DESIGN DOCUMENT

OBSTACLE AVOIDANCE ROBOT V1.0 DESIGN

Prepared by:

Amr Gamal El-Abd

Table of Contents: -

Subject **Page**

Project introduction:

The project aims to design a four-wheel robot equipped with an object detection system. The robot is programmed to avoid any objects in front of it by adjusting its speed and direction accordingly. The main components of the system include an ATmega32 microcontroller, four motors, a button for changing the default rotation direction, a keypad for control inputs, an ultrasonic sensor for object detection, and an LCD for displaying relevant information.

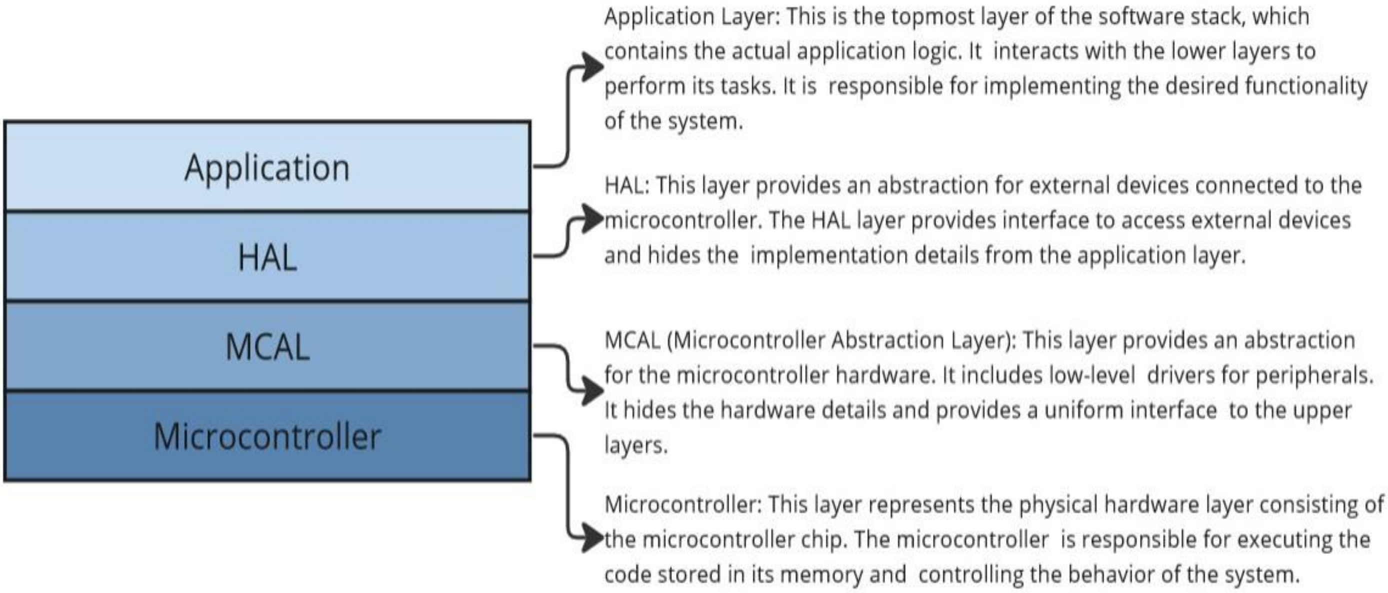
Project Main Flow:

1. Initialize the robot system and components.
2. Wait for user input: Press Keypad button 1 to start the robot.
3. Display "Set Def. Rot." on the LCD line 1 and "Right" on line 2.
4. Allow 5 seconds for the user to press PBUTTON0 to toggle the default rotation direction between Left and Right. Update LCD accordingly.
5. Set the default rotation direction after the 5-second interval.
6. Wait for 2 seconds before the robot starts moving.
7. Start robot movement with 30% speed, moving forward.
8. Display speed, direction, and object distance on the LCD.
9. Check if obstacles are located at a distance between 30 and 70 cm.
10. If obstacles are detected, reduce the robot's speed to 30% and update the LCD.
11. Check if obstacles are located between 20 and 30 cm.
12. If obstacles are detected, stop the robot, and rotate it 90 degrees to the right or left based on the chosen configuration. Update the LCD.
13. Check if obstacles are located less than 20 cm. away.
14. If obstacles are detected, stop the robot, move it backward with 30% speed until the distance is greater than 20 and less than 30 cm. Update the LCD.
15. Repeat steps 9-14 to continuously monitor and react to obstacles.
16. Implement the bonus feature: Check if the robot has rotated 360 degrees without detecting any object at a distance greater than 20 cm.
17. If no object is found, stop the robot and update the LCD.
18. Periodically check every 3 seconds if any obstacles have been removed and move in the direction of the furthest object. Update the LCD.
19. Continue the obstacle avoidance and movement until the robot is manually stopped (Press Keypad button 2).

Hardware components:

- ATmega32 microcontroller
- **Four** motors (**M1**, **M2**, **M3**, **M4**)
- One button to change default rotation direction (**PBUTTON0**)
- Keypad (KPD1: start, KPD2: stop)
- One ultrasonic sensor
- LCD

Layered Architectures:-



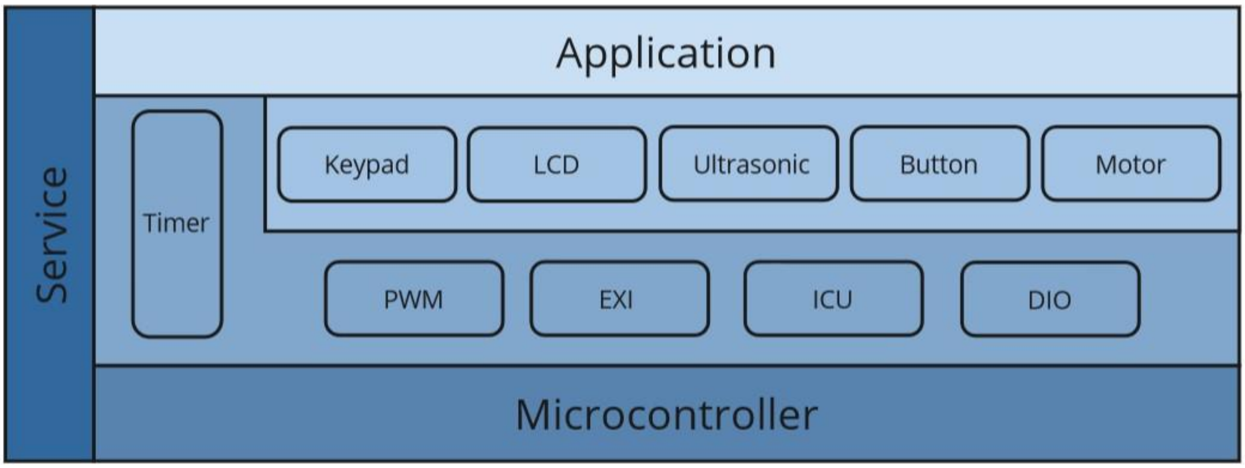
Application Layer: This is the topmost layer of the software stack, which contains the actual application logic. It interacts with the lower layers to perform its tasks. It is responsible for implementing the desired functionality of the system.

HAL: This layer provides an abstraction for external devices connected to the microcontroller. The HAL layer provides interface to access external devices and hides the implementation details from the application layer.

MCAL (Microcontroller Abstraction Layer): This layer provides an abstraction for the microcontroller hardware. It includes low-level drivers for peripherals. It hides the hardware details and provides a uniform interface to the upper layers.

Microcontroller: This layer represents the physical hardware layer consisting of the microcontroller chip. The microcontroller is responsible for executing the code stored in its memory and controlling the behavior of the system.

System modules:-



Module Descriptions:

1. **Microcontroller** (ATmega32): The ATmega32 microcontroller serves as the brain of the robot, responsible for processing inputs, controlling the motors, and interacting with other components. It executes the programmed logic to ensure proper movement and obstacle avoidance.

2. **Motors** (M1, M2, M3, M4): The four motors are responsible for driving the robot's wheels. By controlling the motors individually or in pairs, the robot can move forward, backward, rotate, or stop according to the desired actions and obstacle detection.

3. **Button** (PBUTTON0): This button allows the user to change the default rotation direction of the robot. Pressing it toggles between the left and right rotation configurations, which are displayed on the LCD.

4. **Keypad Buttons**: The keypad consists of two buttons for controlling the robot. Button 1 starts the robot's movement, while button 2 stops it. These buttons enable the user to initiate and halt the robot's operation as needed.

5. **Ultrasonic Sensor**: The ultrasonic sensor is used for object detection. It emits ultrasonic waves and measures the time it takes for the waves to bounce back. By calculating the distance to nearby objects based on the wave's travel time, the sensor provides essential data for the robot to navigate and avoid obstacles.

6. **ICU**: The ICU driver is responsible for interfacing the ATmega32 microcontroller with the ultrasonic sensor in the object detection robot project. The ultrasonic sensor measures the distance between the robot and objects in its path by emitting ultrasonic waves and measuring the time taken for the waves to bounce back.

7. **LCD**: The LCD display serves as an interface for conveying information to the user. It displays various details, such as the selected default rotation direction, current speed and direction of the robot, object distances, and system status.

8. **Timer**: The Timer module is responsible for setting up and controlling the timers of the microcontroller. This driver will be used to create the timing delays required in the project
9. **DIO**: The DIO (Digital Input Output) module is responsible for setting up the digital pins of the microcontroller to either input or output mode. This module will be used to control the external pins states.
10. **External Interrupt**: The Interrupt module is responsible for setting up and controlling the interrupts of the microcontroller. This module will be used to detect button presses.

Drivers' documentation:-

HAL drivers:

1. LCD Driver:

Description: This driver controls the LCD display and provides an interface between the microcontroller and the LCD hardware, allowing the microcontroller to display the temperature readings and messages to the user.

Functions:

```
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);
```

2. Keypad Driver:

Description: This driver provides an interface between the microcontroller and the keypad hardware, allowing the microcontroller to receive input from the user through the keypad buttons.

Functions:

```
void KEYPAD_init(void);
uint8_t KEYPAD_getKey(void);
```

3. Button Driver:

Description: The Button driver is responsible for setting up and controlling the buttons of the microcontroller. This driver will be used to detect button presses.

Functions:

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

4. Motor Driver:

Description: The Motor driver is responsible for setting up and controlling the motors of the car. This driver will be used to control the speed and direction of the motors.

Functions:

```
En_motorError_t Motors_init(void);
En_motorError_t Motors_Start(void);
En_motorError_t Motors_Rotating(void);
En_motorError_t Motors_Stop(void);
```

5. Ultrasonic:

Description: The ultrasonic sensor is used for object detection. It emits ultrasonic waves and measures the time it takes for the waves to bounce back. By calculating the distance to nearby objects based on the wave's travel time, the sensor provides essential data for the robot to navigate and avoid obstacles.

Functions:

```
void ULTRASONIC_init (u8 a_triggerPin, u8 a_echoPin);
float ULTRASONIC_read (void);
```

MCAL drivers:

1. DIO Driver:

Description: The DIO (Digital Input Output) driver is responsible for setting up the digital pins of the microcontroller to either input or output mode. This driver will be used to control the buttons and LEDs.

Functions:

```
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_TogglePin(DIO_PIN_TYPE pin);
```

2. Timer Driver:

Description: The Timer driver is responsible for setting up and controlling the timers of the microcontroller. This driver will be used to create the timing delays required in the project.

Functions:

//timer 0 prototypes

```
Timer_ErrorStatus TIMER_0_init(Timer_Mode mode);
Timer_ErrorStatus TIMER_0_start(Timer_Prescaler prescaler);
void TIMER_0_stop(void);
Timer_ErrorStatus TIMER_0_setIntialValue(uint8_t value);
Timer_ErrorStatus TIMER_0_OvfNum(double overflow);
void TIMER_0_DELAY_MS(double _delay);
```

//timer 2 prototypes

```
Timer_ErrorStatus TIMER_2_init(Timer_Mode mode);
Timer_ErrorStatus TIMER_2_start(Timer_Prescaler prescaler);
void TIMER_2_stop(void);
Timer_ErrorStatus TIMER_2_setIntialValue(uint8_t value);
Timer_ErrorStatus TIMER_2_OvfNum(double overflow);
void TIMER_2_DELAY_MS(double _delay);
void TIMER_2_INT();

//PWM Function prototype

void TIMER_0_pwm(float intial);
```

3. ICU

Description: The ICU driver is responsible for interfacing the ATmega32 microcontroller with the ultrasonic sensor in the object detection robot project. The ultrasonic sensor measures the distance between the robot and objects in its path by emitting ultrasonic waves and measuring the time taken for the waves to bounce back.

Functions:

```
Void ICU_init (void);
Float ICU_getTime (u16 a_Time);
```

4. External interrupt:

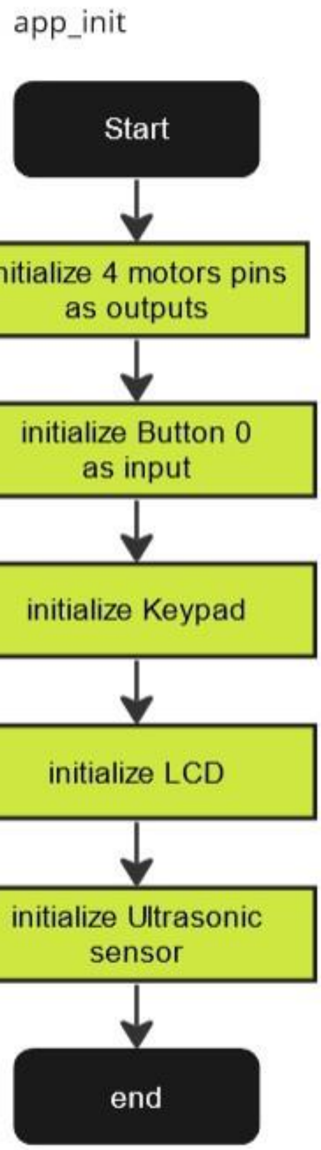
Description: This driver enables the microcontroller to detect and respond to external events from sensors, switches, or other devices. It provides an interface between the microcontroller and the external interrupt hardware, allowing the microcontroller to quickly respond to important events and take appropriate action.

Functions:

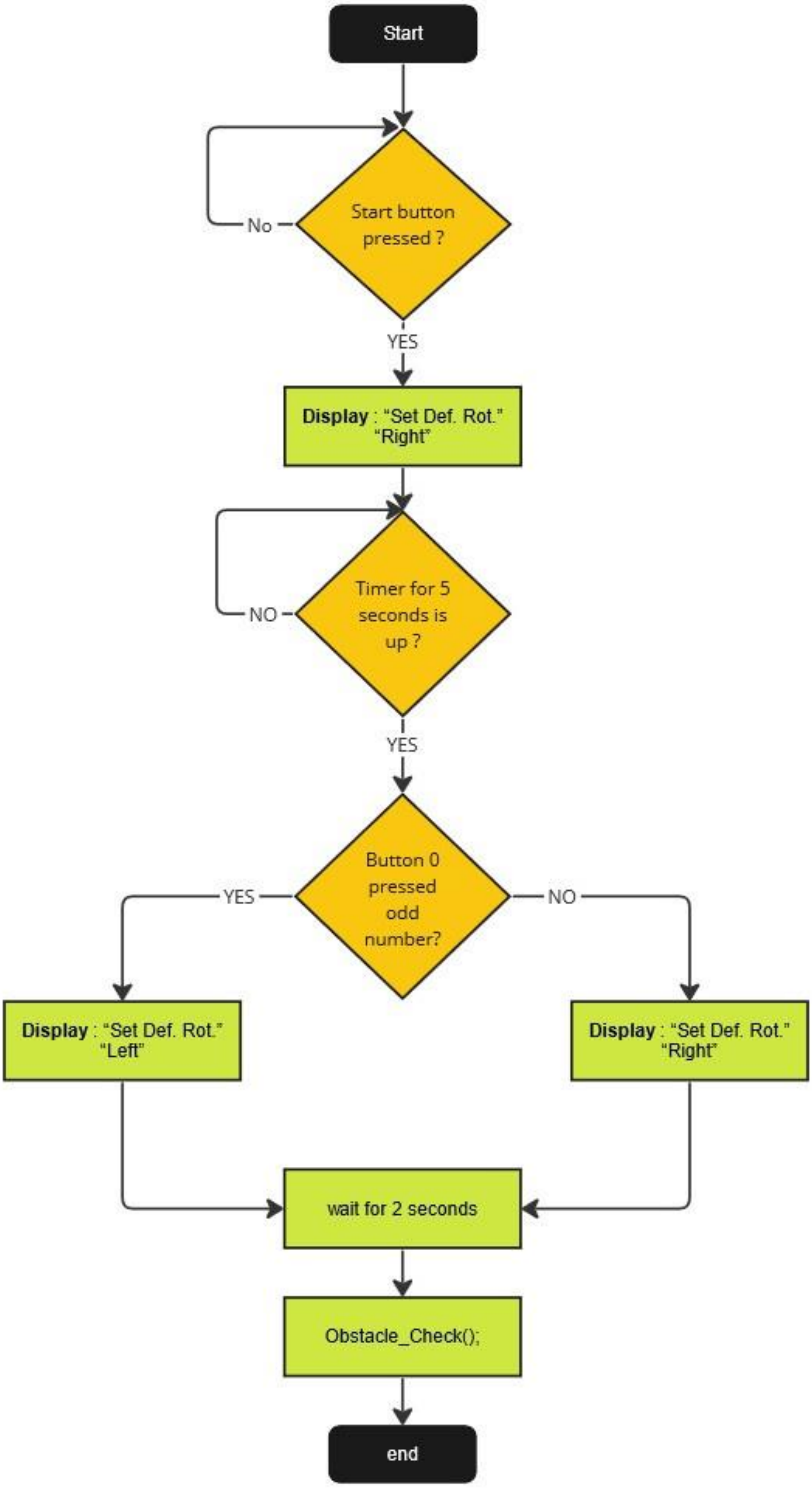
```
EN_int__error_t EXI_Enable (EN_int_t Interrupt);
EN_int__error_t EXI_Disable (EN_int_t Interrupt);
EN_int__error_t EXI_Trigger(EN_int_t Interrupt,EN_trig trigger);
void EXI_SetCallBack(EN_int_t Interrupt,void(*ptrf)(void));
```

Flowcharts for Functions from Higher layers downwards:

App layer functions flowcharts:

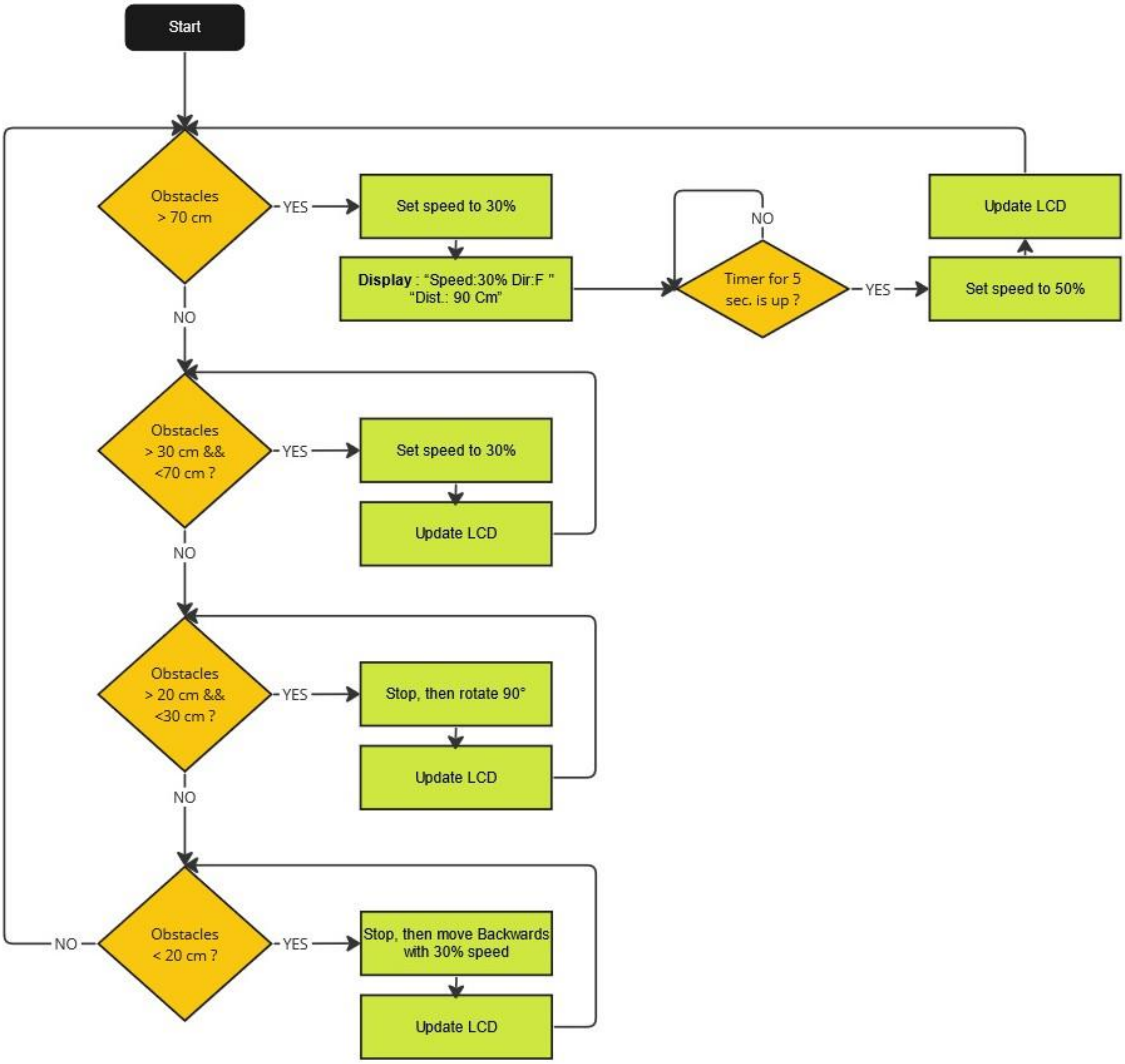

app_init
Start
initialize 4 motors pins as outputs
initialize Button 0 as input
initialize Keypad
initialize LCD
initialize Ultrasonic sensor
end

Start_Stage



Start
Start button pressed ?
No
YES
Display : "Set Def. Rot." "Right"
Timer for 5 seconds is up ?
NO
YES
Button 0 pressed odd number?
YES
NO
Display : "Set Def. Rot." "Left"
Display : "Set Def. Rot." "Right"
wait for 2 seconds
Obstacle_Check();
end

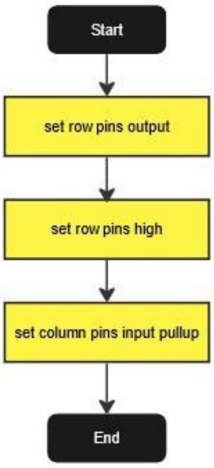
Obstacle_Check();



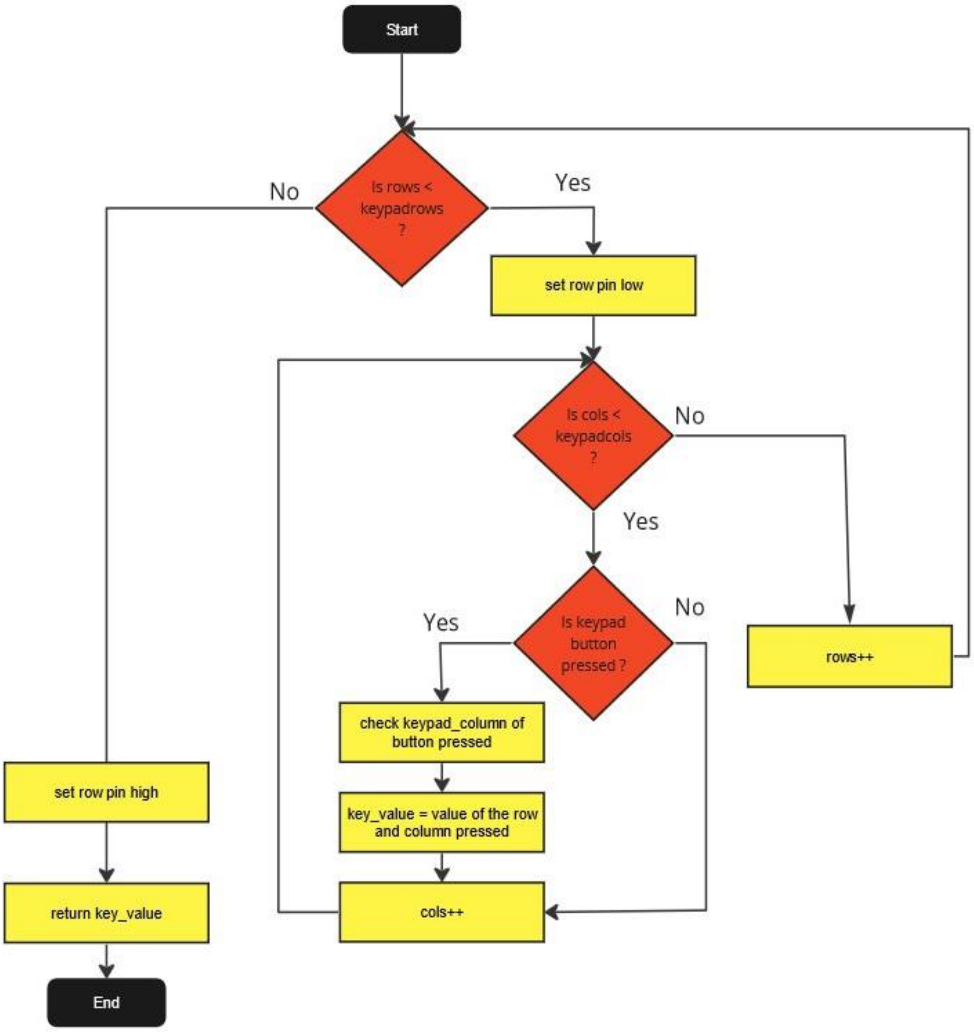
HAL Layer:

Keypad functions:-

```
void KEYPAD_init(void);
```

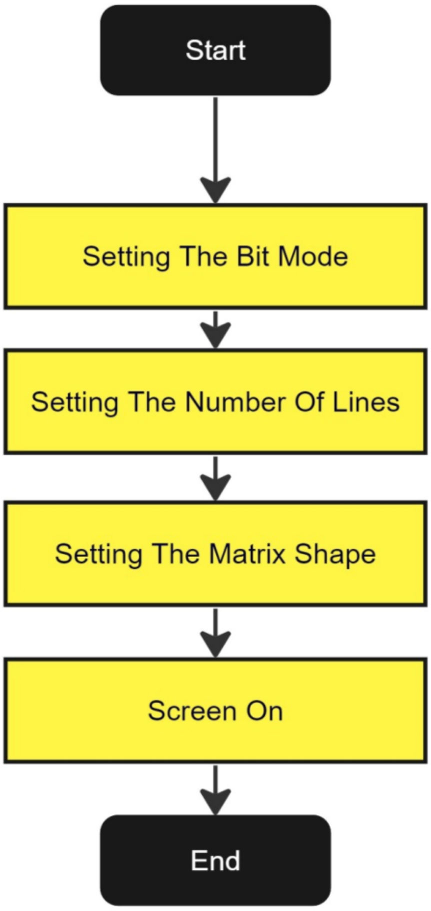


```
uint8_t KEYPAD_getKey(void);
```

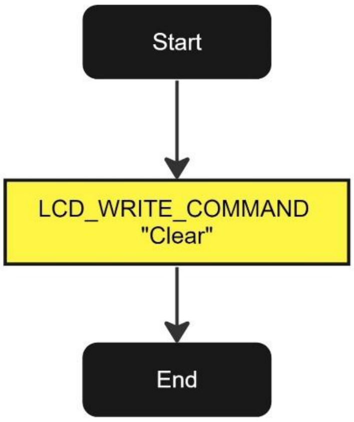


LCD functions:-

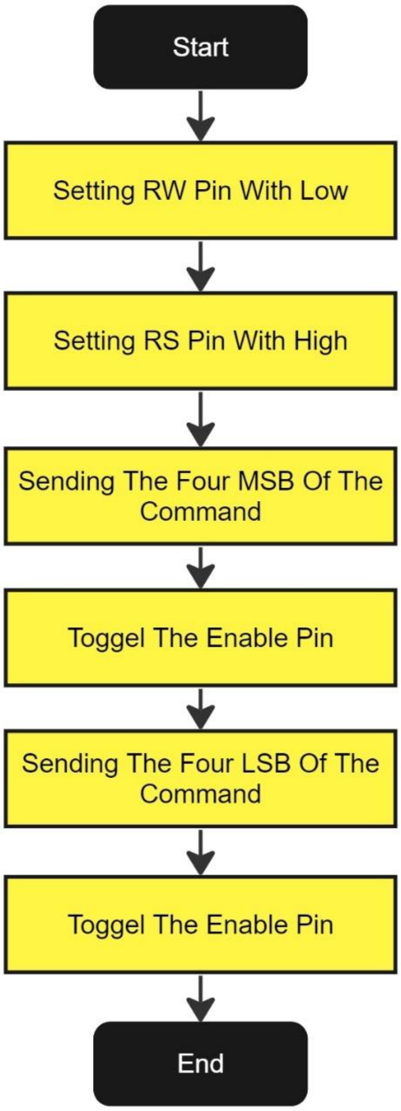
LCD_INIT



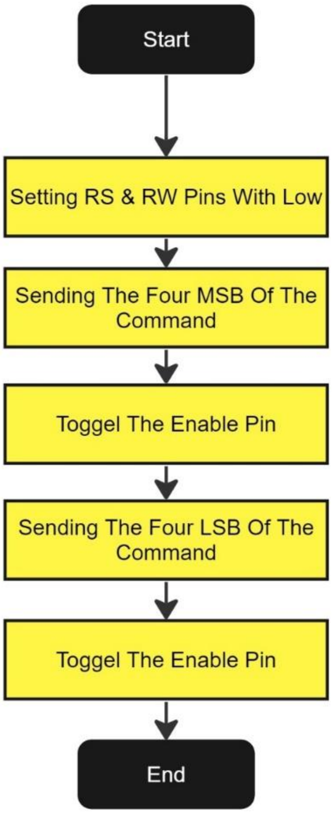
LCD_CLEAR



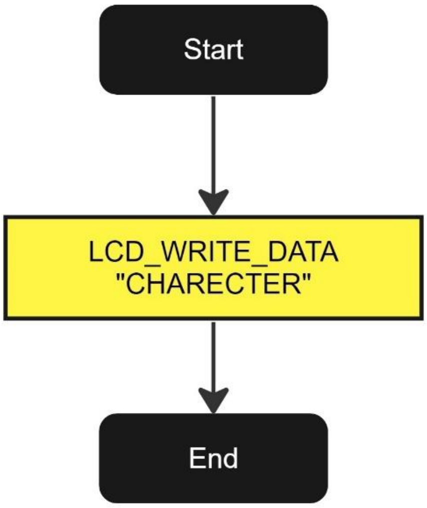
LCD_WRITE_DATA



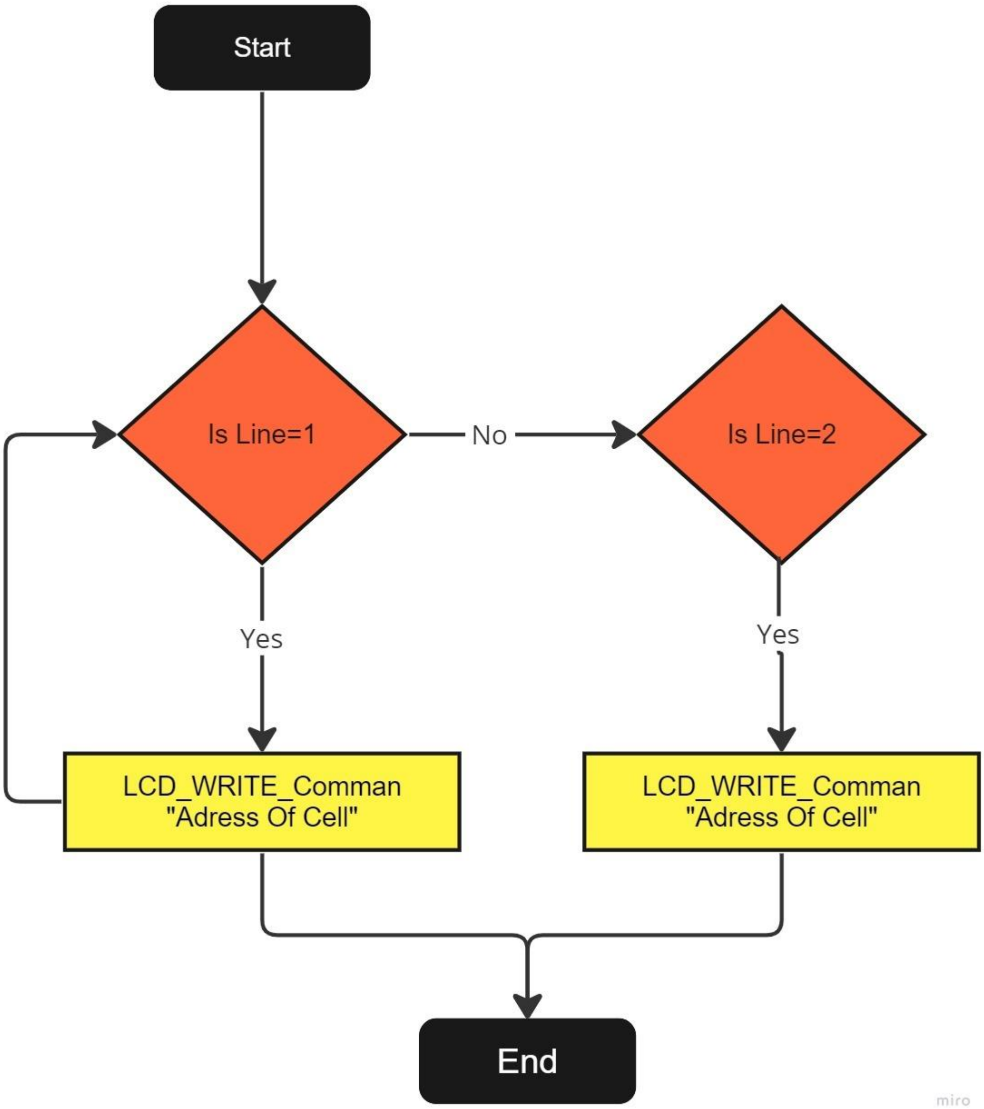
LCD_WRITE_COMMAND

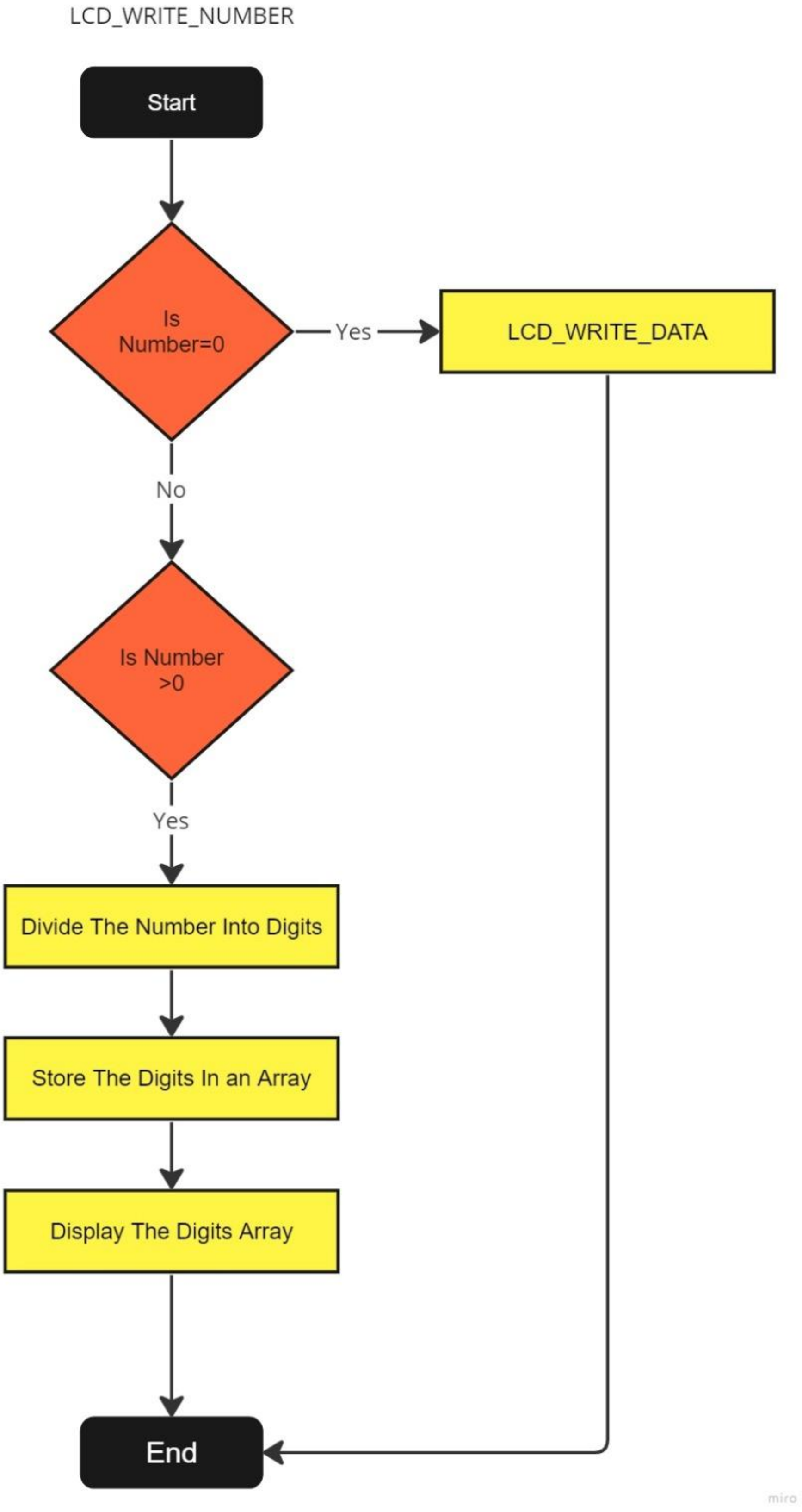

Start
Setting RS & RW Pins With Low
Sending The Four MSB Of The Command
Toggel The Enable Pin
Sending The Four LSB Of The Command
Toggel The Enable Pin
End

LCD_Write_Charecter

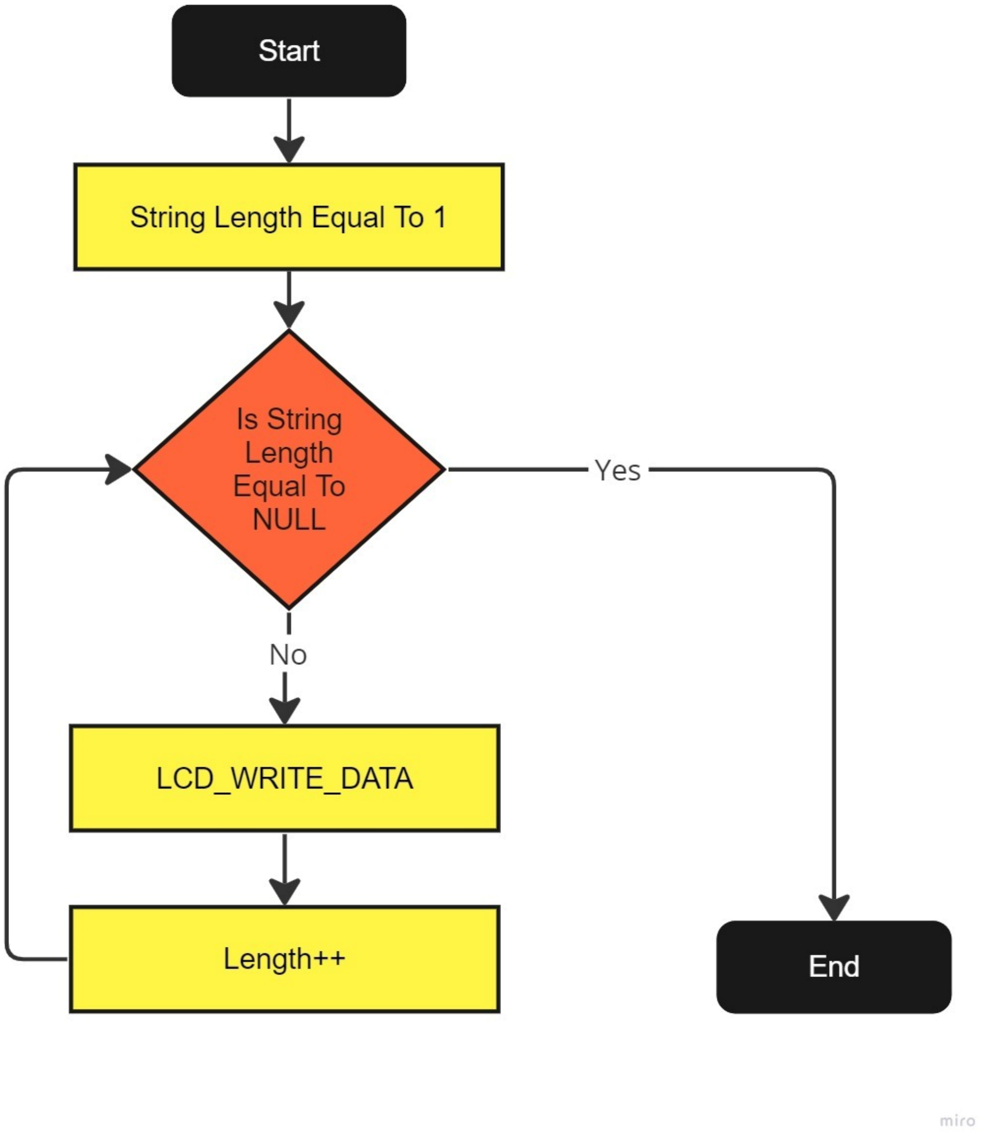


LCD_GoTo

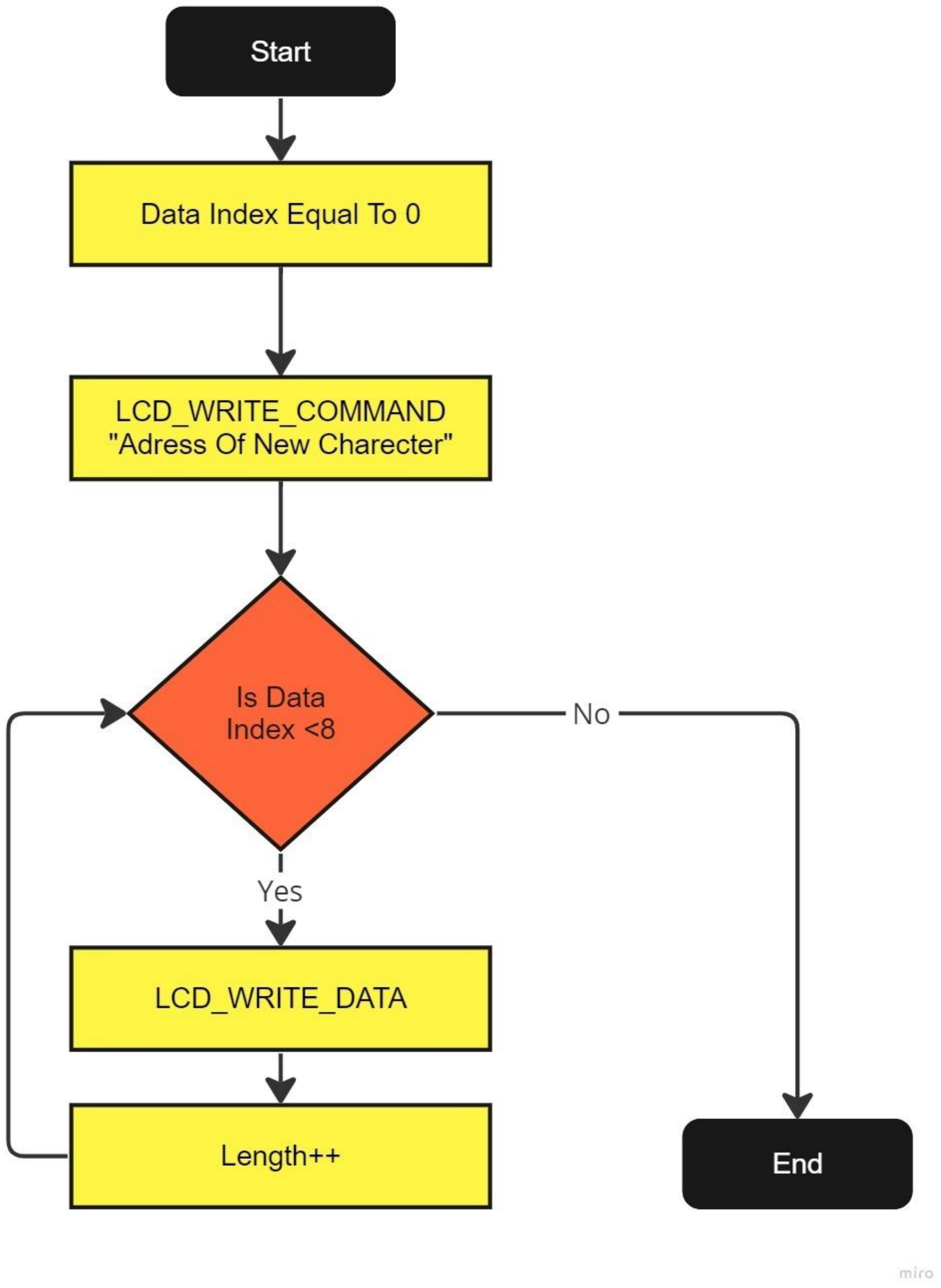

Start
Is Line=1
No
Is Line=2
Yes
Yes
LCD_WRITE_Comman "Adress Of Cell"
LCD_WRITE_Comman "Adress Of Cell"
End
miro


LCD_WRITE_NUMBER
Start
Is Number=0
Yes
LCD_WRITE_DATA
No
Is Number >0
Yes
Divide The Number Into Digits
Store The Digits In an Array
Display The Digits Array
End
miro

LCD_Write_String



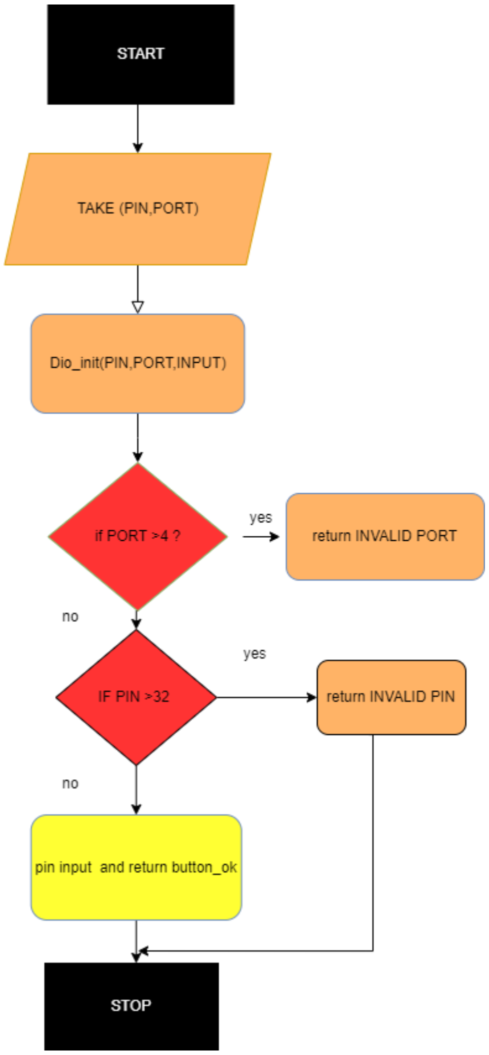
LCD_Create_Charecter



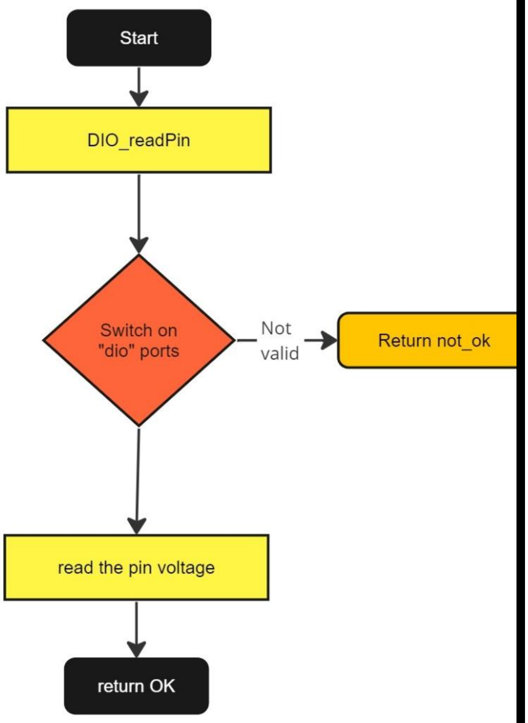
Button functions flowcharts:-

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

```
BUTTON_ERROR_TYPE Button_INIT(DIO_PIN_TYPE PIN);
```

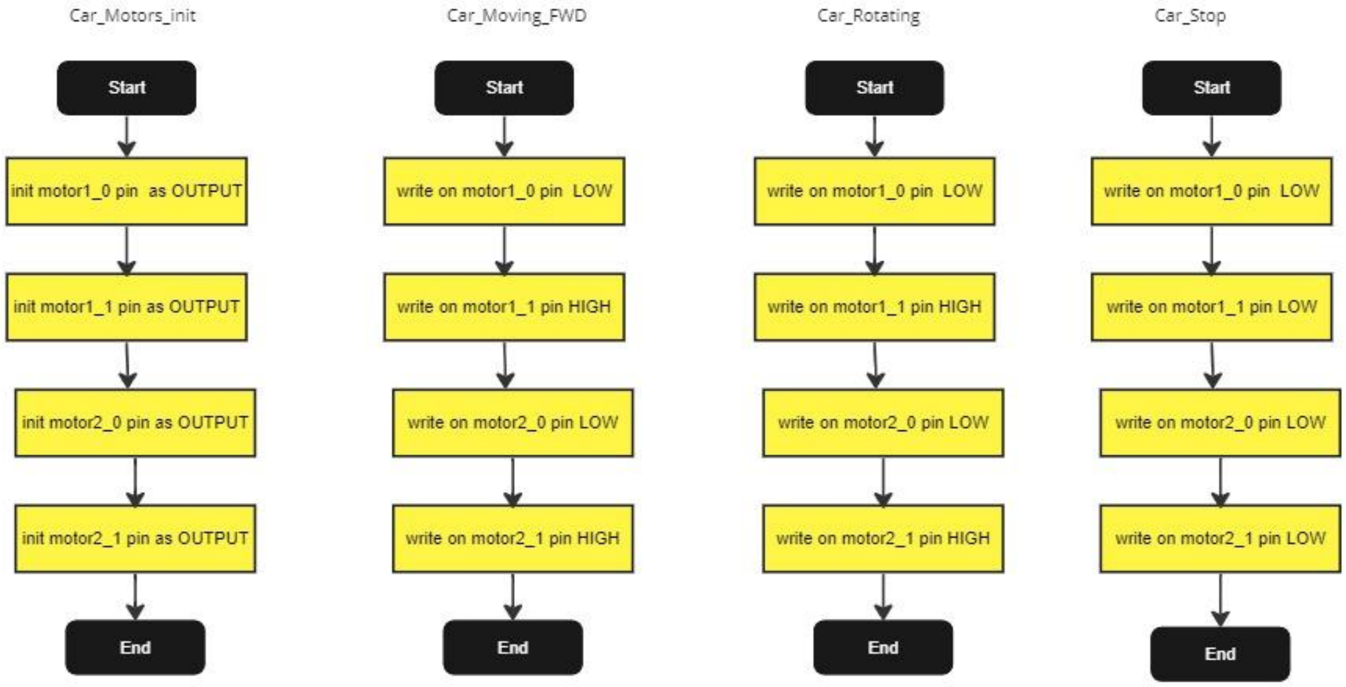


```
BUTTON_ERROR_TYPE Button_read(DIO_PIN_TYPE PIN,DIO_VOLTAGE_TYPE VOLT);
```



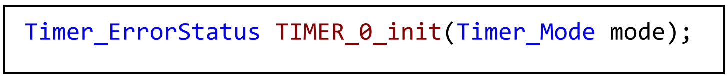
Motor functions flowcharts:-

```
En_motorError_t Motors_init(void);
En_motorError_t Motors_Start(void);
En_motorError_t Motors_Rotating(void);
En_motorError_t Motors_Stop(void);
```

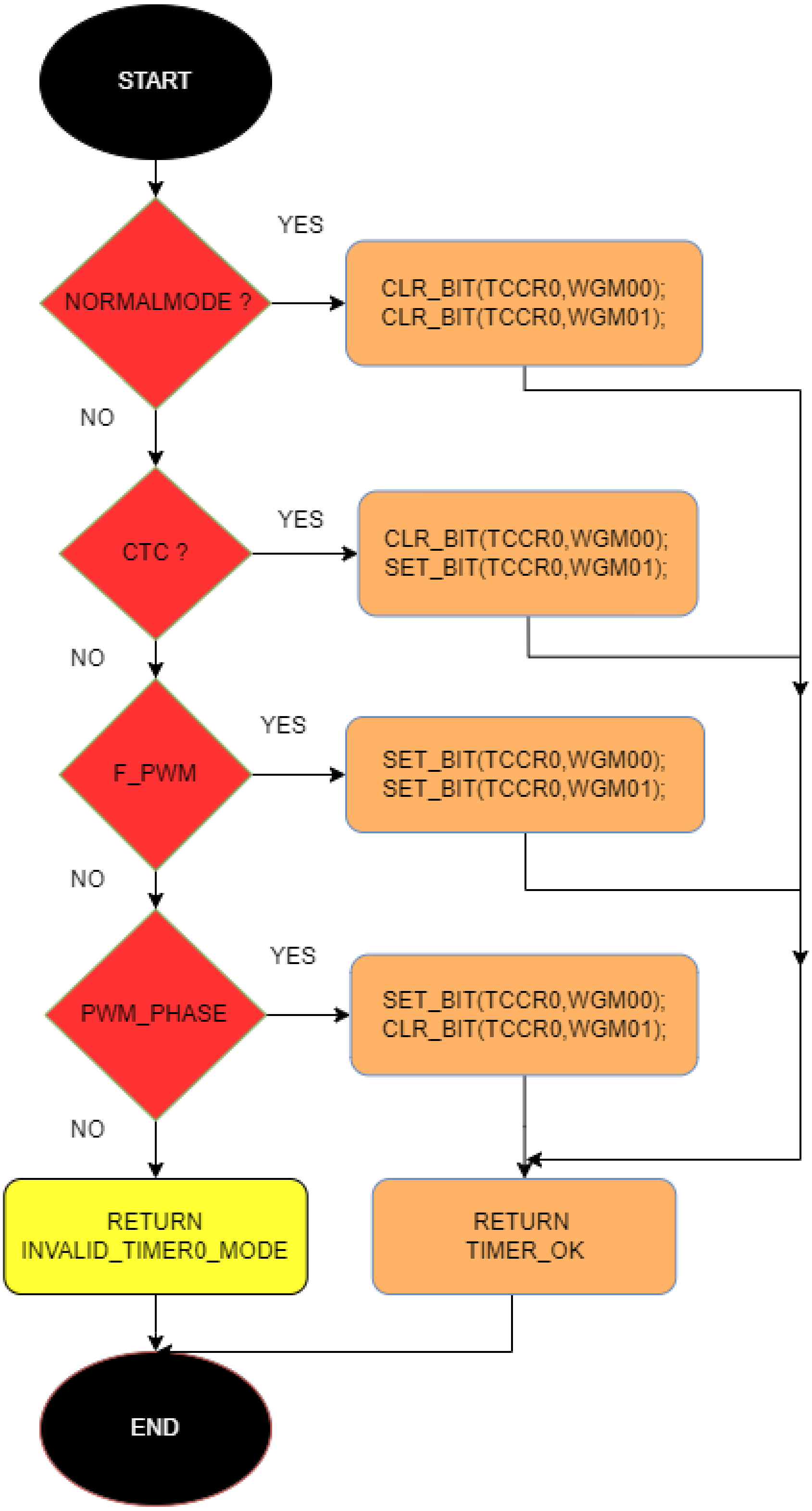

Car_Motors_init
Start
init motor1_0 pin as OUTPUT
init motor1_1 pin as OUTPUT
init motor2_0 pin as OUTPUT
init motor2_1 pin as OUTPUT
End
Car_Moving_FWD
Start
write on motor1_0 pin LOW
write on motor1_1 pin HIGH
write on motor2_0 pin LOW
write on motor2_1 pin HIGH
End
Car_Rotating
Start
write on motor1_0 pin LOW
write on motor1_1 pin HIGH
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

MCAL Layer:-

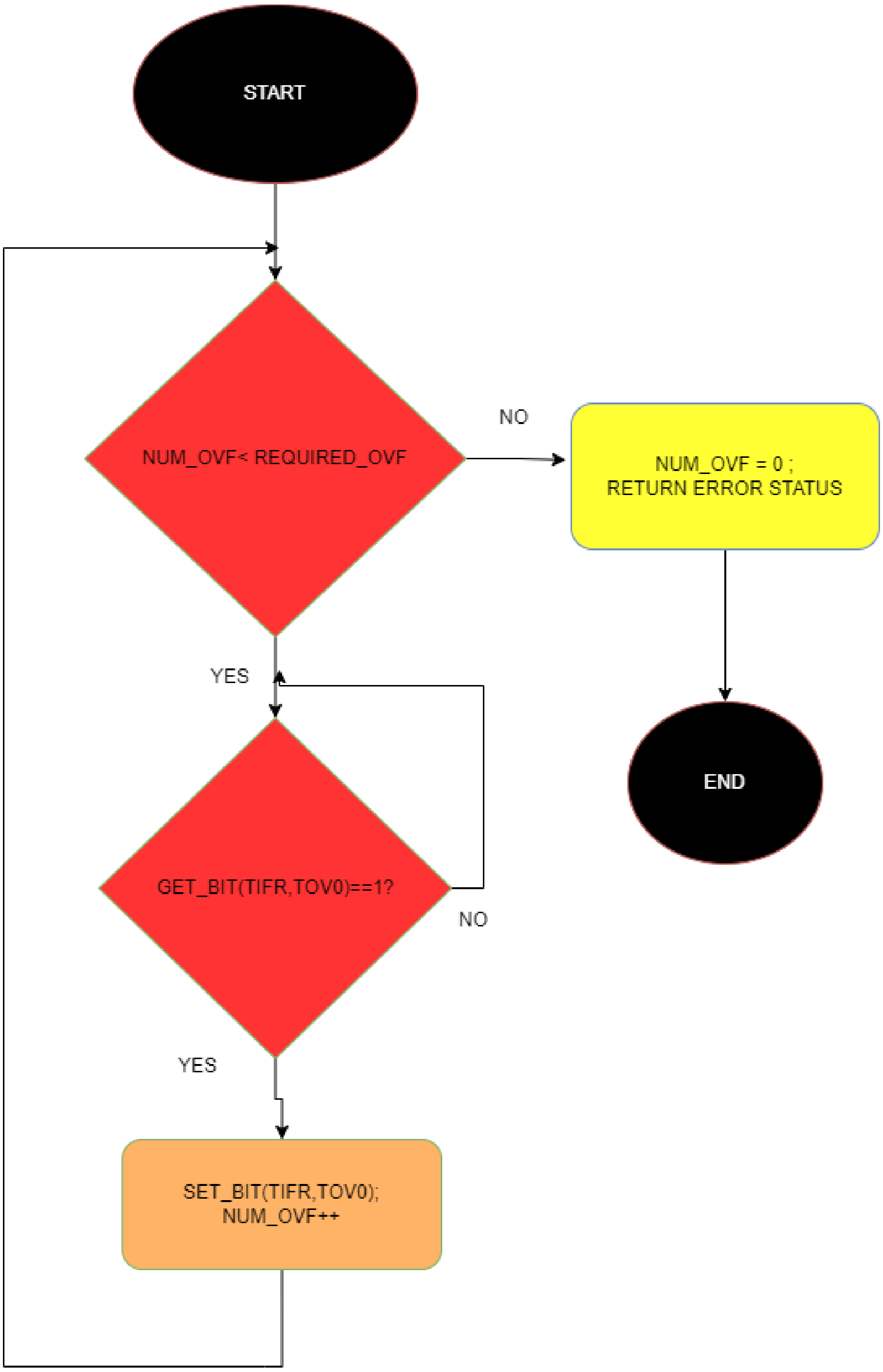
Timer functions’ flowcharts:-



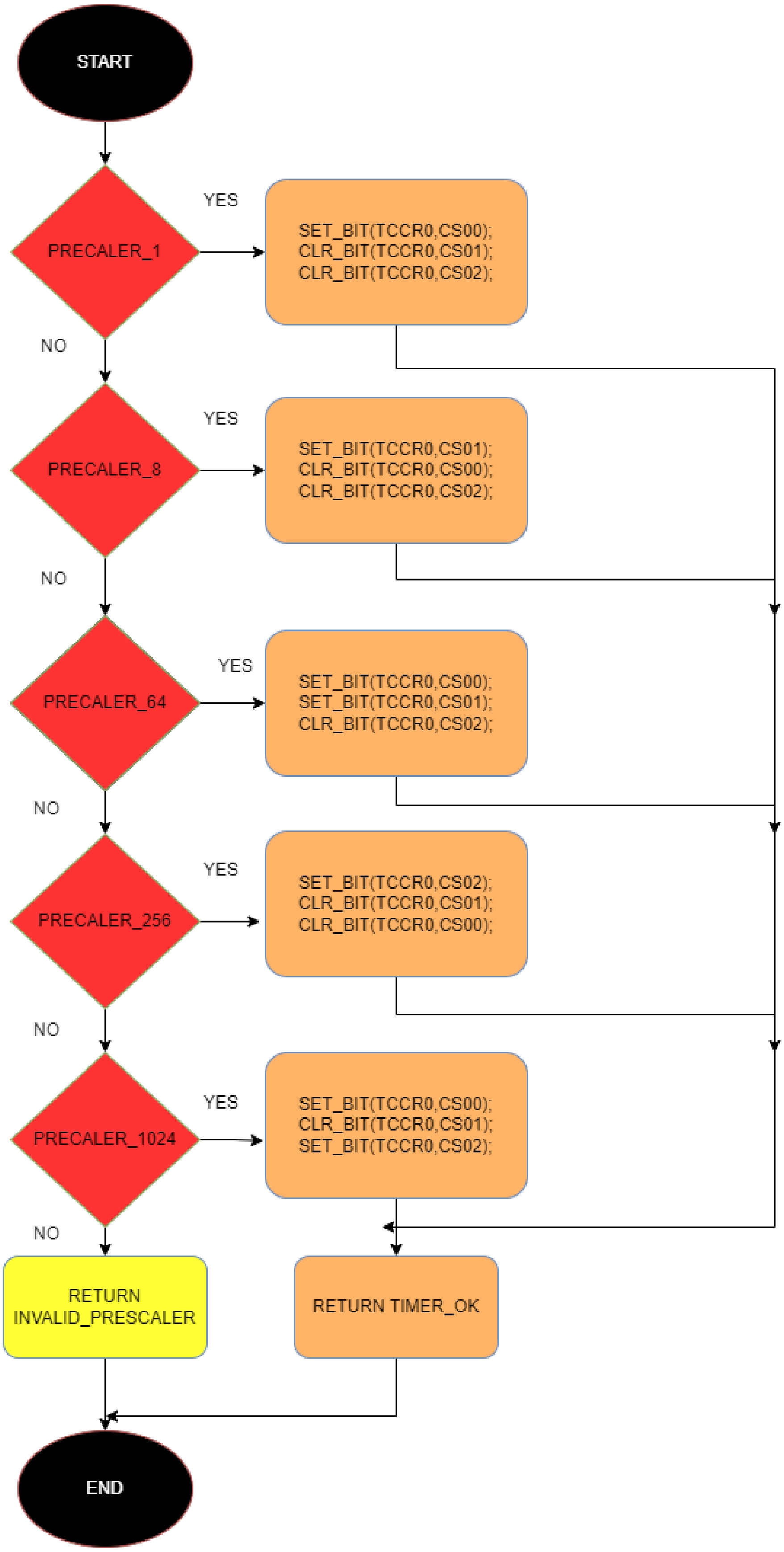
```
Timer_ErrorStatus TIMER_0_init(Timer_Mode mode);
```

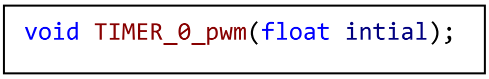


```
Timer_ErrorStatus TIMER_0_OvfNum(double overflow);
```

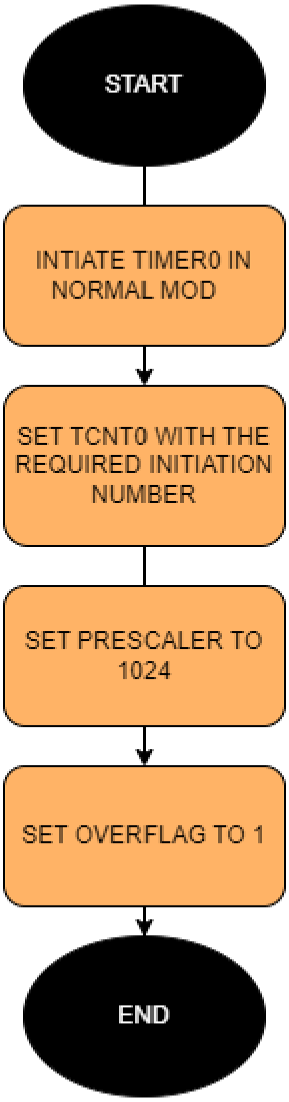


```
Timer_ErrorStatus TIMER_0_start(Timer_Prescaler prescaler);
```



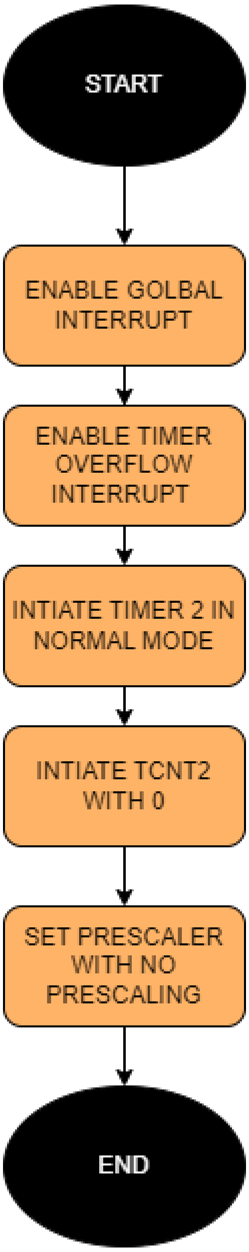


```
void TIMER_0_pwm(float intial);
```



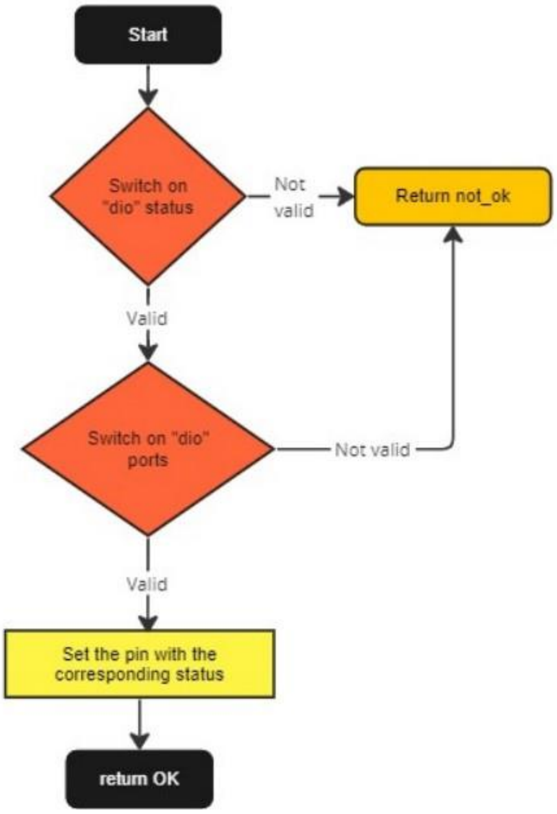
START
INTIATE TIMER0 IN NORMAL MOD
SET TCNT0 WITH THE REQUIRED INITIATION NUMBER
SET PRESCALER TO 1024
SET OVERFLAG TO 1
END

TIMER 2 WITH INTERRUPT

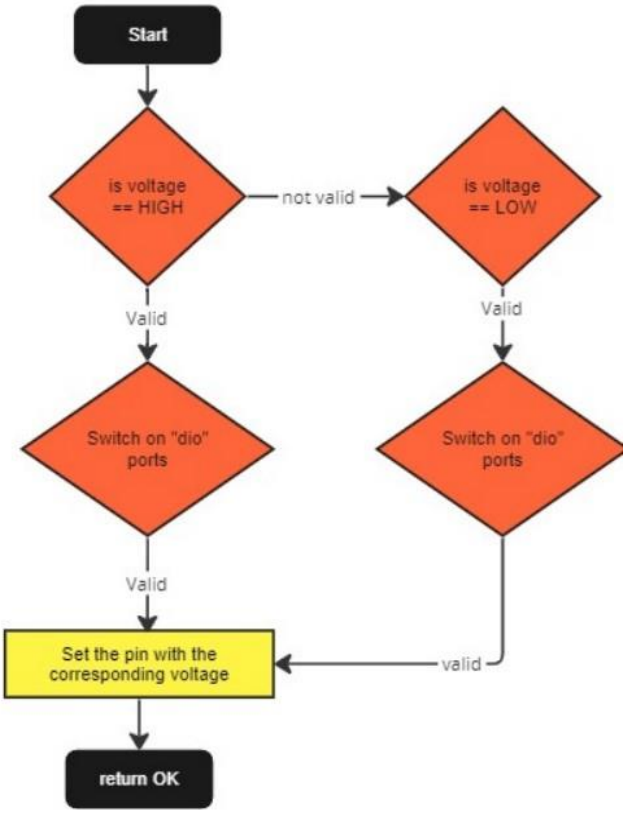

START
ENABLE GOLBAL INTERRUPT
ENABLE TIMER OVERFLOW INTERRUPT
INTIATE TIMER 2 IN NORMAL MODE
INTIATE TCNT2 WITH 0
SET PRESCALER WITH NO PRESCALING
END

DIO functions flowcharts:-

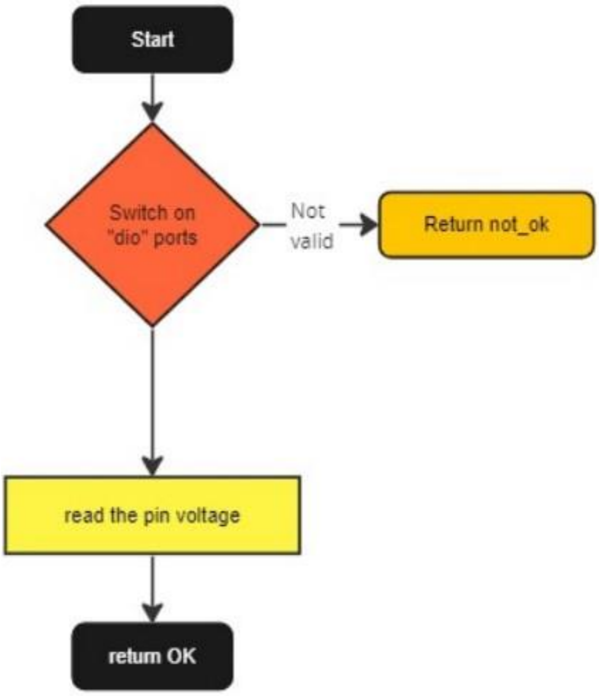
DIO_INITPIN



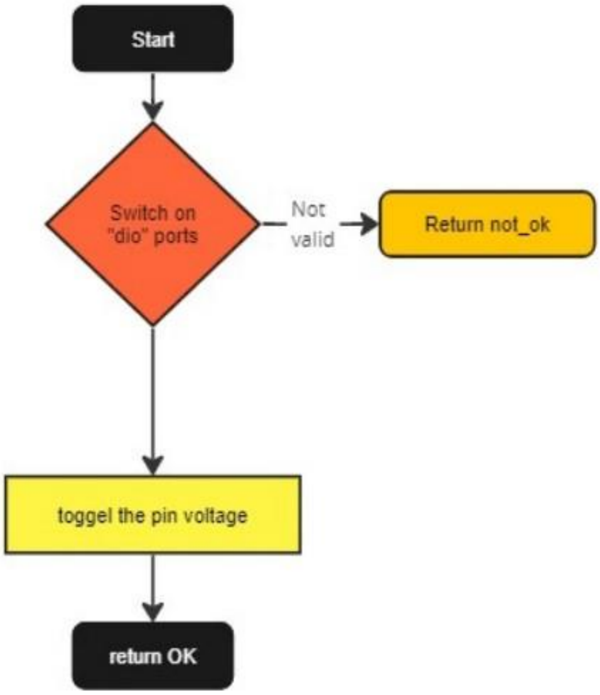
DIO_WRITEPIN



DIO_READPIN

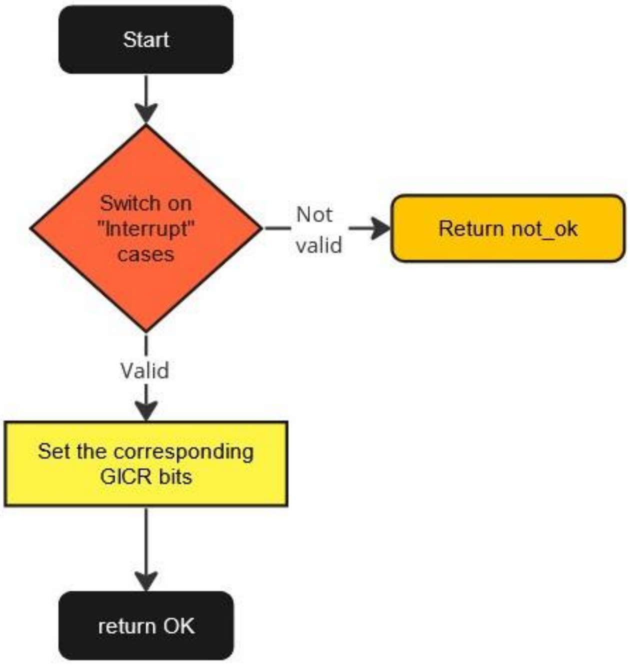


DIO_TogglePin

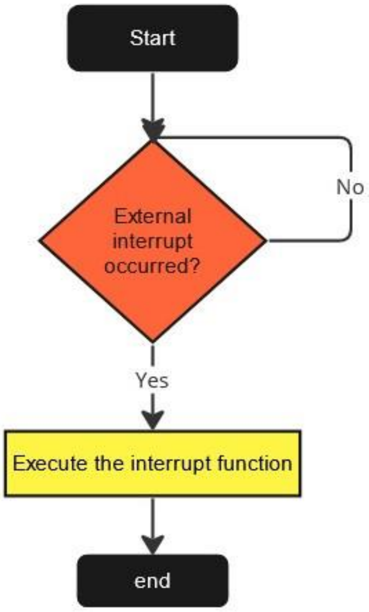


Interrupt functions’ flowcharts

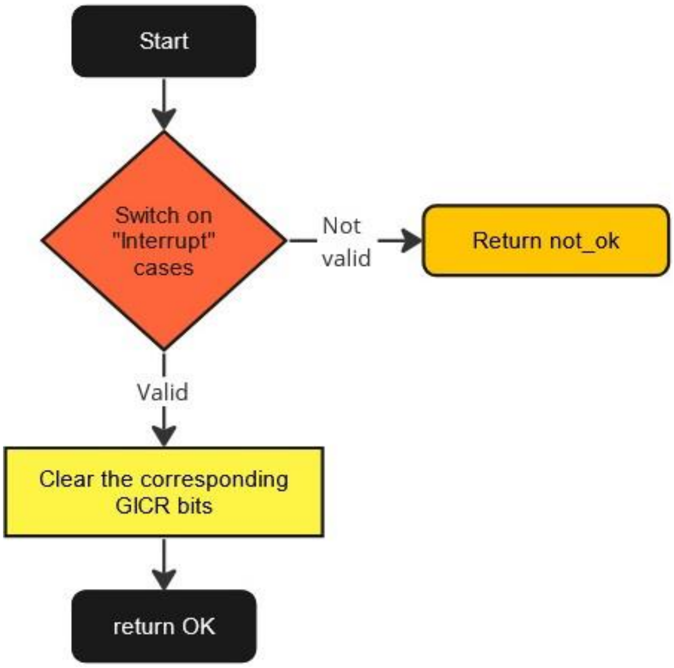
`EN_int__error_t EXI_Enable (EN_int_t Interrupt)`



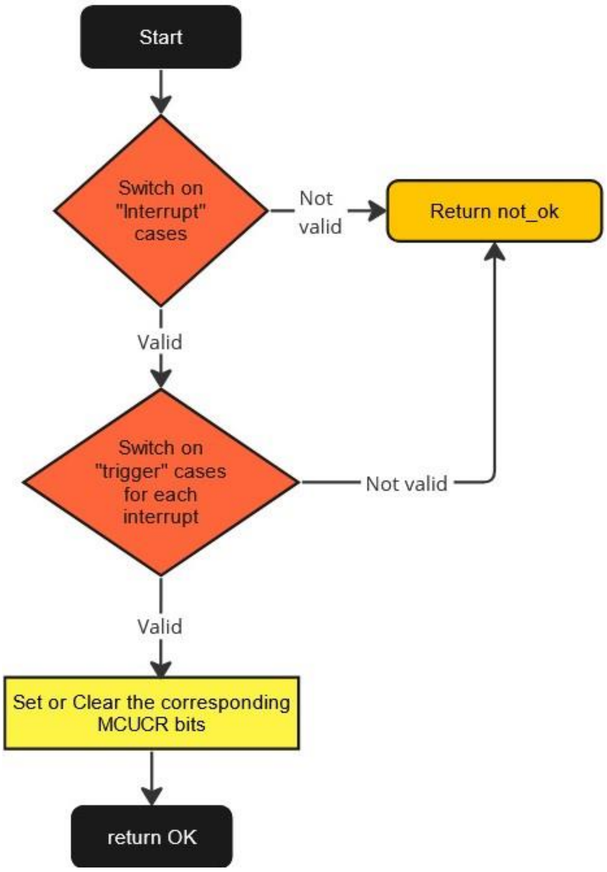
`ISR (INTx_vect)`



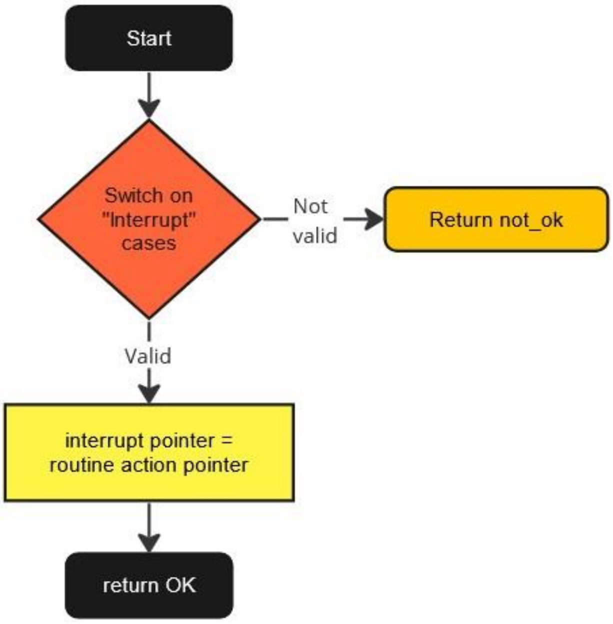
`EN_int__error_t EXI_Disable (EN_int_t Interrupt)`


Start
Switch on "Interrupt" cases
Not valid
Return not_ok
Valid
Clear the corresponding GICR bits
return OK

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


Start
Switch on "Interrupt" cases
Not valid
Return not_ok
Valid
Switch on "trigger" cases for each interrupt
Not valid
Valid
Set or Clear the corresponding MCUCR bits
return OK

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


Start
Switch on "Interrupt" cases
Not valid
Return not_ok
Valid
interrupt pointer = routine action pointer
return OK