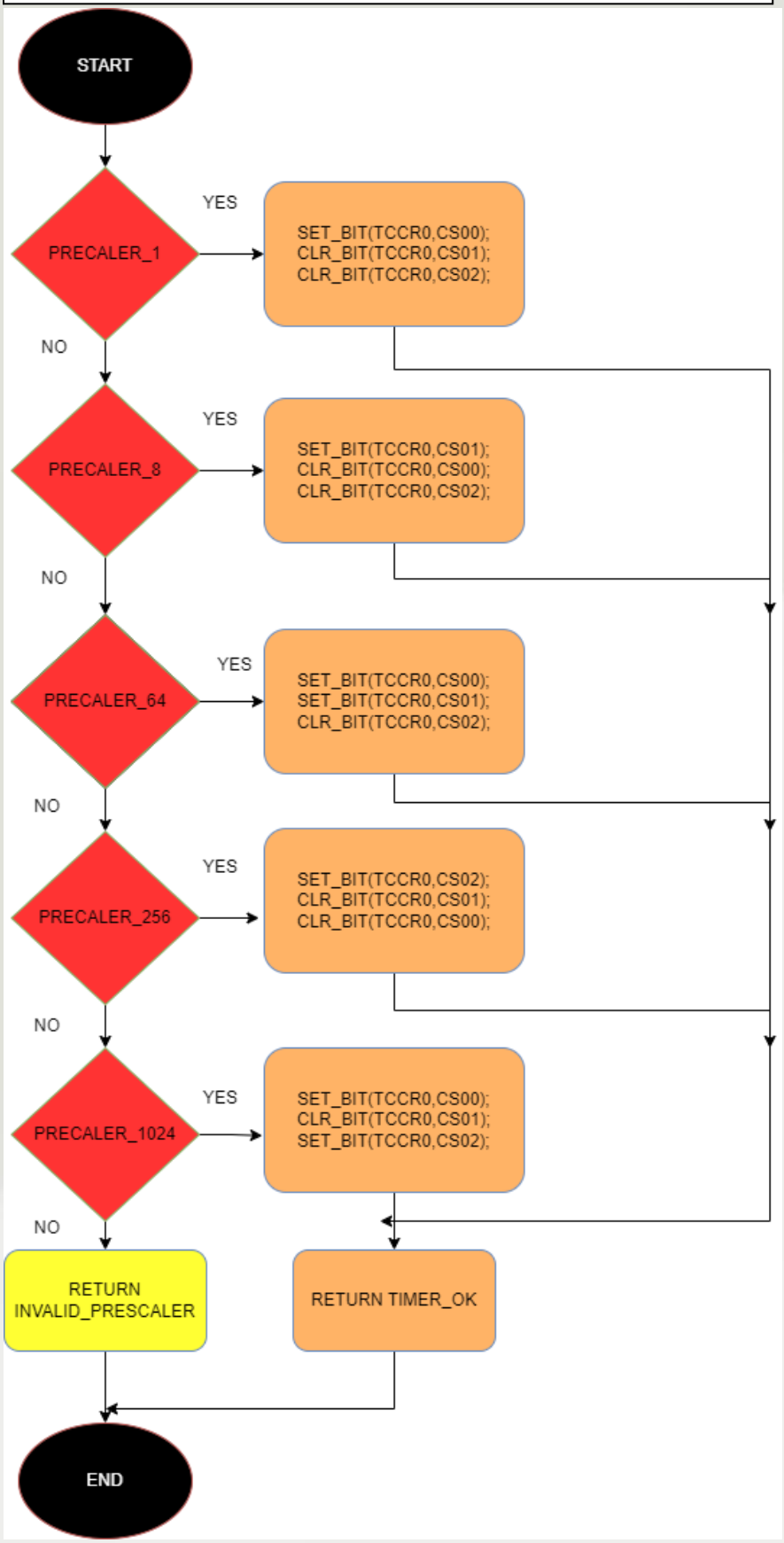
```
Timer_ErrorStatus TIMER_0_init(Timer_Mode mode);
```



```
Timer_ErrorStatus TIMER_0_OvfNum(double overflow);
```



```
Timer_ErrorStatus TIMER_0_start(Timer_Prescaler prescaler);
```



TIMER 2 WITH INTERRUPT