DESIGN DOCUMENT

SIMPLE ATM

MACHINE DESIGN

Team7:

-Amr El-Abd

-Kareem Magdy

-Nada Abdelazim

-Moustafa Abdelrahim

Table of Contents: -

Subject **Page**

Project introduction:

This project aims to design and develop an Automated Teller Machine (ATM) system that simulates the main functionalities of a real-world ATM, including card insertion, PIN verification, transaction authorization, and balance update. The system consists of two microcontroller units (MCUs) - one for the ATM and the other for the card - that communicate with each other to execute the transaction process.

Project description :-

The system's functionality can be summarized in the following points:

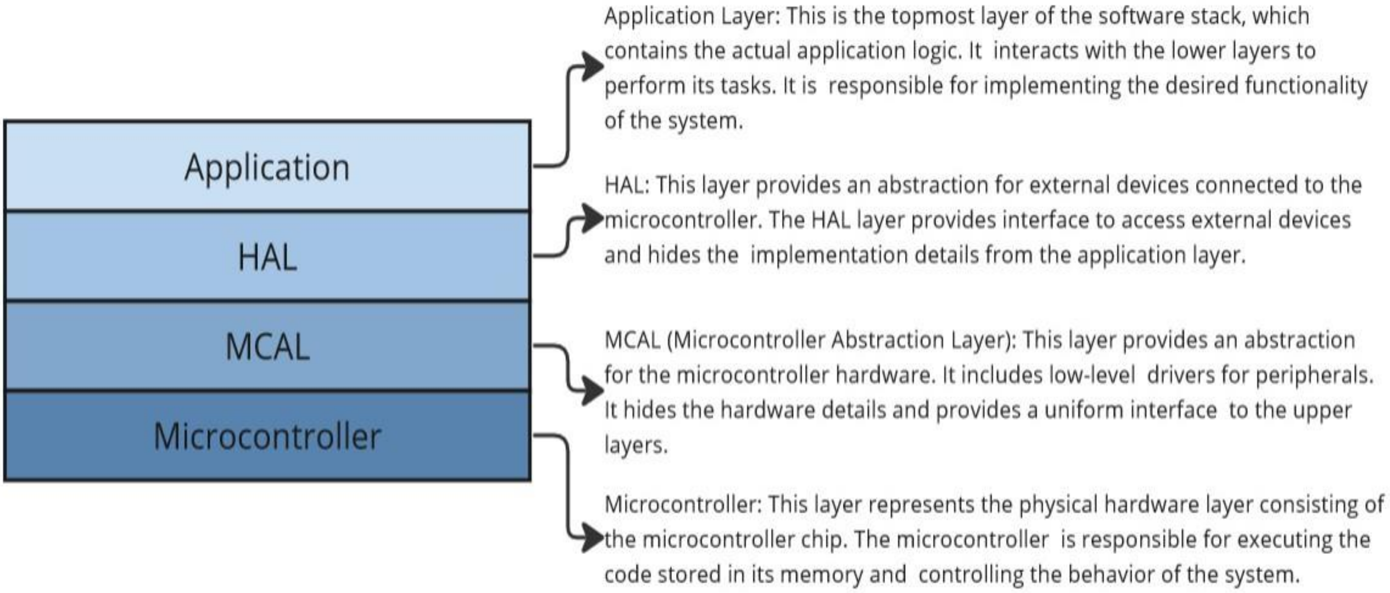
- The ATM MCU will handle the transaction main flows, including displaying welcome messages, requesting card insertion, entering PIN, and validating transactions.
- The CARD MCU has two modes of operation - programming and user mode. In programming mode, the MCU will request the user to enter the card PAN and PIN and store them in the EEPROM. In user mode, the CARD MCU will send a trigger signal to the ATM MCU to initiate the transaction flow.
- The project uses a hard-coded array of structures for accounts that contain the PAN, account state (blocked/running), and balance.
- The maximum allowed limit for a transaction is hardcoded as $5000.00.
- The system will perform several checks on the database before finalizing the transaction, including verifying the card PAN, checking if the card is blocked, checking if the amount required exceeds the maximum daily limit, and verifying the available balance.
- If all checks pass, the system will display the remaining balance and eject the card. If any check fails, the system will display a declined message accordingly and initiate an alarm if necessary.

Hardware components:

- ATM ECU
  1. ATM MCU
  2. 16 x 2 LCD
  3. 3 x 3 Keypad
  4. Buzzer
  5. Enter/Set Button

- CARD ECU
  1. CARD MCU
  2. EEPROM
  3. Serial Terminal

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.

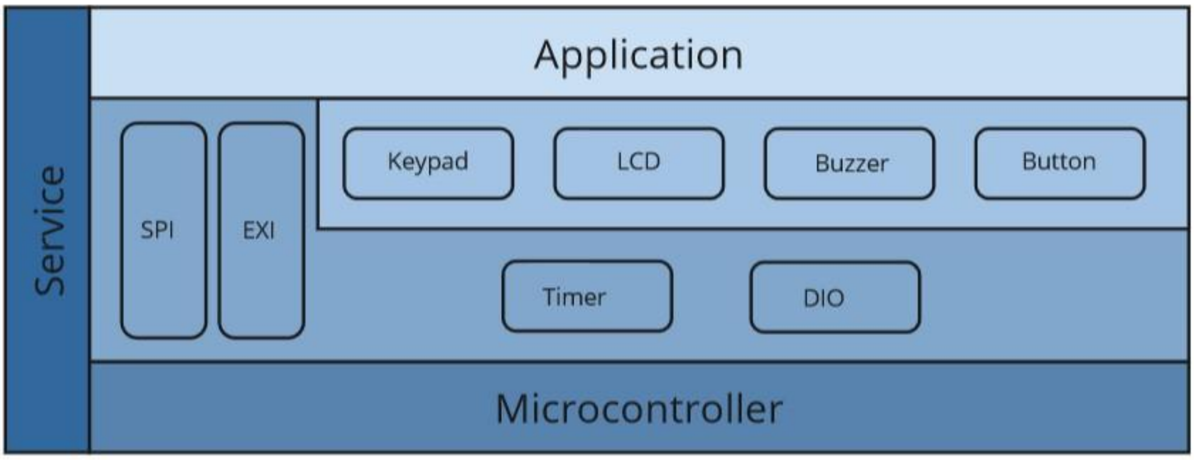
HAL: This layer provides an abstraction for external devices connected to the microcontroller. The HAL layer provides interface to access external devices and hides the implementation details from the application layer.

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.

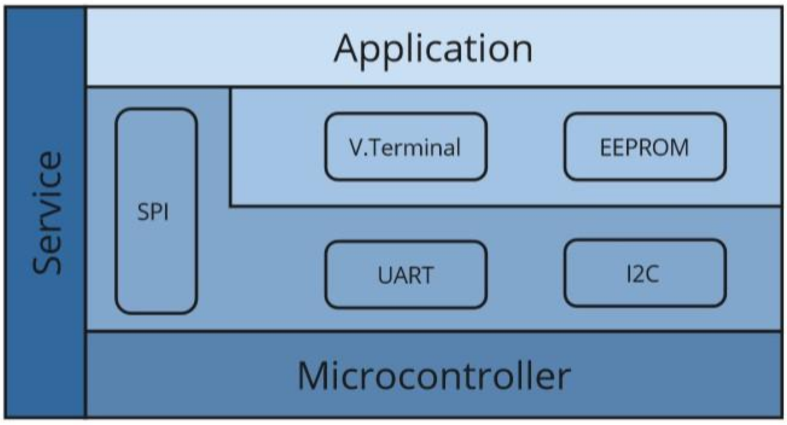
Microcontroller: This layer represents the physical hardware layer consisting of the microcontroller chip. The microcontroller is responsible for executing the code stored in its memory and controlling the behavior of the system.

System modules:-

ATM MCU



CARD MCU



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. Buzzer Driver:

Description: This driver controls the buzzer and provides an interface between the microcontroller and the buzzer hardware, allowing the microcontroller to activate and deactivate the buzzer to alert the user when the temperature exceeds the set range.

Functions:

```
void buzzer_init(void);
void buzzer_On(void);
void buzzer_Off(void);
```

4. EEPROM Driver:

Description: This driver enables the microcontroller to interact with EEPROM (Electrically Erasable Programmable Read-Only Memory) chips. It provides functions and APIs to read from and write to the EEPROM, managing the necessary operations for data storage and retrieval.

Functions:

```
void EEPROM_Write (uint8_t *data, uint8_t* _addres);
void EEPROM_Read (uint8_t *data) ;
void EEPROM_Read_String(uint8_t * a_STR);
void EEPROM_Write_WithSize (uint8_t *data, uint8_t *_address,uint8_t LV_Length);
void EEPROM_Read_String_With_size(uint8_t * a_STR, uint8_t a_Length);
```

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. 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));
```

4. SPI communication protocol:-

Description: This driver enables the microcontroller to establish and manage communication with peripheral devices using the SPI protocol. It handles tasks such as configuring the SPI interface, controlling data transfer, and managing the synchronization between the microcontroller and SPI devices. This driver allows the microcontroller to effectively communicate with SPI-based peripherals like sensors, displays, and memory chips, providing a seamless interface for data exchange.

Functions:

```
void SPI_MasterInit(en_SPI_Prescaler prescaler);
void SPI_SlaveInit(en_SPI_Prescaler prescaler);
uint8_t SPI_SendReceive(uint8_t data);
uint8_t SPI_RecievePeriodicChecking(uint8_t * pdata);
```

5. USART communication protocol:-

Description: This driver enables the microcontroller to establish and control serial communication with peripheral devices using the UART protocol. It handles tasks such as configuring the UART interface, setting baud rates, and managing data transmission and reception. This driver enables the microcontroller to communicate with UART-compatible devices, such as wireless modules, GPS receivers, and Bluetooth devices, allowing for reliable and efficient data transfer.

Functions:

```
void USART_init(void);
void USART_transmit(uint8_t data);
uint8_t USART_receive(void);
void USART_Transmit_string (char * str);
char * USART_receive_string (uint8_t *string) ;
```

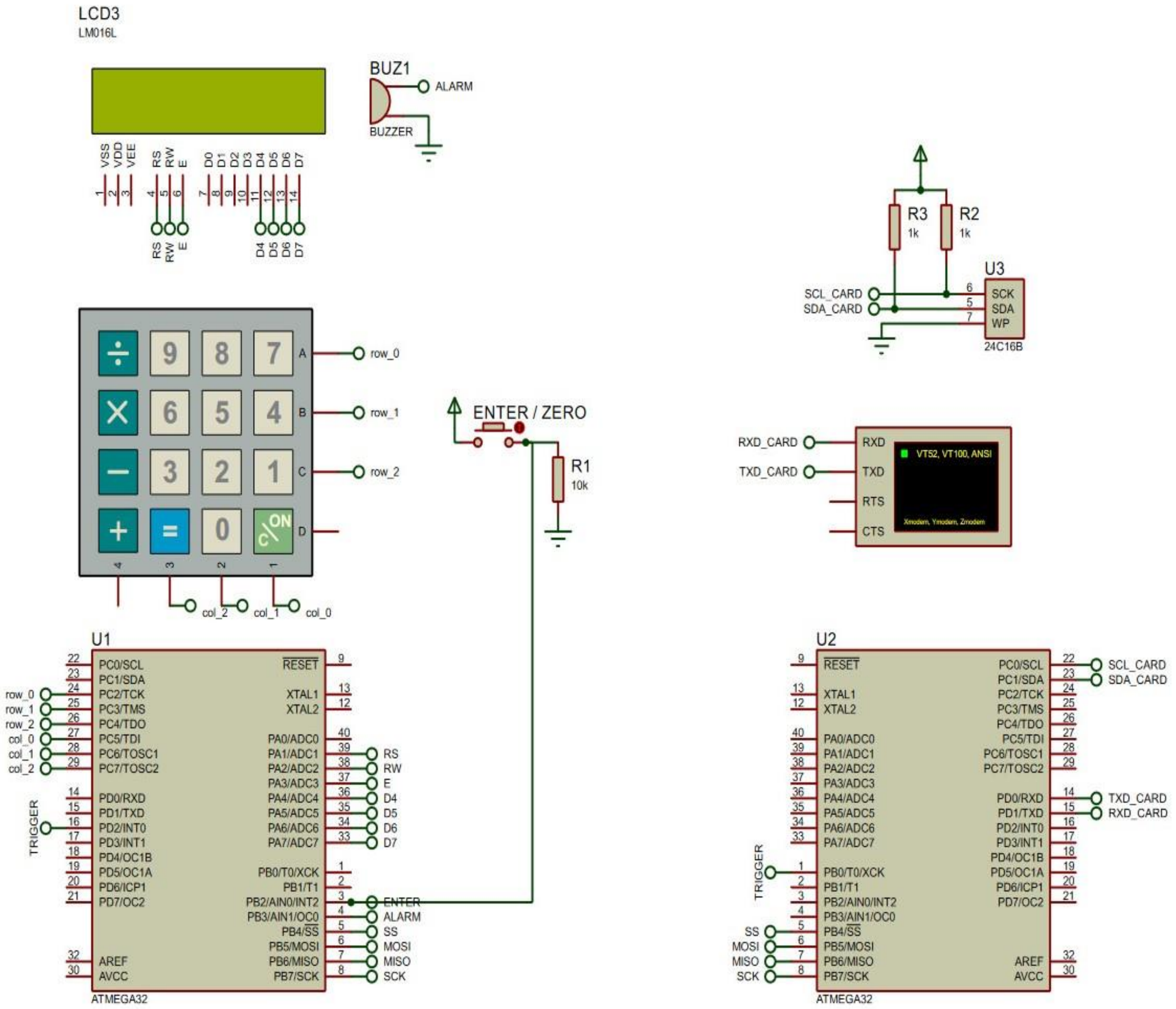
6. I2C communication protocol:-

Description: This driver enables the microcontroller to communicate with peripheral devices using the I2C protocol. It provides functions and utilities for configuring the I2C interface, managing data transfer, and addressing multiple devices on the I2C bus. This driver facilitates communication between the microcontroller and I2C-compatible devices, including sensors, real-time clocks, and EEPROMs.

Functions:

```
void I2C_init(I2C_PRESCALER prescaler);
uint8_t I2C_start();
uint8_t I2C_addressEvent(uint8_t a_address ,R_W r_w );
uint8_t I2C_sendData(uint8_t *data);
uint8_t I2C_receiveData(uint8_t *data, ACKOLEDGMENT ack);
uint8_t I2C_dataEvent(uint8_t *data ,uint8_t s_r , uint8_t ack );
void I2C_stop();
uint8_t I2c_Restart();
```

Proteus simulation design :-

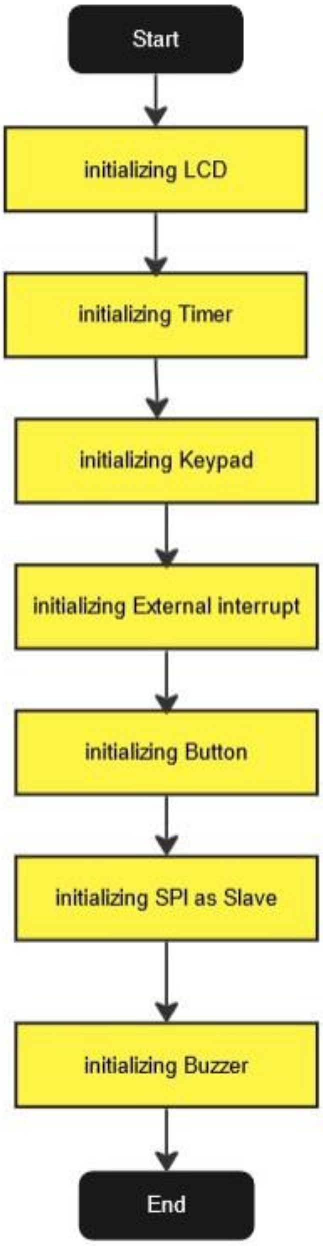


Flowcharts for Functions from Higher layers downwards:

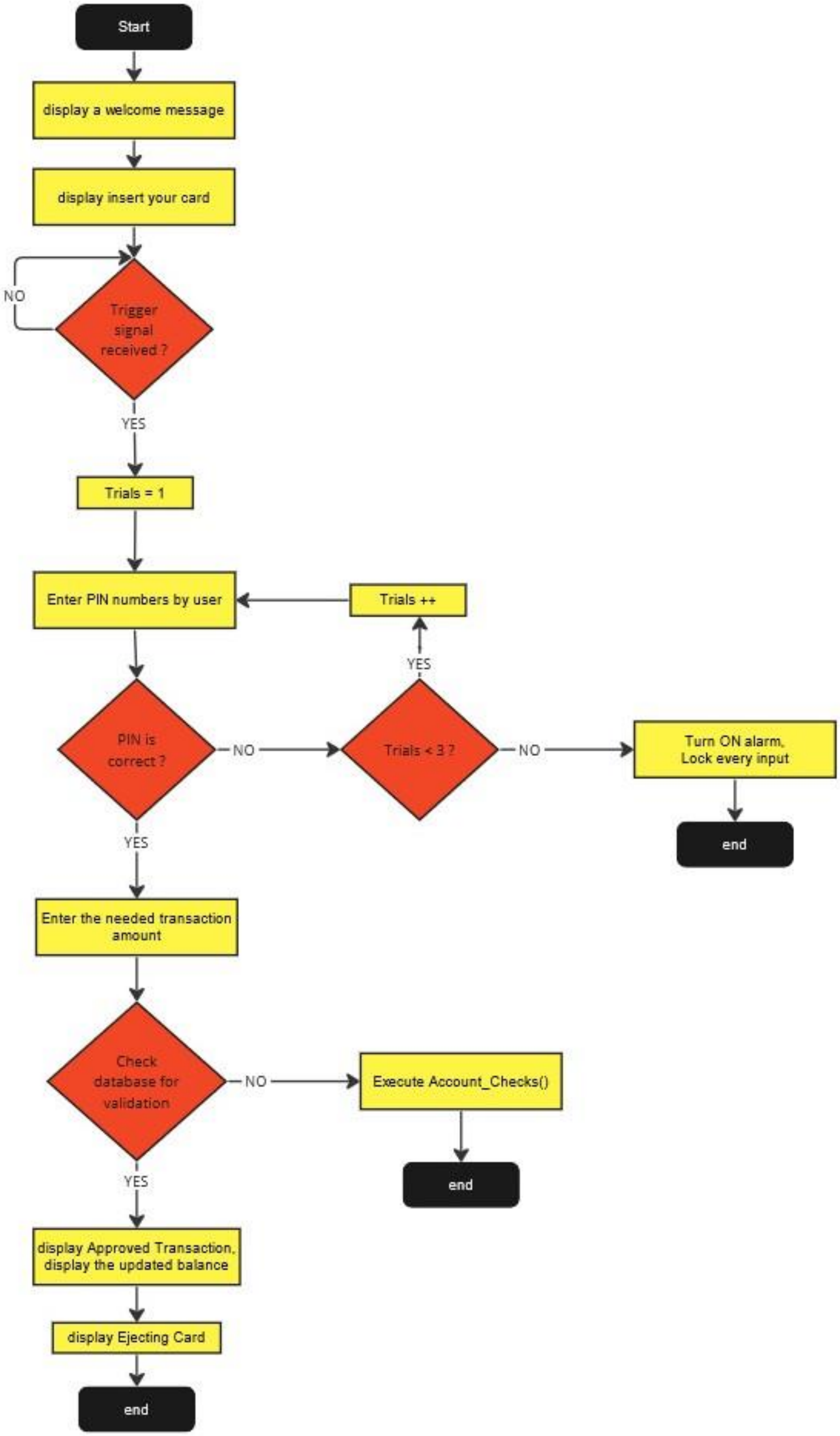
App layer functions flowcharts:

ATM APP:

```
void ATM_app_Init(void);
```

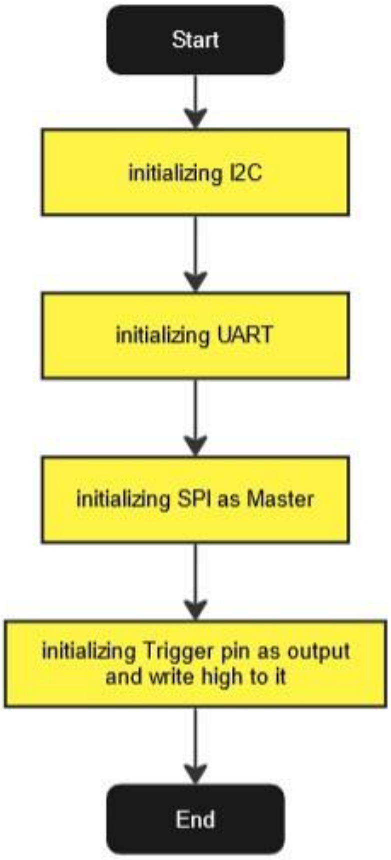


```
void ATM_app_Start(void);
```

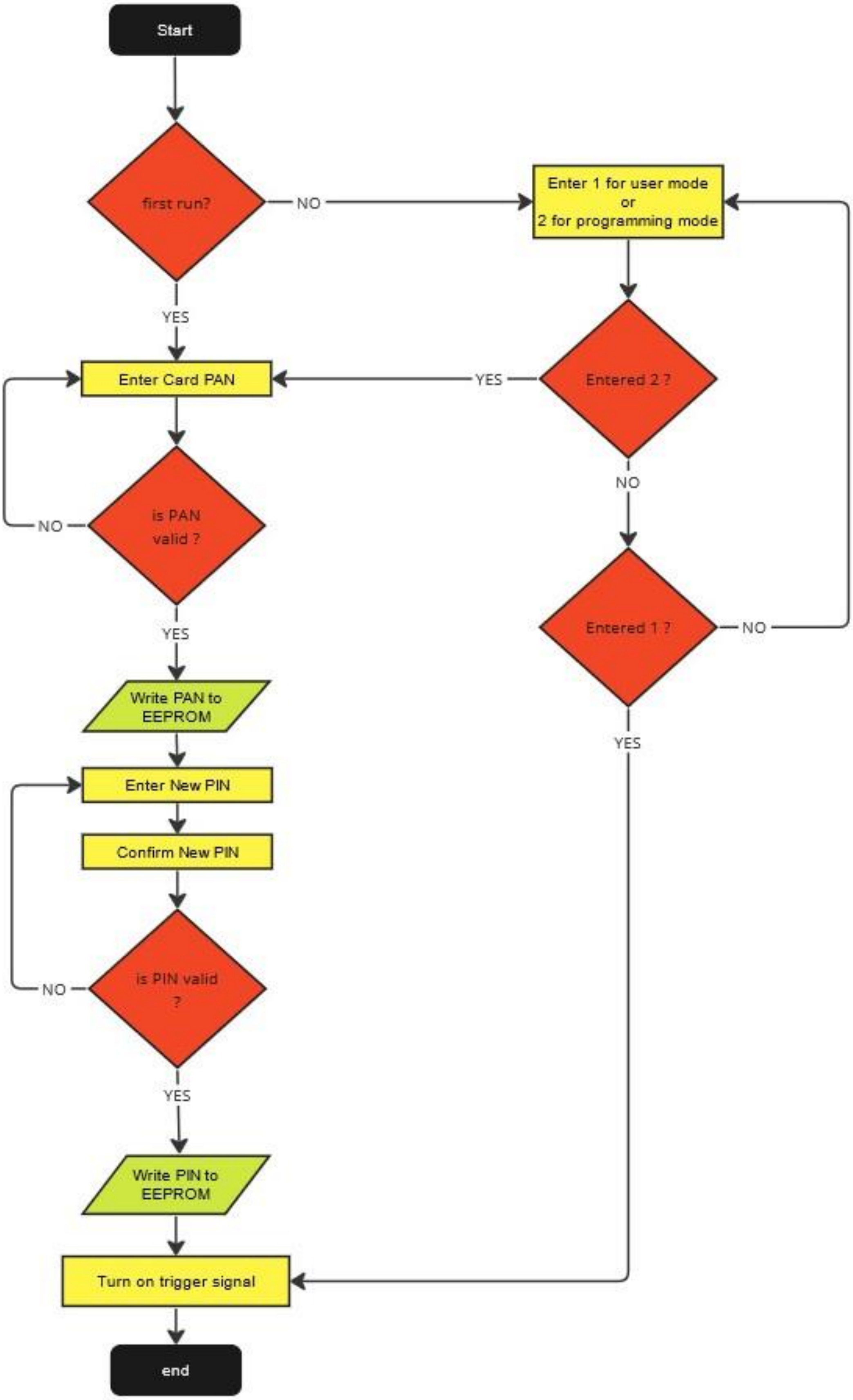

Start
display a welcome message
display insert your card
NO
Trigger signal received ?
YES
Trials = 1
Enter PIN numbers by user
Trials ++
YES
PIN is correct ?
NO
Trials < 3 ?
NO
Turn ON alarm, Lock every input
end
YES
Enter the needed transaction amount
Check database for validation
NO
Execute Account_Checks()
end
YES
display Approved Transaction, display the updated balance
display Ejecting Card
end

CARD APP:

```
void Card_app_Init(void);
```


Start
initializing I2C
initializing UART
initializing SPI as Master
initializing Trigger pin as output and write high to it
End

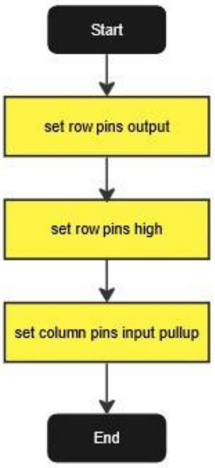
```
void Card_app_Start(void);
```



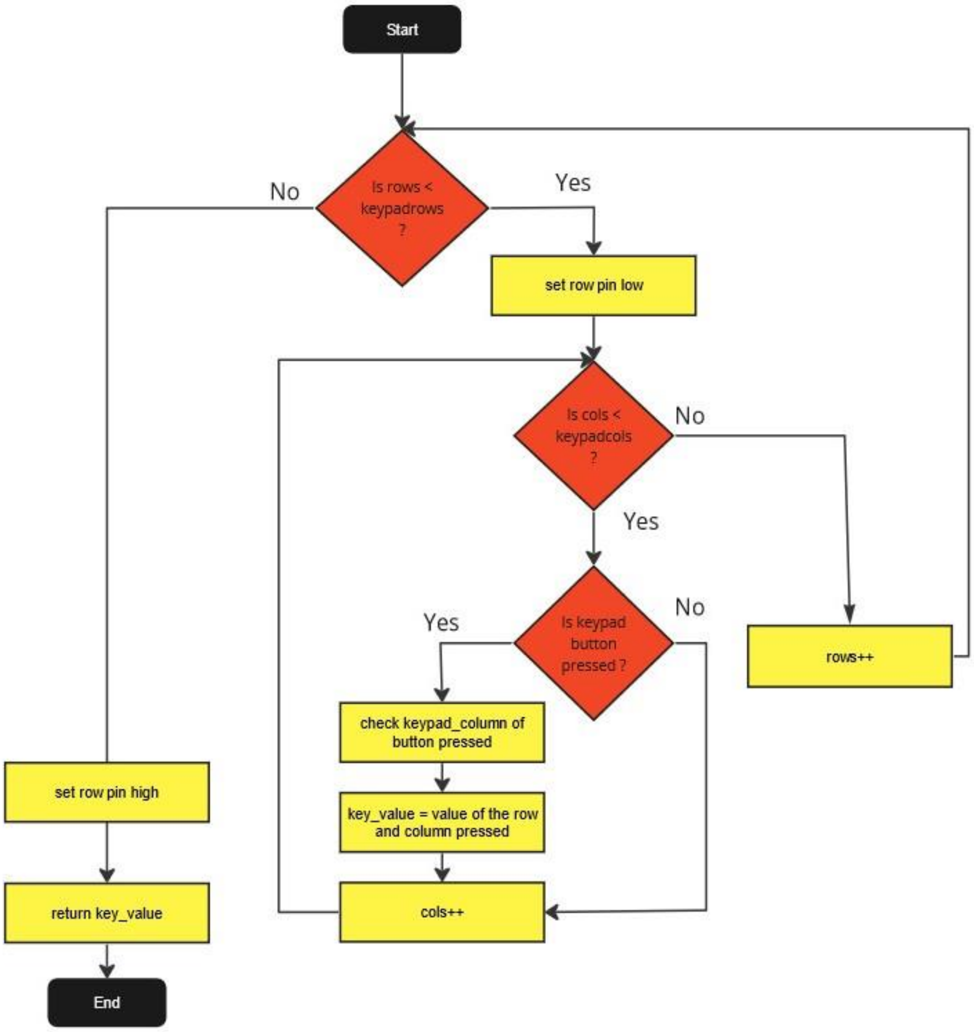
HAL Layer:

Keypad functions:-

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

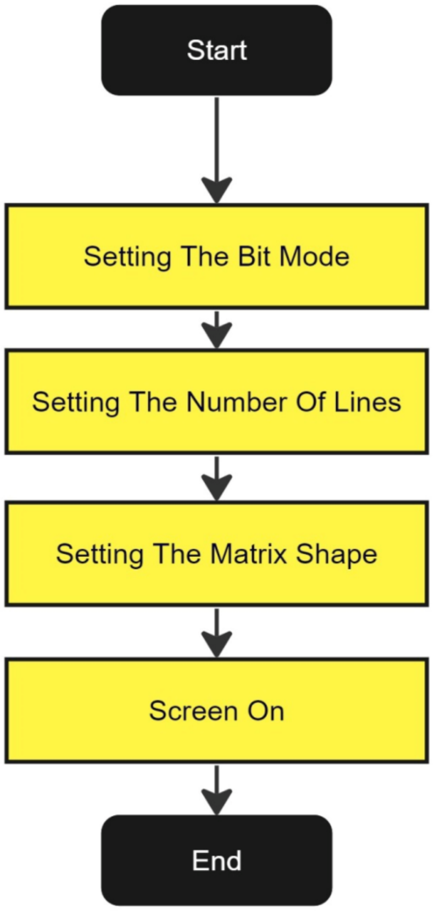


```
uint8_t KEYPAD_getKey(void);
```

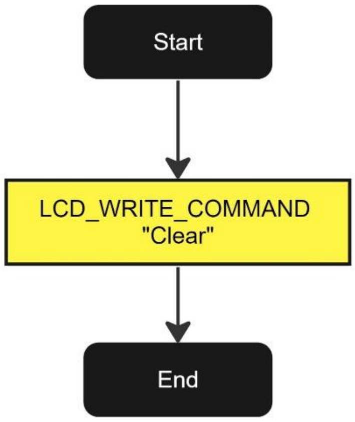


LCD functions:-

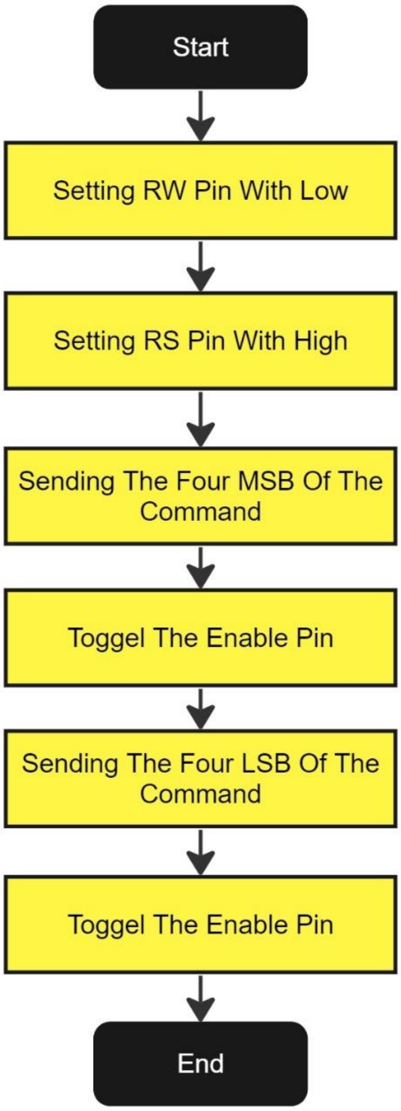
LCD_INIT



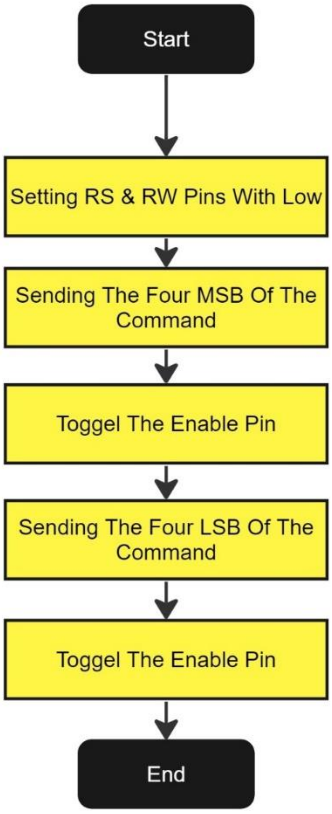
LCD_CLEAR



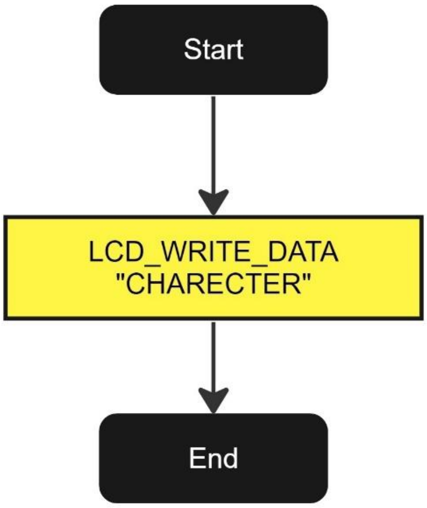
LCD_WRITE_DATA



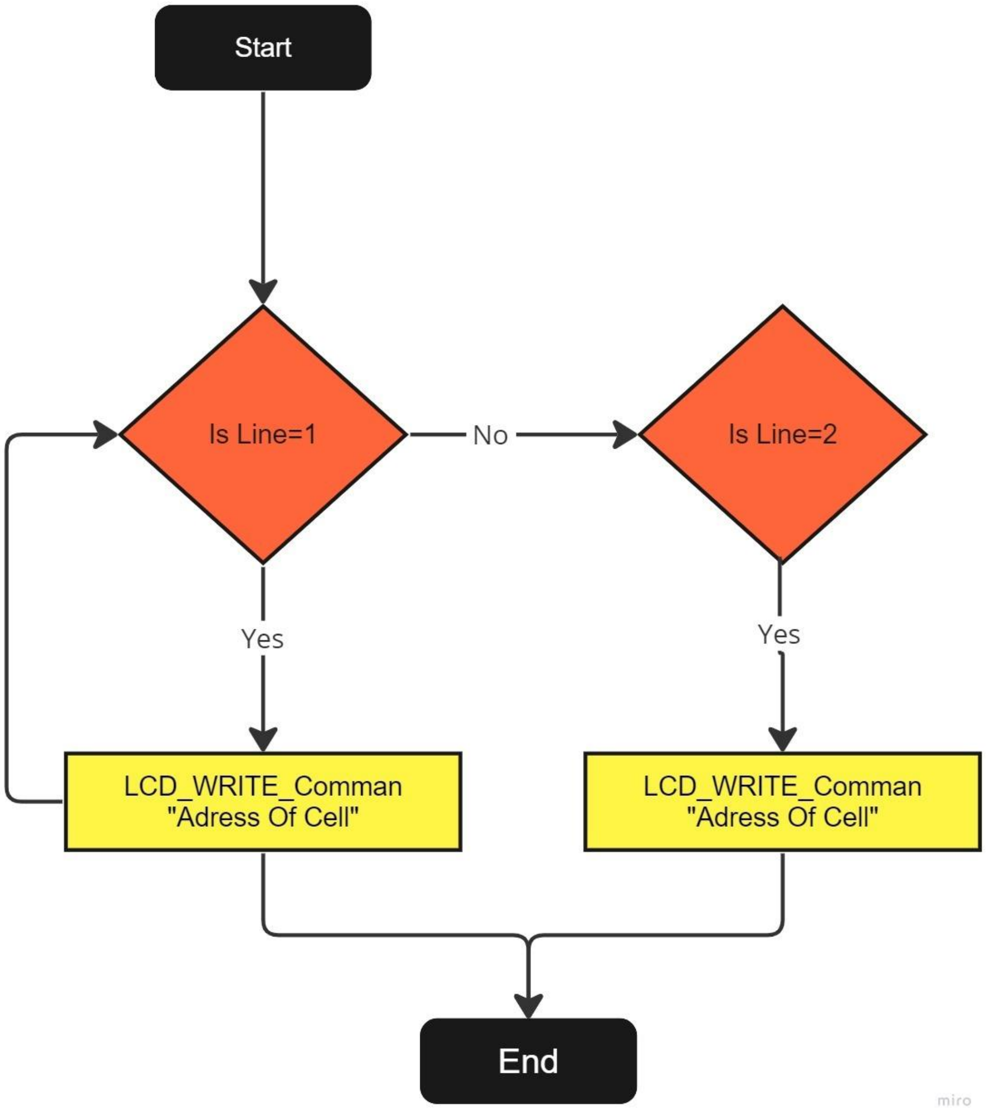
LCD_WRITE_COMMAND


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

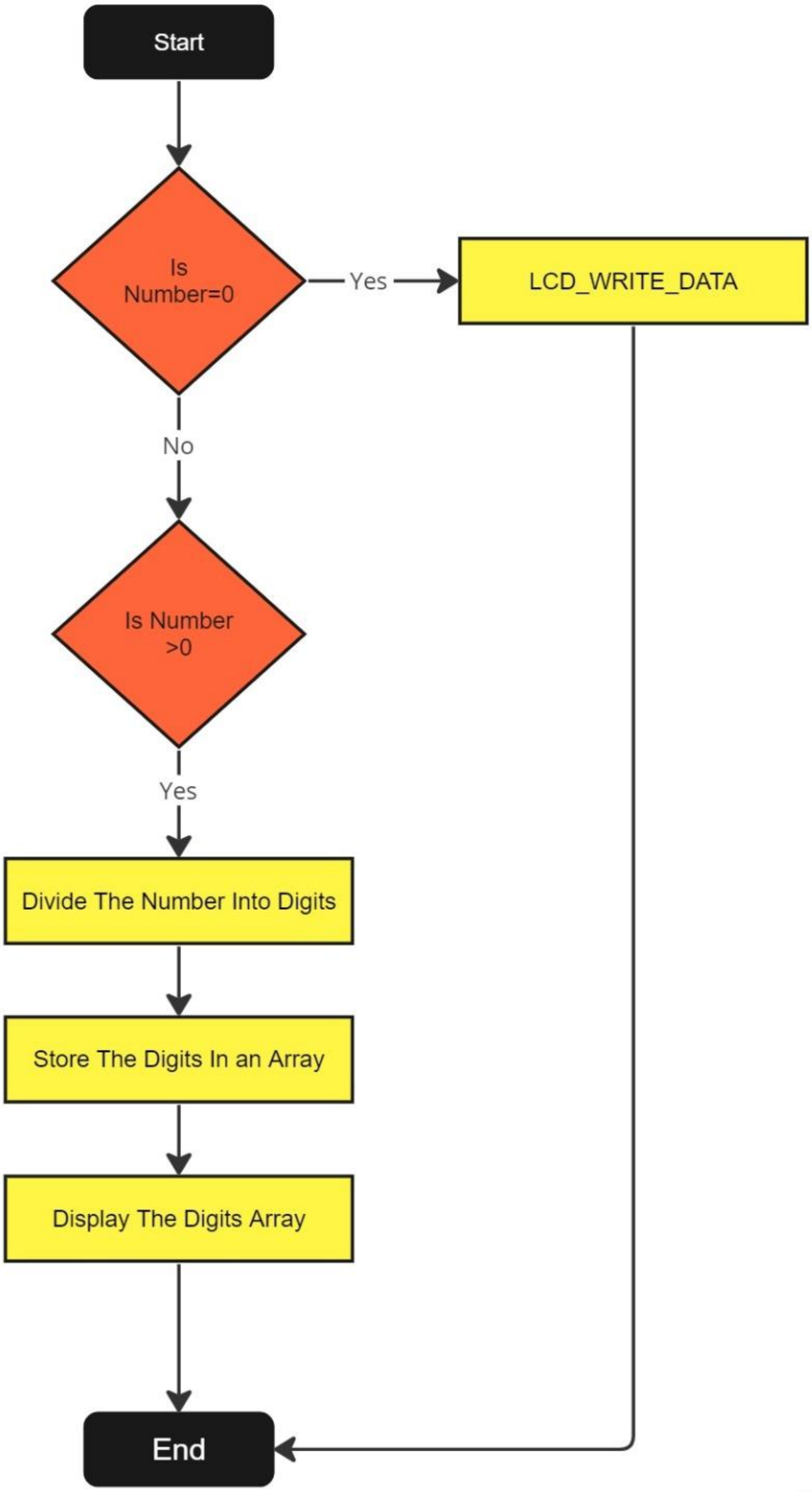
LCD_Write_Charecter



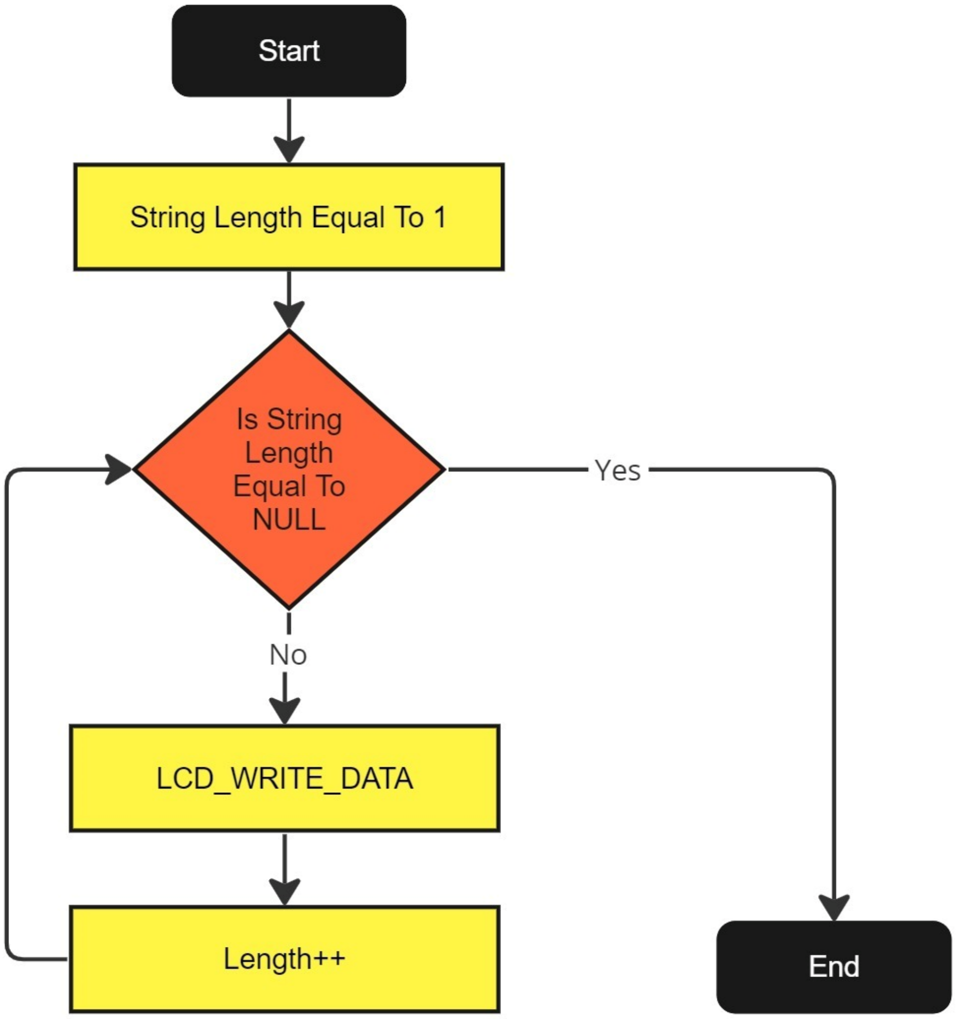
LCD_GoTo


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

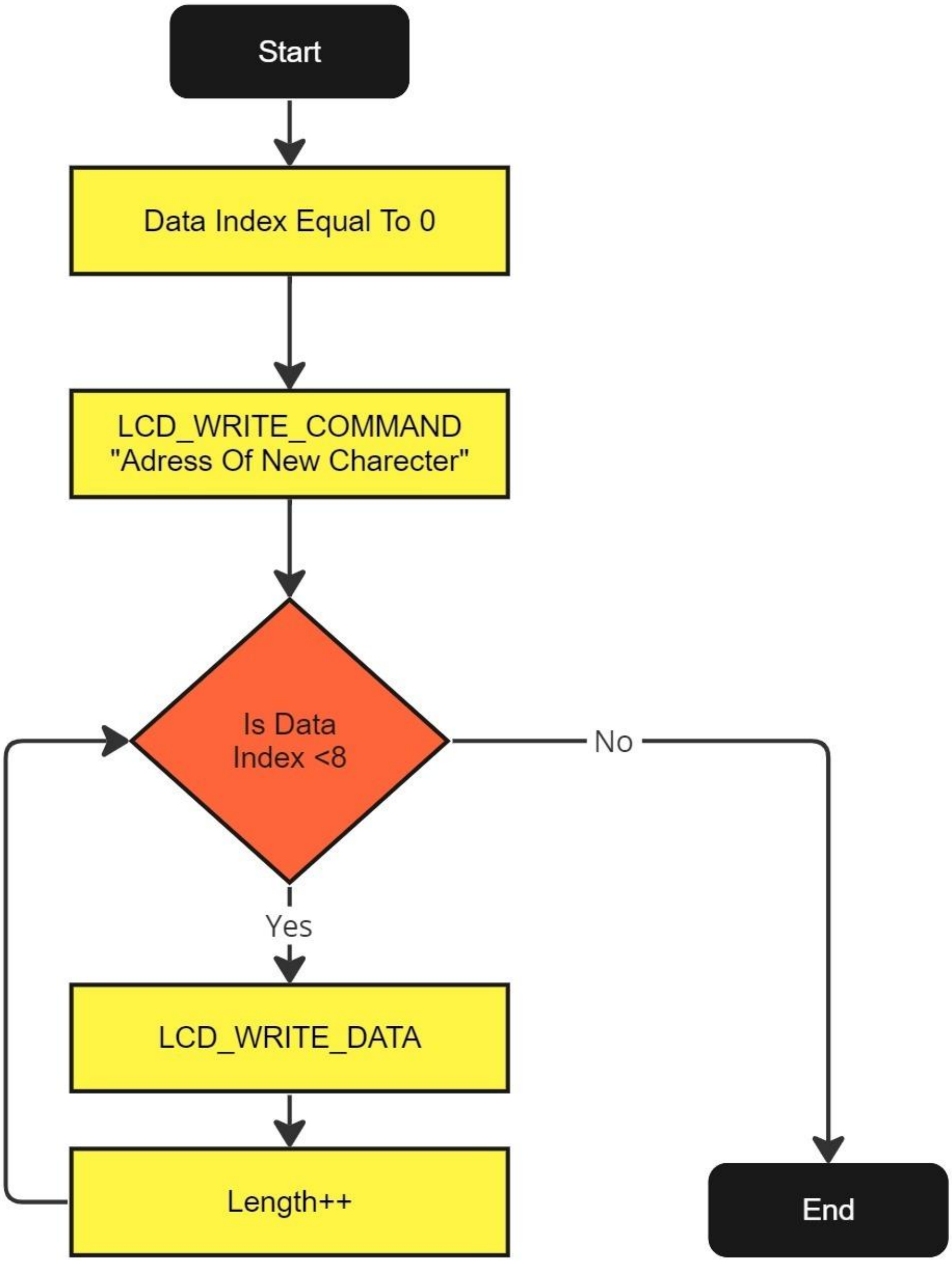
LCD_WRITE_NUMBER



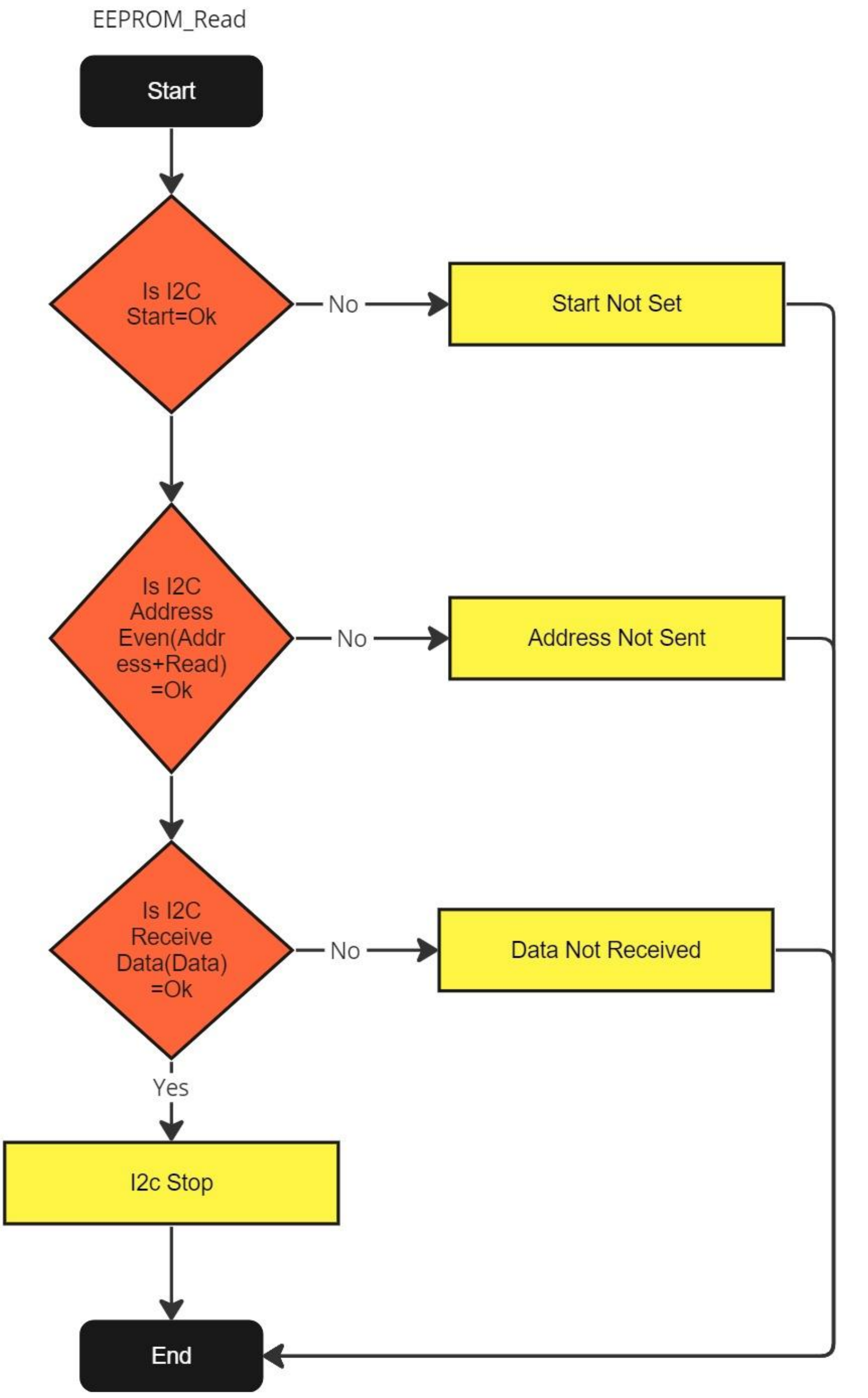
LCD_Write_String

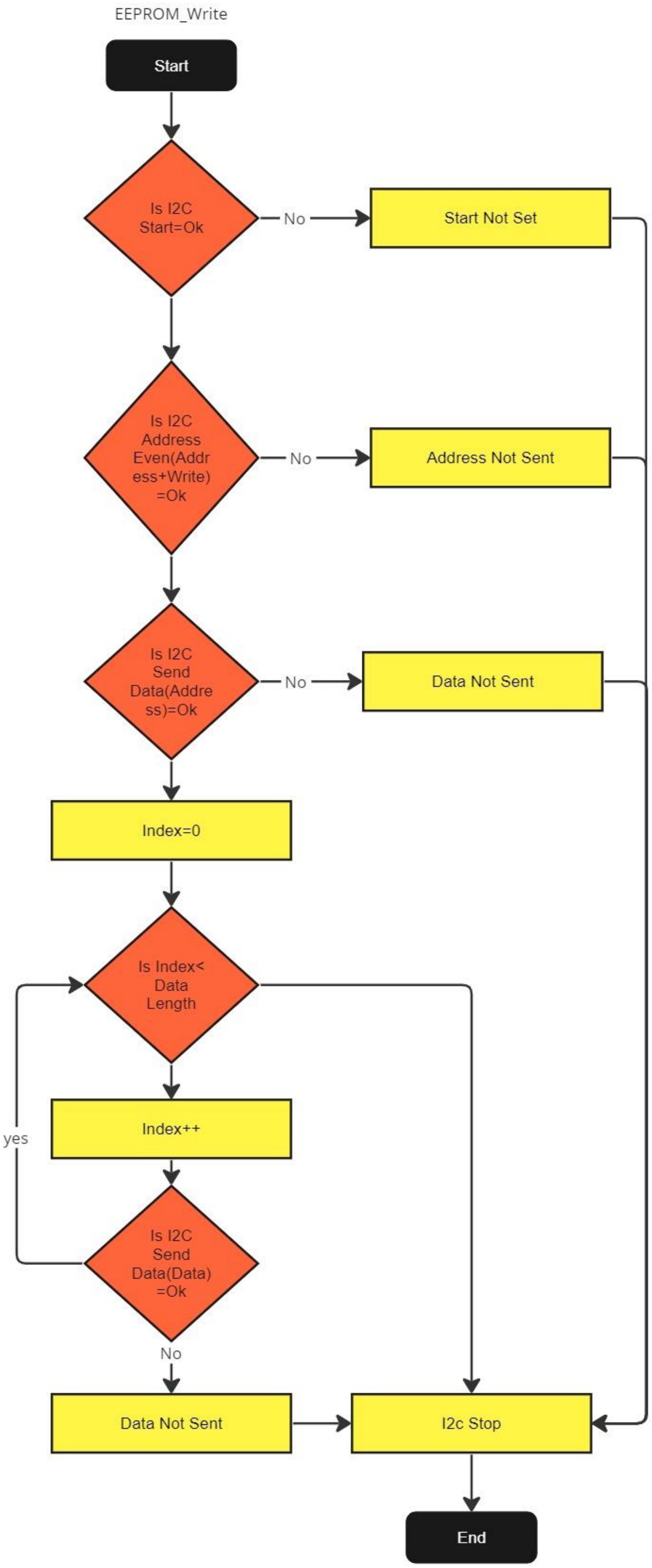


LCD_Create_Charecter

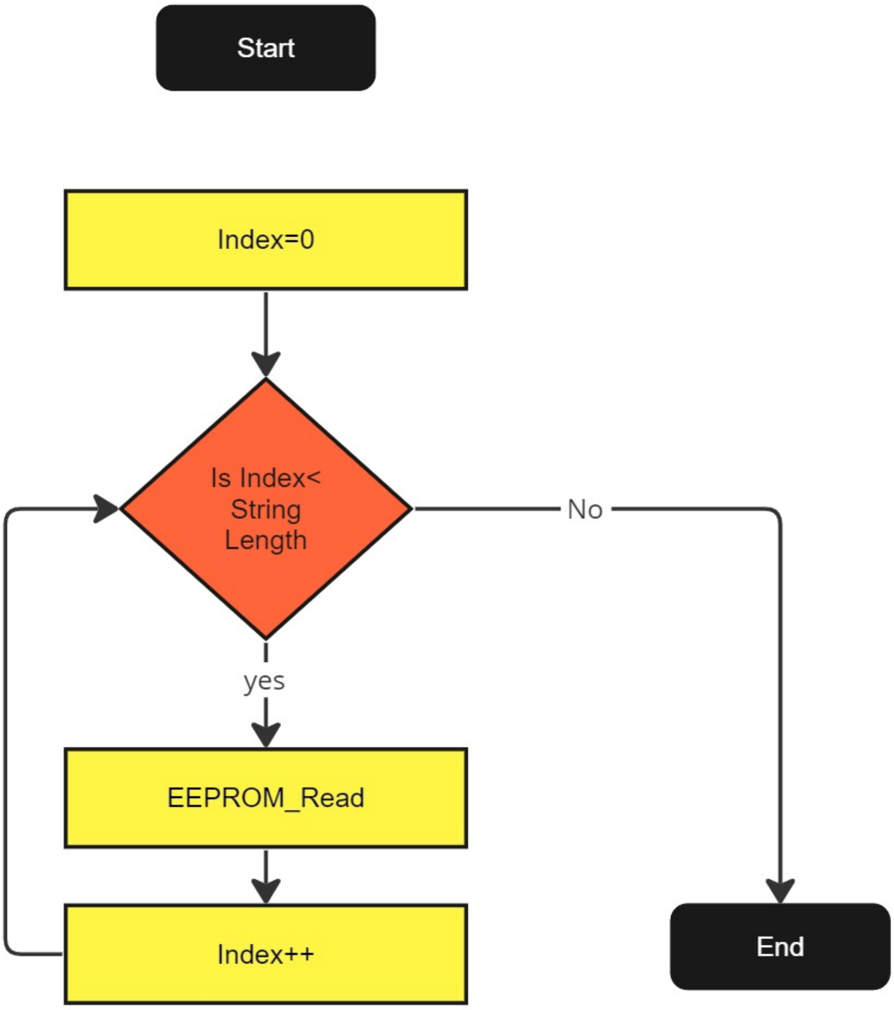


EEPROM functions’ flowcharts:-




EEPROM_Write
Start
Is I2C Start=Ok
No
Start Not Set
Is I2C Address Even(Address+Write) =Ok
No
Address Not Sent
Is I2C Send Data(Address)=Ok
No
Data Not Sent
Index=0
Is Index< Data Length
Index++
yes
Is I2C Send Data(Data) =Ok
No
Data Not Sent
I2c Stop
End

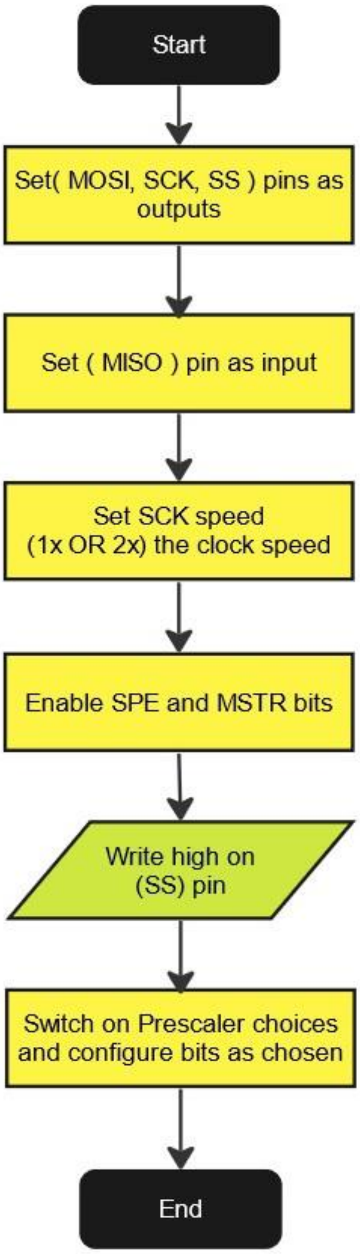
EEPROM_ReadString



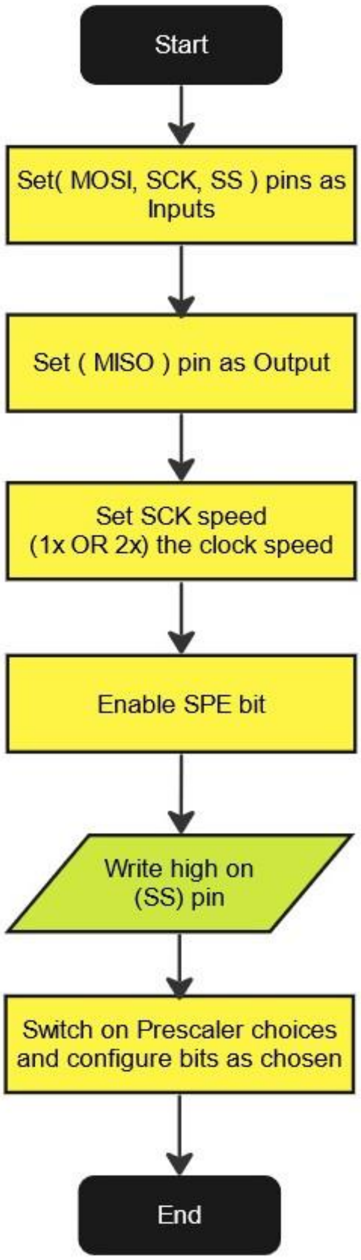
MCAL Layer:-

SPI functions’ flowcharts :-

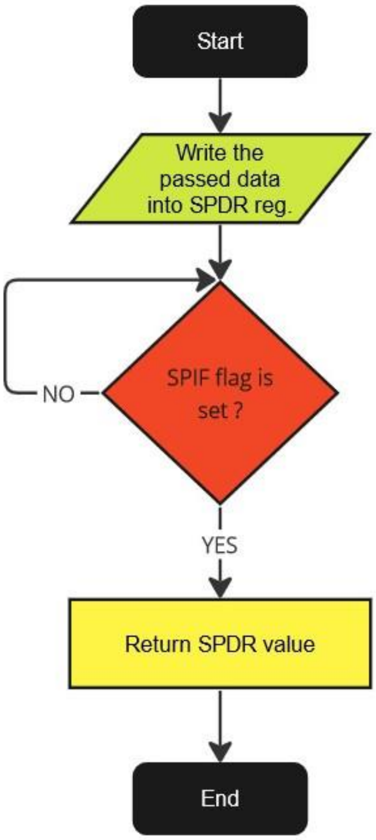
void SPI_MasterInit(en_SPI_Prescaler prescaler)


Start
Set(MOSI, SCK, SS) pins as outputs
Set (MISO) pin as input
Set SCK speed
(1x OR 2x) the clock speed
Enable SPE and MSTR bits
Write high on
(SS) pin
Switch on Prescaler choices and configure bits as chosen
End

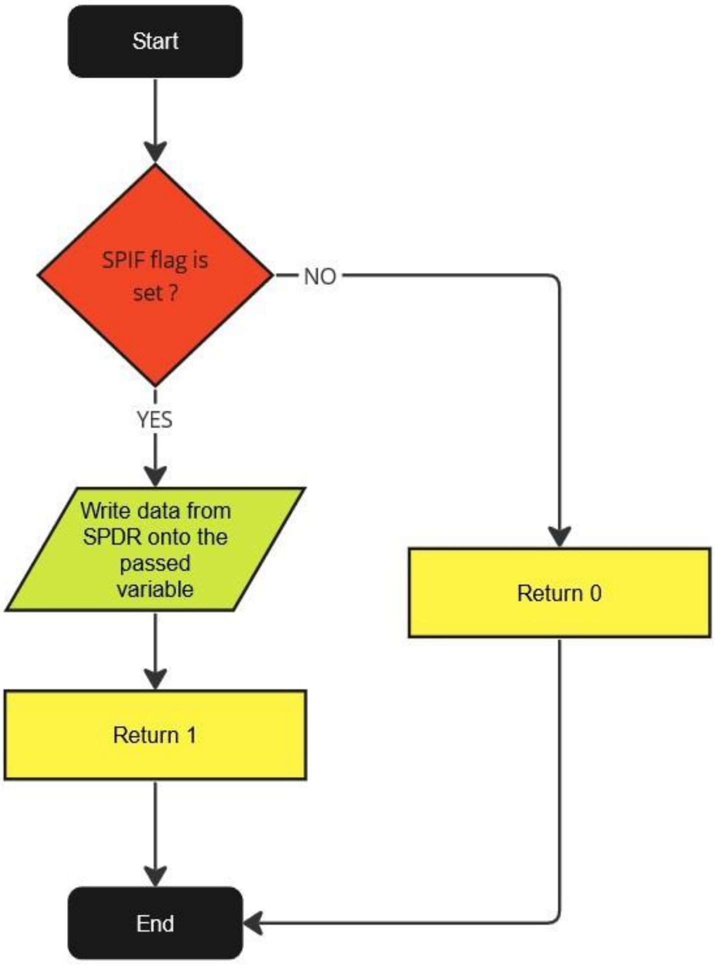
void SPI_SlaveInit(en_SPI_Prescaler prescaler)


Start
Set(MOSI, SCK, SS) pins as Inputs
Set (MISO) pin as Output
Set SCK speed
(1x OR 2x) the clock speed
Enable SPE bit
Write high on
(SS) pin
Switch on Prescaler choices and configure bits as chosen
End

uint8_t SPI_SendReceive(uint8_t data)

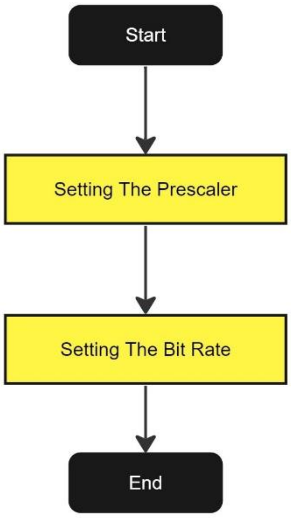


uint8_t SPI_RecievePeriodicChecking(uint8_t * pdata)

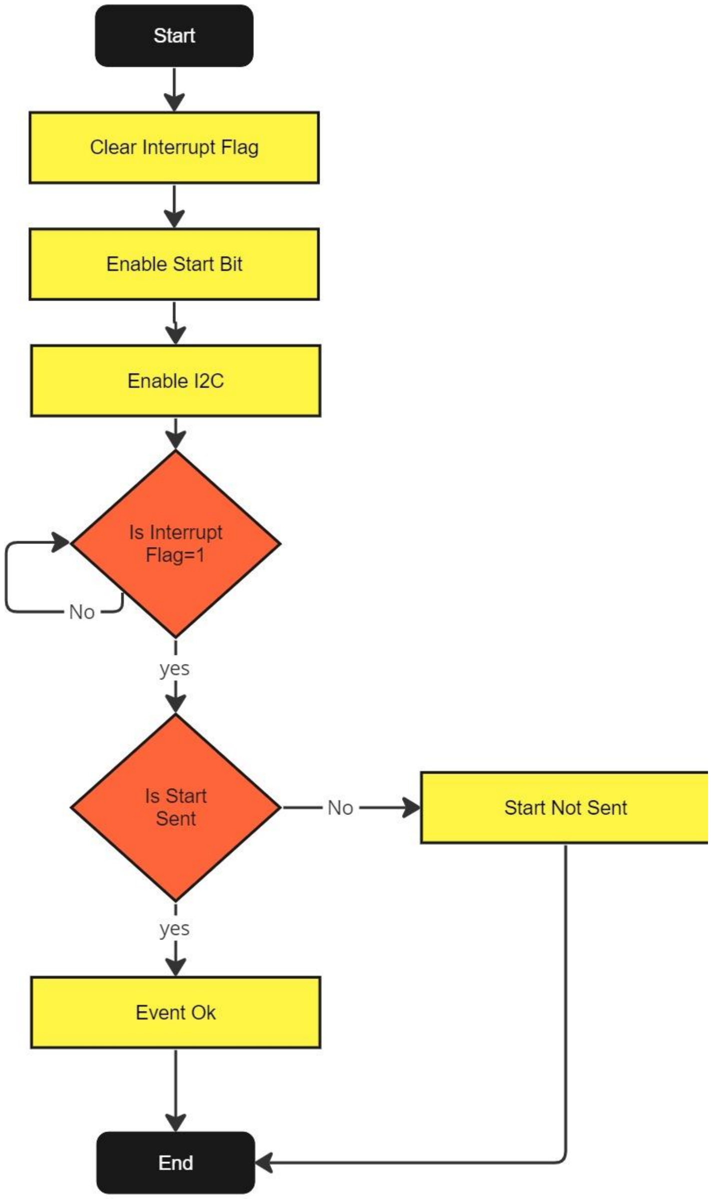


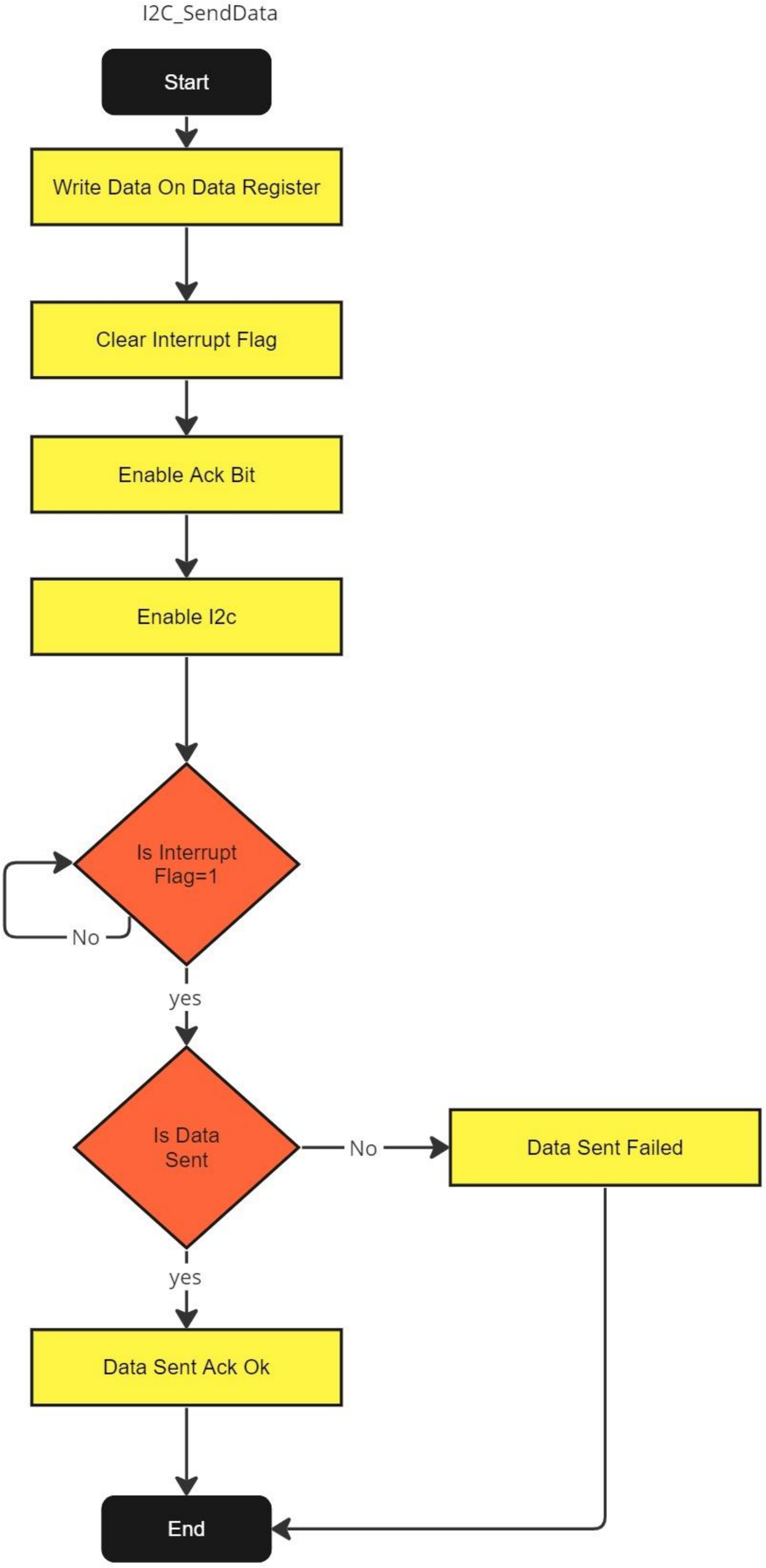
I2C functions’ flowcharts:-

I2C_INIT

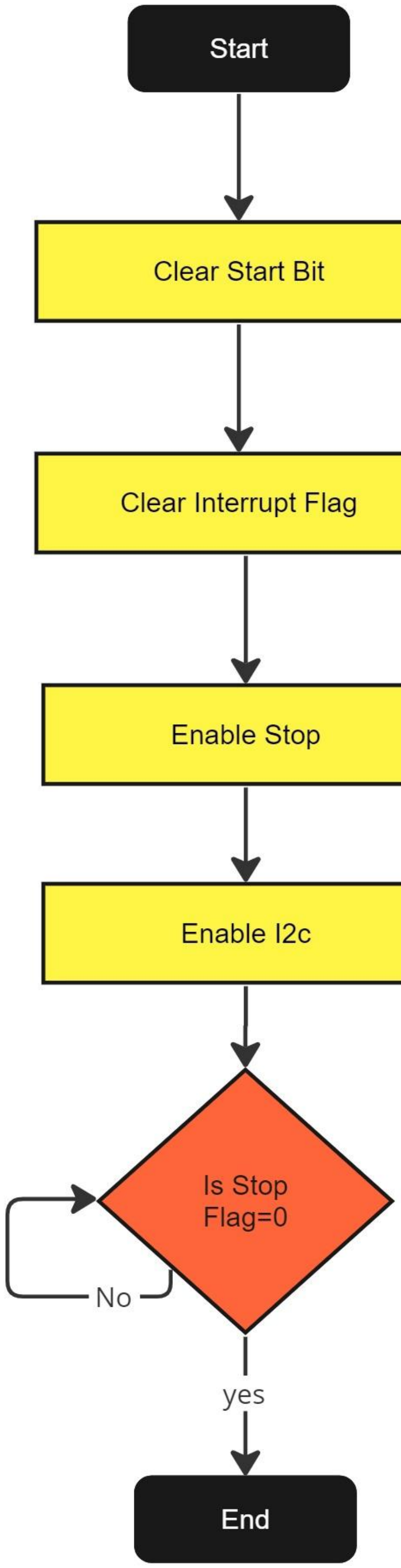


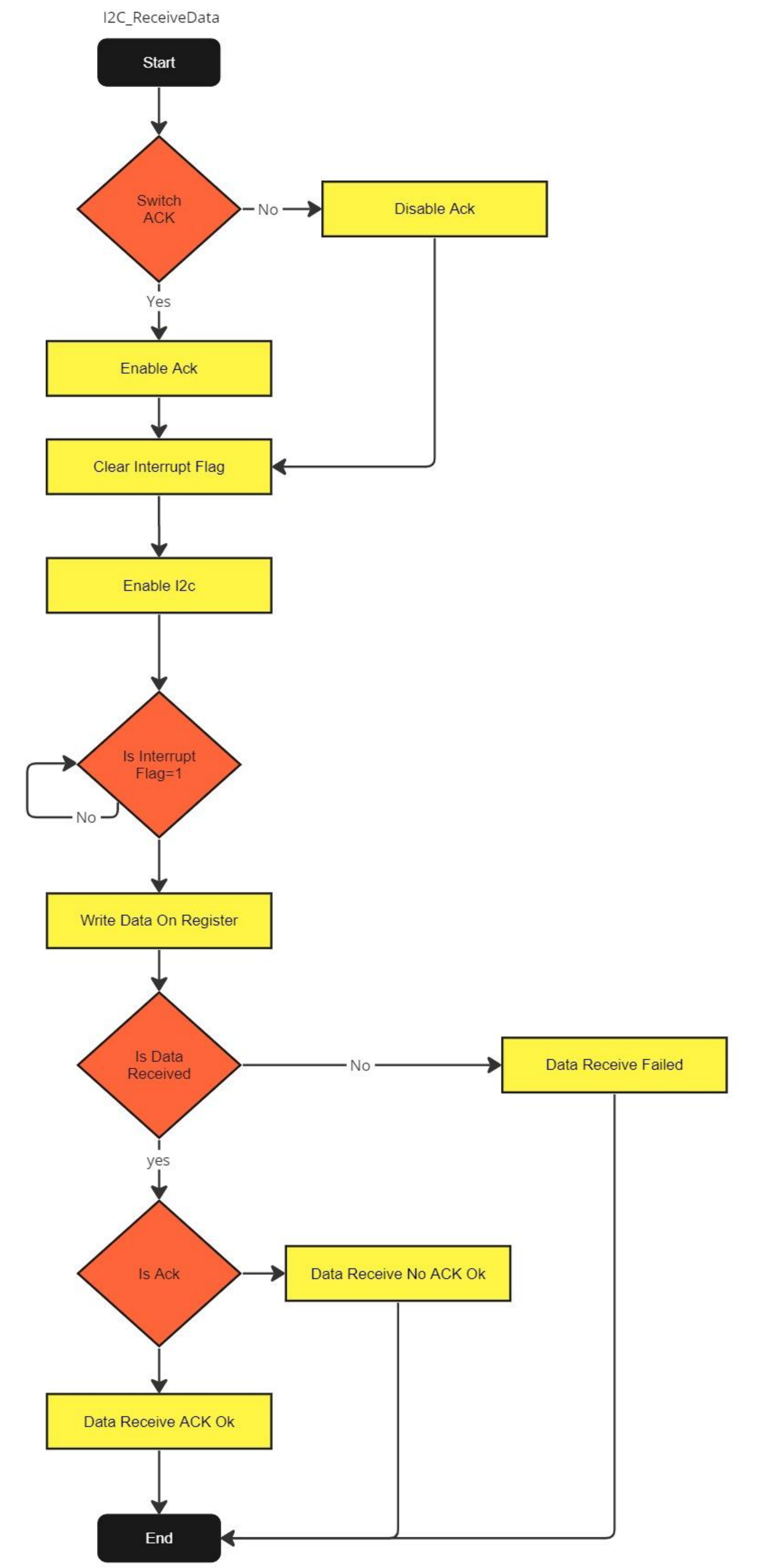
I2C_Start



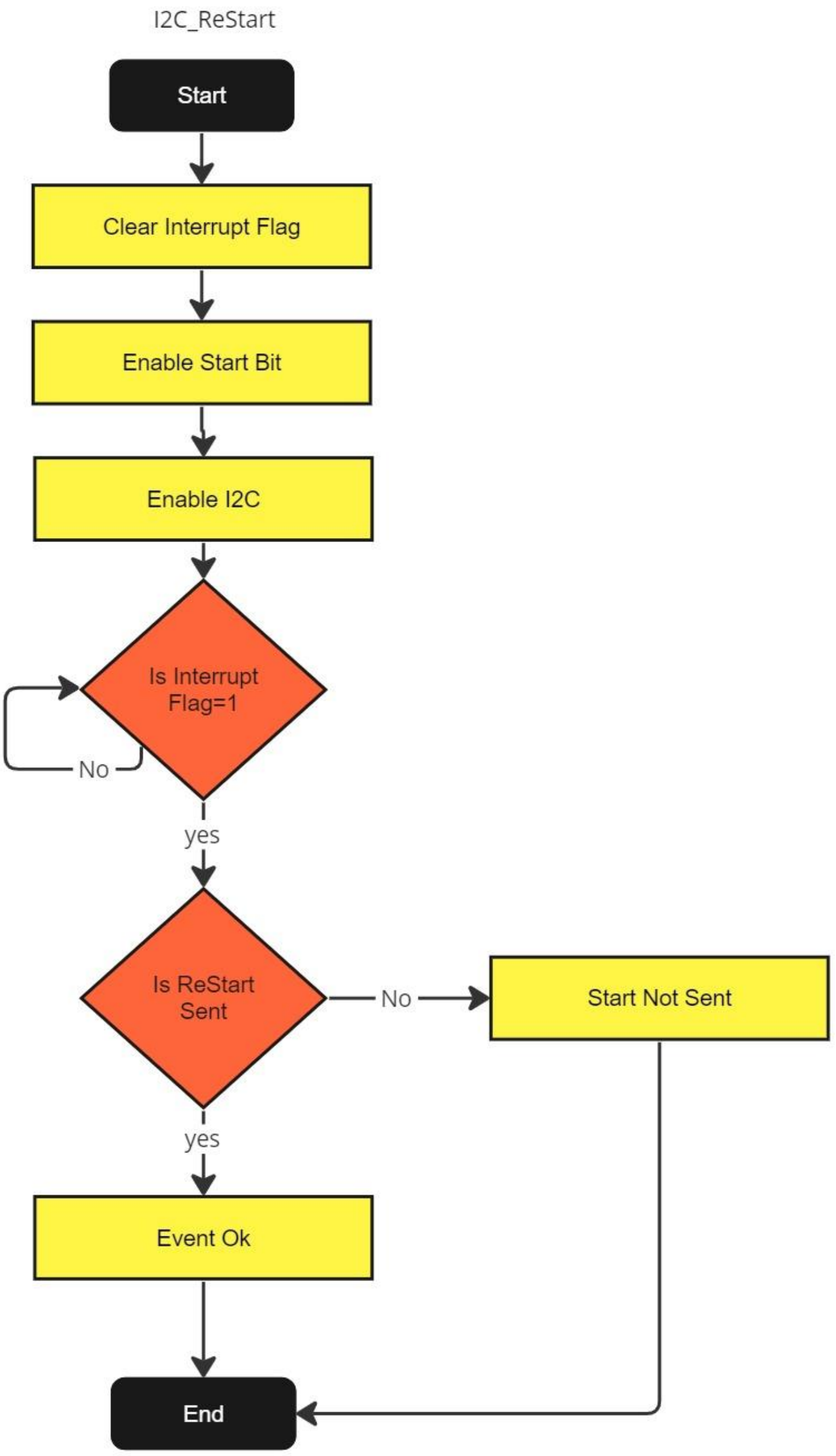

I2C_SendData
Start
Write Data On Data Register
Clear Interrupt Flag
Enable Ack Bit
Enable I2c
Is Interrupt Flag=1
No
yes
Is Data Sent
No
Data Sent Failed
yes
Data Sent Ack Ok
End

I2C_Stop


Start
Clear Start Bit
Clear Interrupt Flag
Enable Stop
Enable I2c
Is Stop Flag=0
No
yes
End

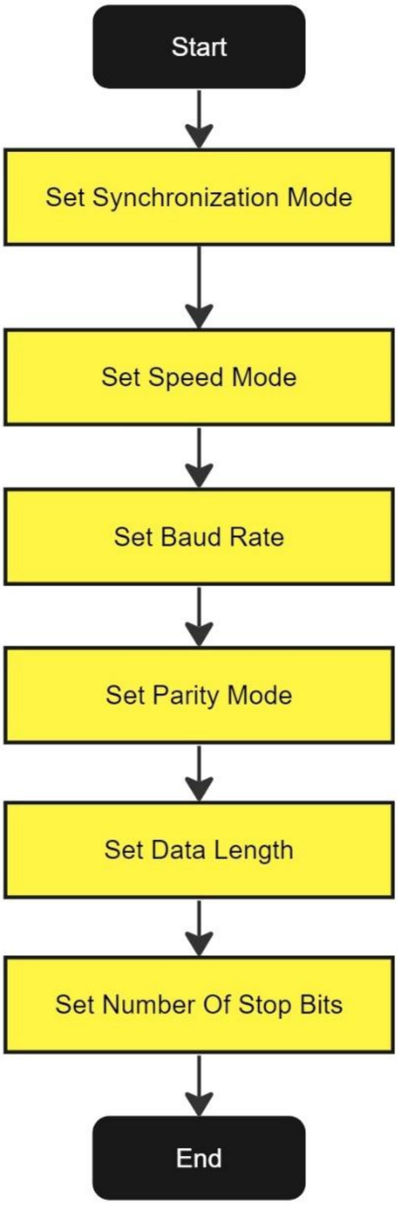

I2C_ReceiveData
Start
Switch ACK
No
Disable Ack
Yes
Enable Ack
Clear Interrupt Flag
Enable I2c
Is Interrupt Flag=1
No
Write Data On Register
Is Data Received
No
Data Receive Failed
yes
Is Ack
Data Receive No ACK Ok
Data Receive ACK Ok
End

miro

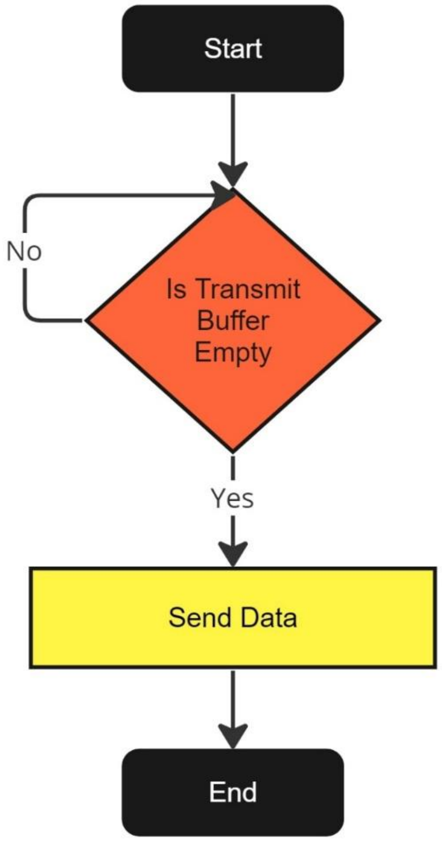

I2C_ReStart
Start
Clear Interrupt Flag
Enable Start Bit
Enable I2C
Is Interrupt Flag=1
No
yes
Is ReStart Sent
No
Start Not Sent
yes
Event Ok
End

USART functions' flowcharts:-

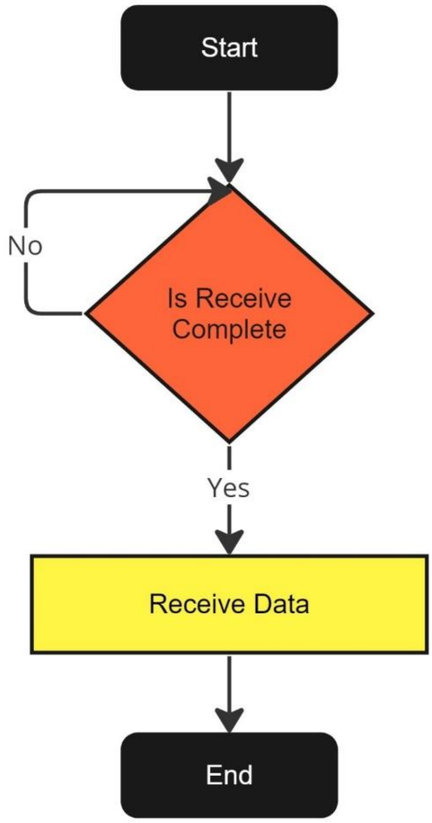
USART_Init



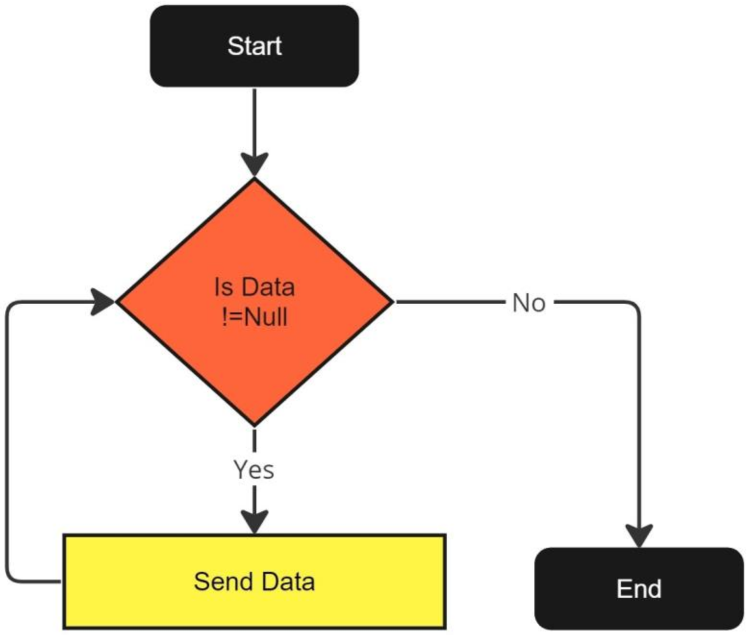
USART_Transmit



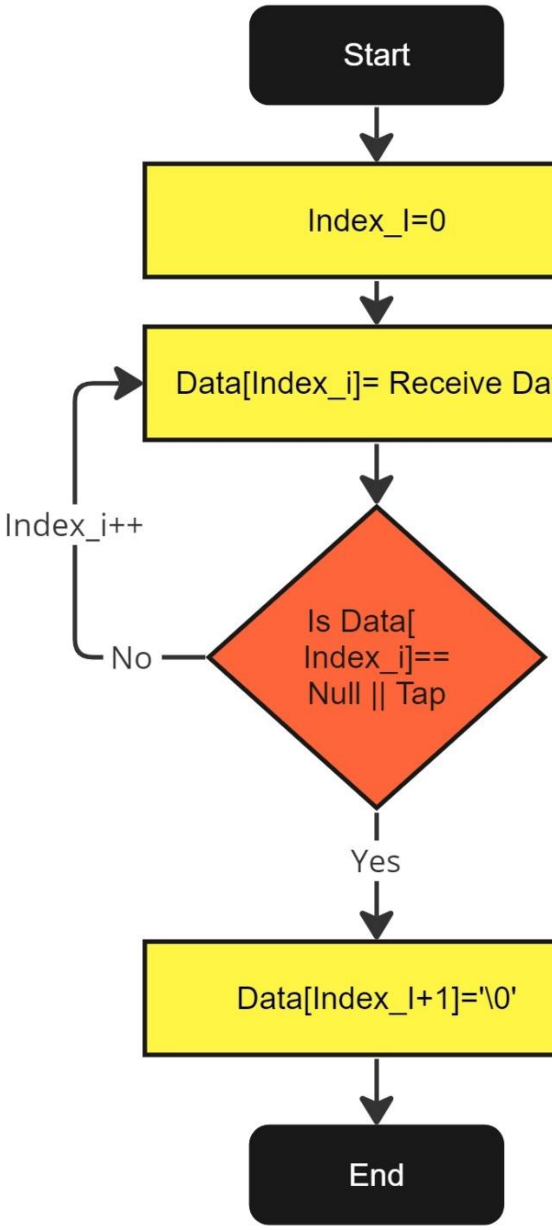
USART_Receive



USART_Transmit_String

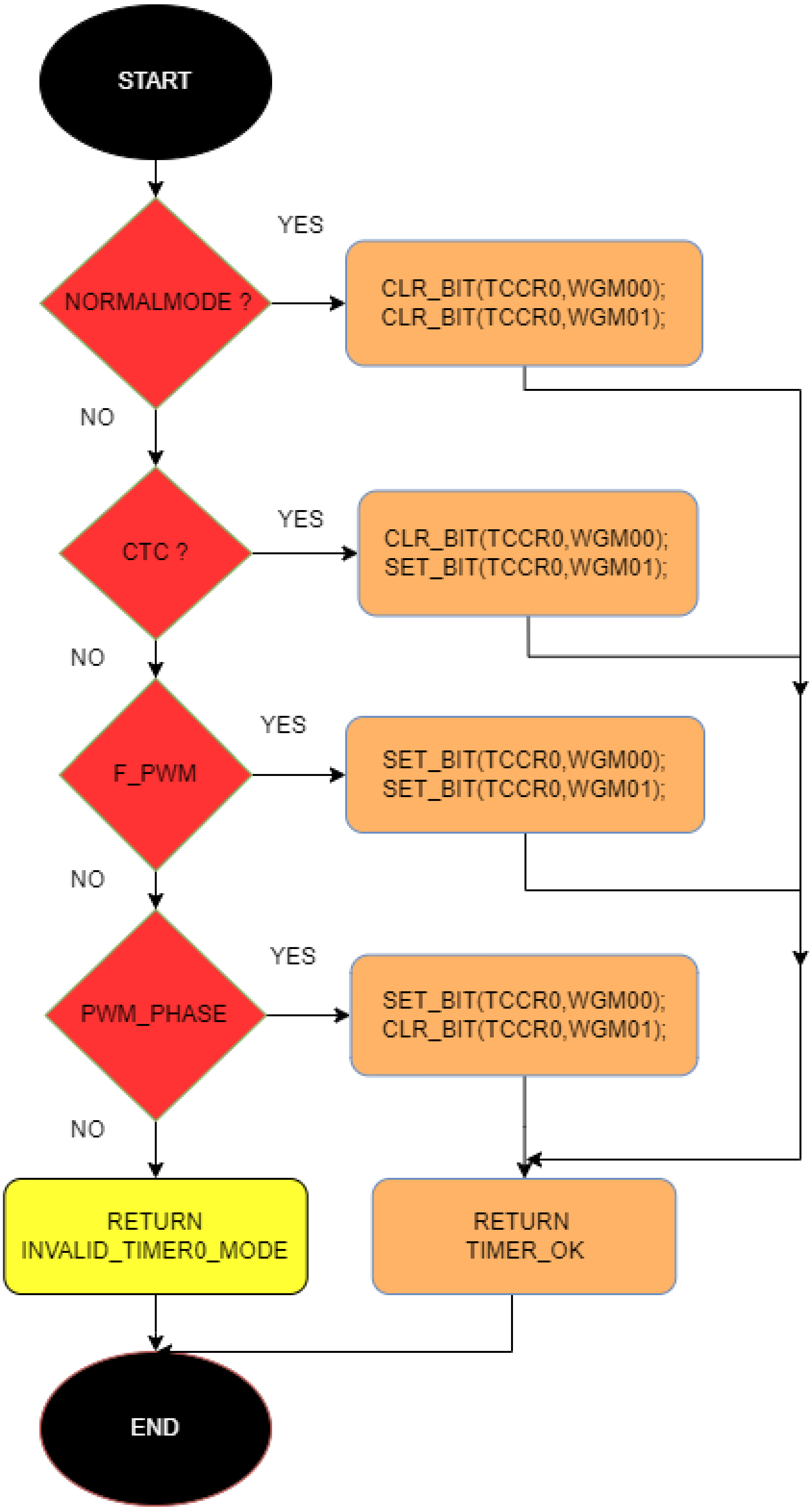


USART_Receive_String

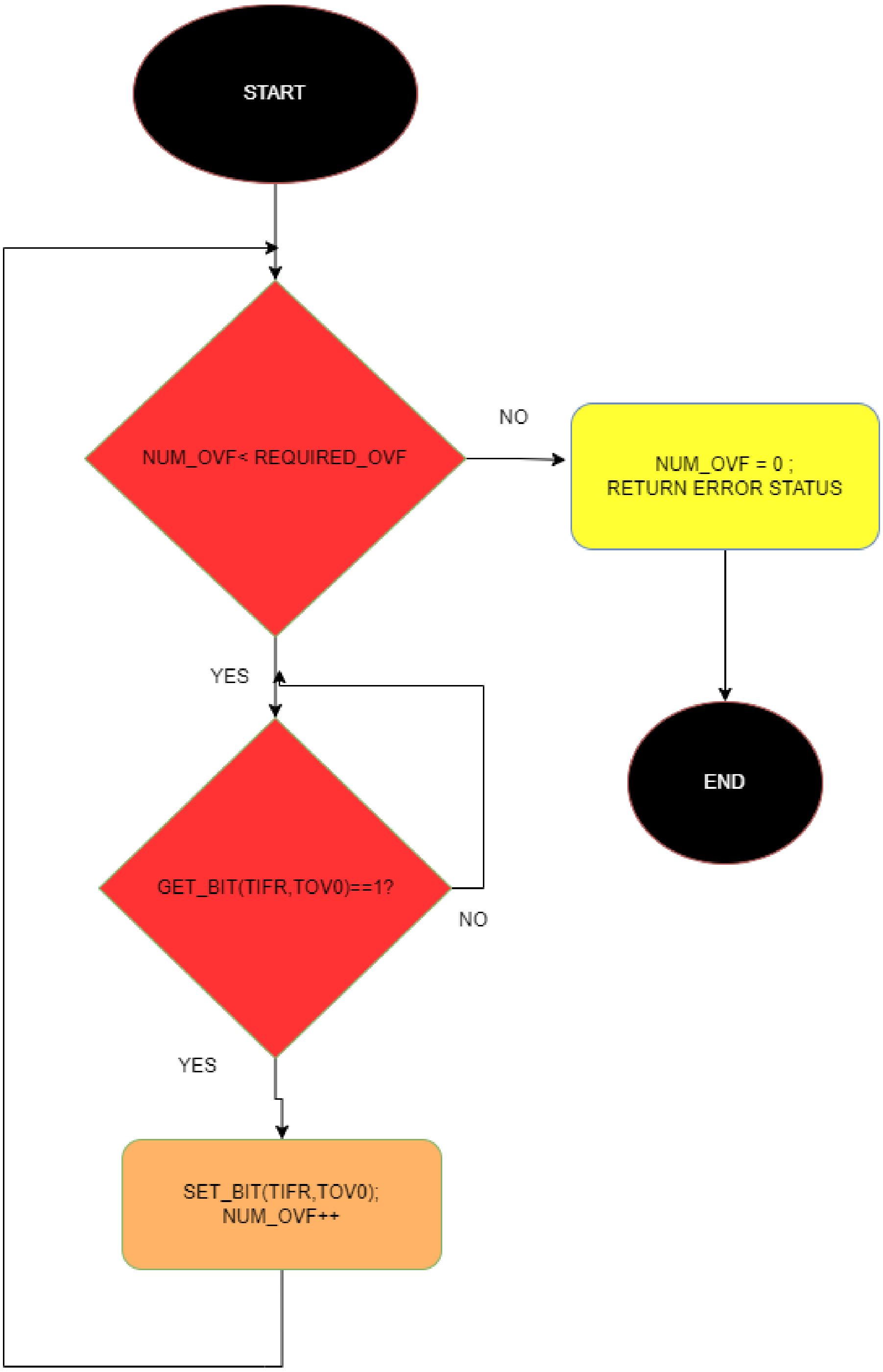


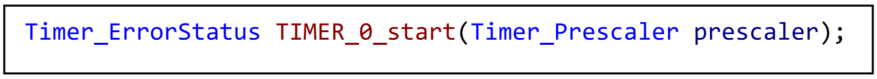
Timer functions’ flowcharts:-

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

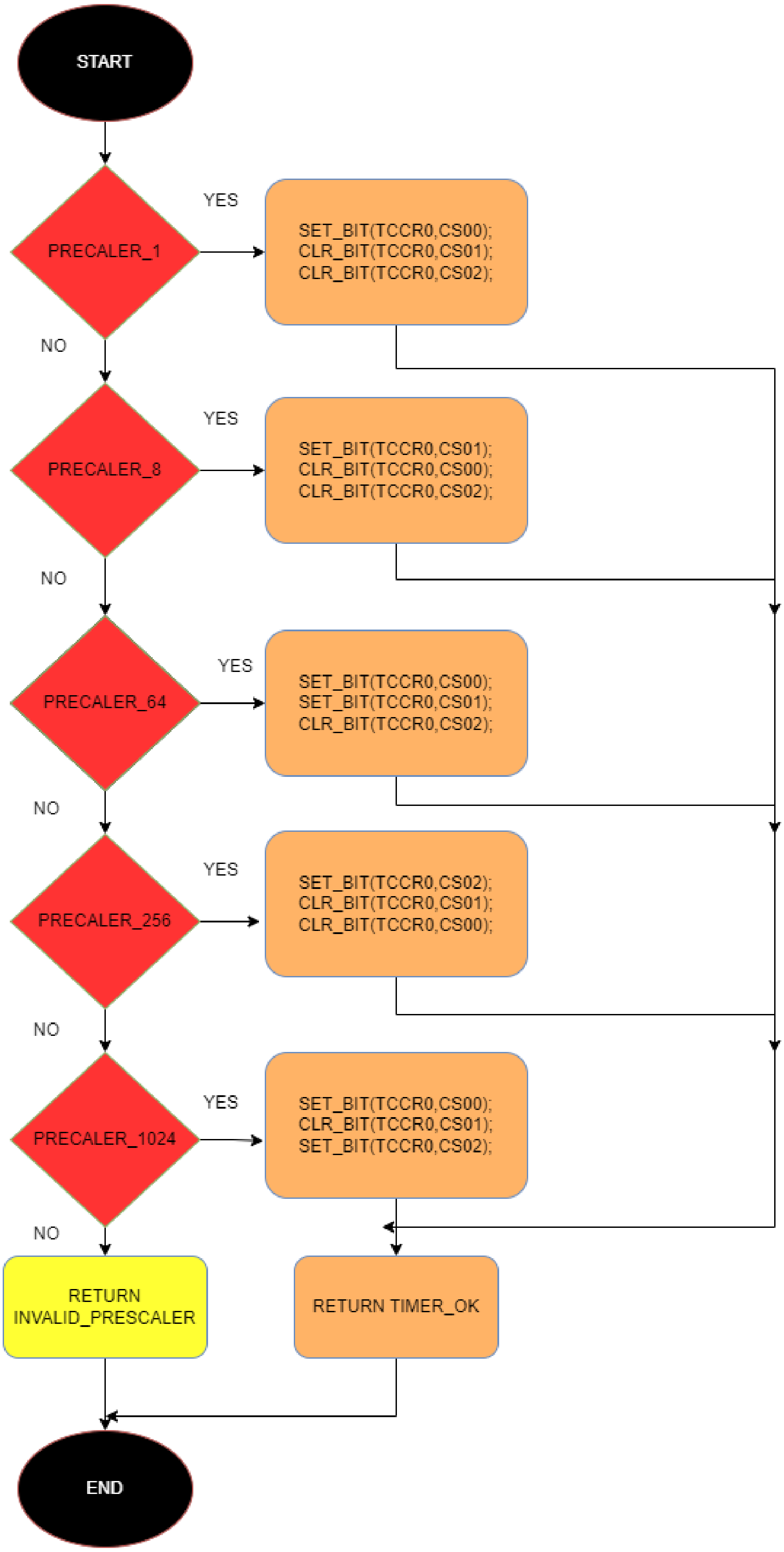


```
Timer_ErrorStatus TIMER_0_OvfNum(double overflow);
```

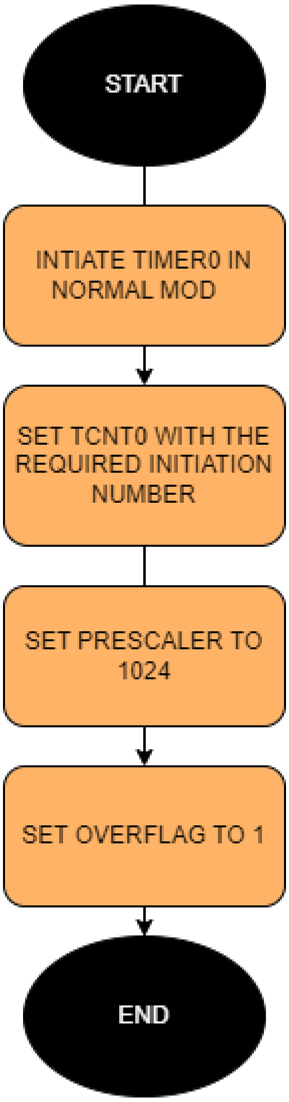




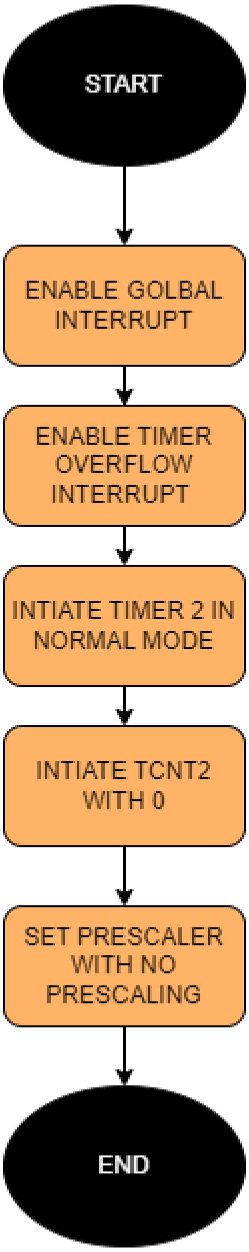
```
Timer_ErrorStatus TIMER_0_start(Timer_Prescaler prescaler);
```



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

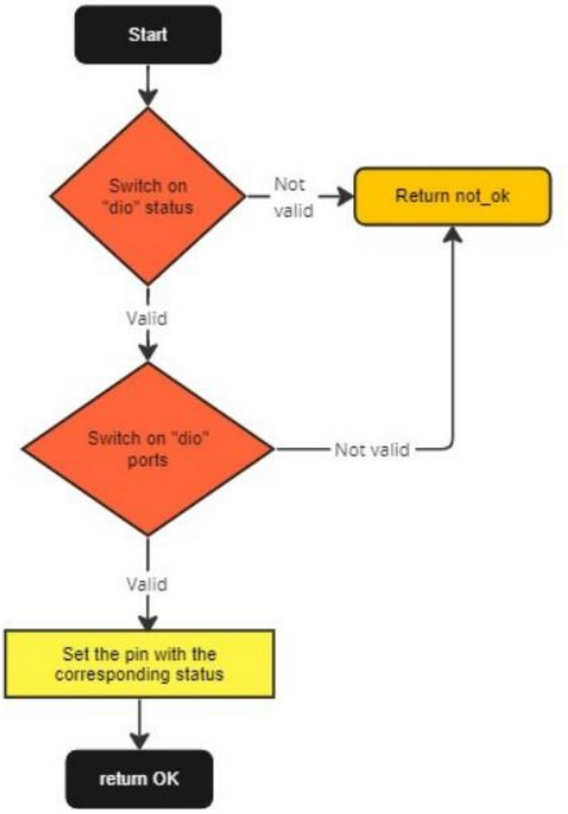

START
INTIATE TIMER0 IN NORMAL MOD
SET TCNT0 WITH THE REQUIRED INITIATION NUMBER
SET PRESCALER TO 1024
SET OVERFLAG TO 1
END

TIMER 2 WITH INTERRUPT

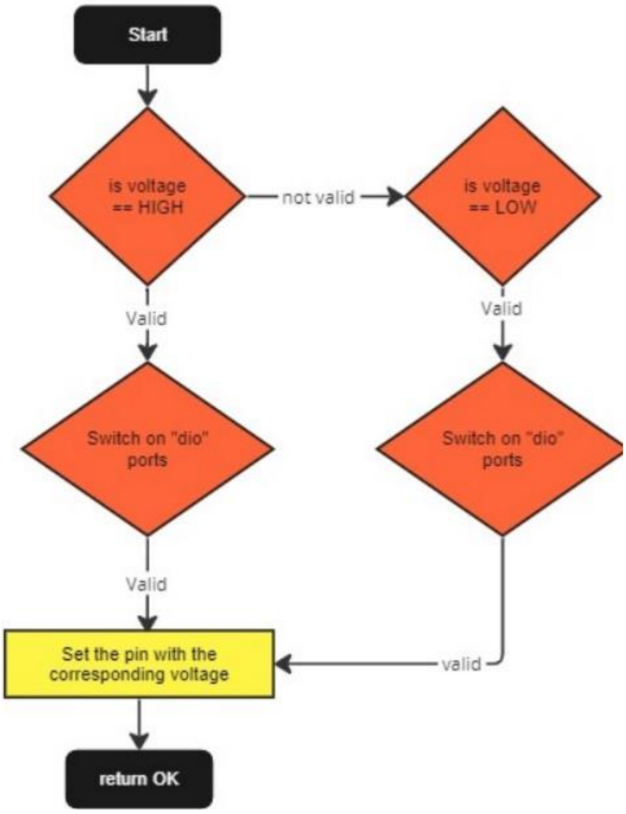

START
ENABLE GOLBAL INTERRUPT
ENABLE TIMER OVERFLOW INTERRUPT
INTIATE TIMER 2 IN NORMAL MODE
INTIATE TCNT2 WITH 0
SET PRESCALER WITH NO PRESCALING
END

DIO functions flowcharts:-

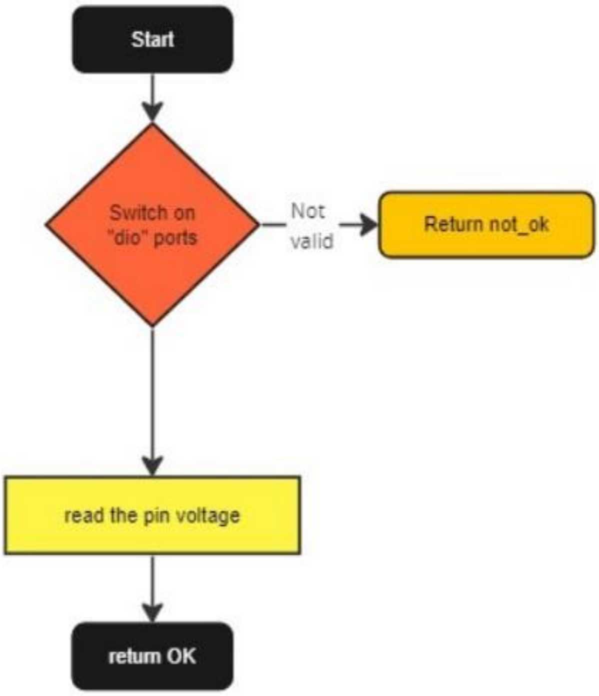
DIO_INITPIN



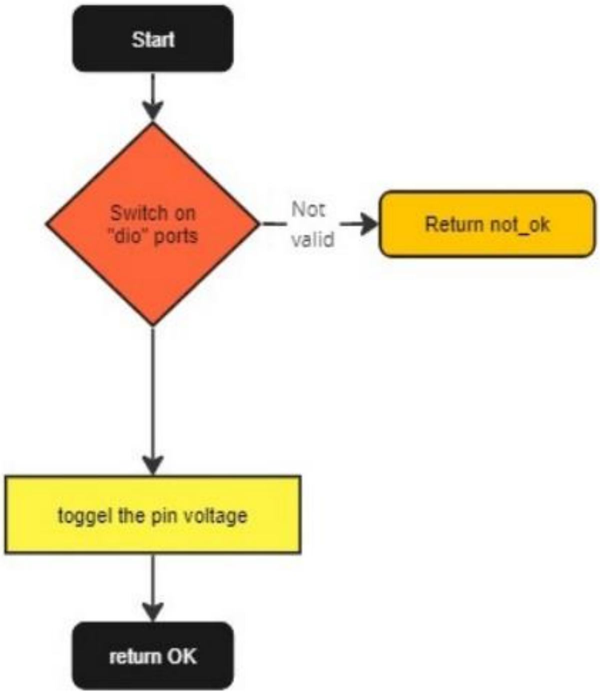
DIO_WRITEPIN



DIO_READPIN

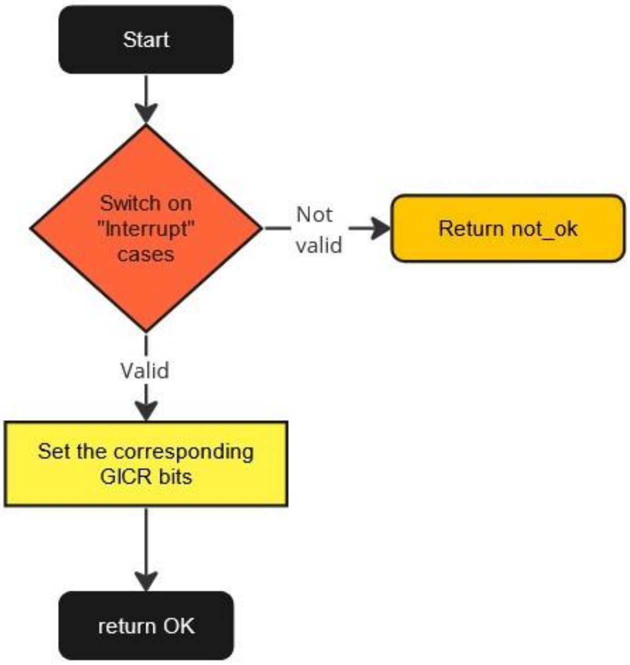


DIO_TogglePin

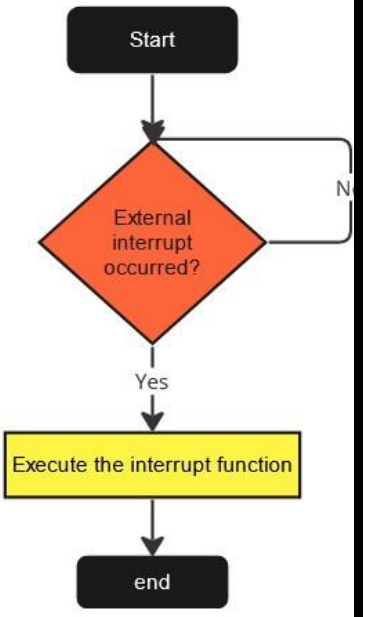


Interrupt functions’ flowcharts

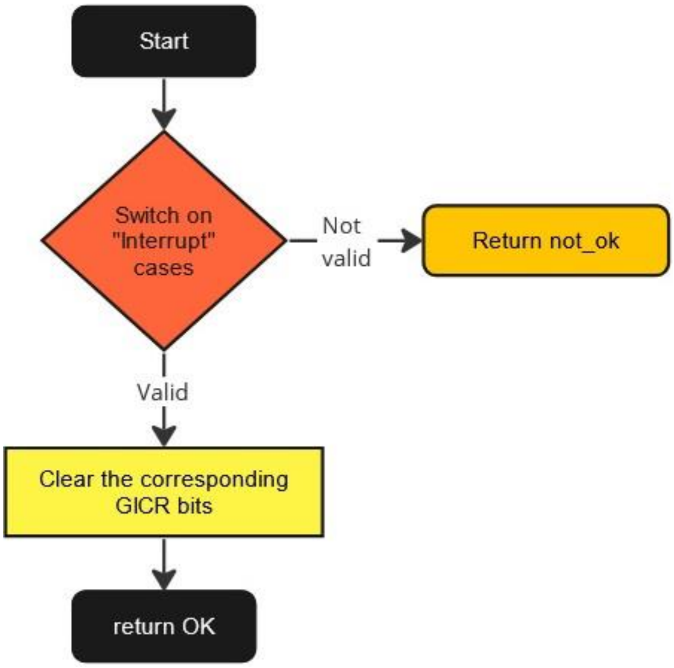
```
EN_int__error_t EXI_Enable (EN_int_t Interrupt)
```



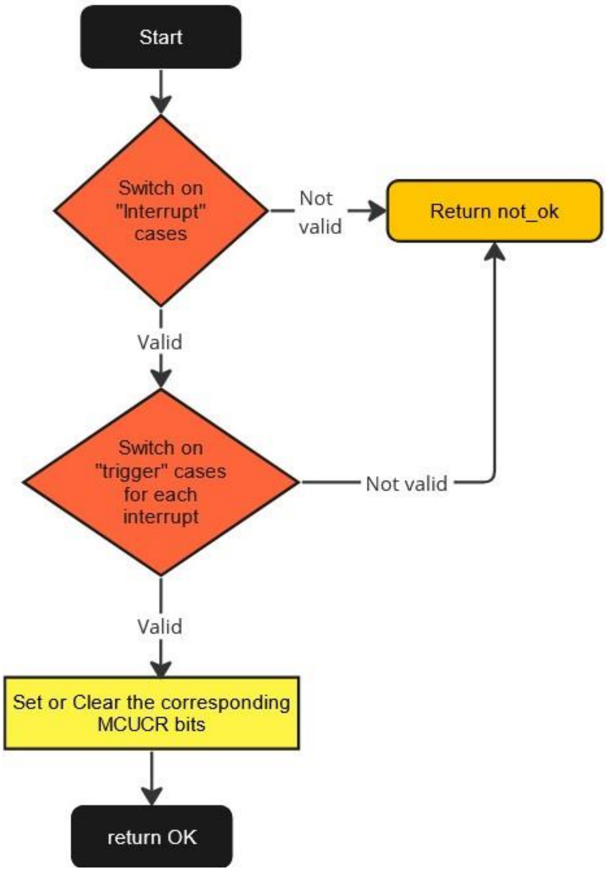
```
ISR (INTx_vect)
```



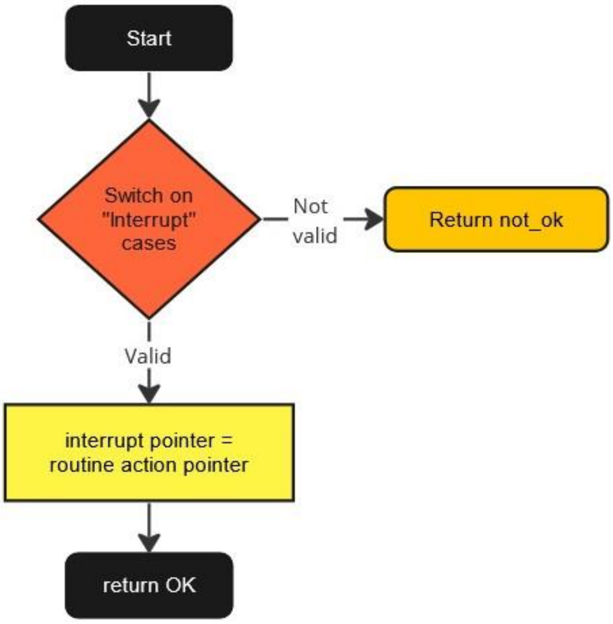
```
EN_int__error_t EXI_Disable (EN_int_t Interrupt)
```


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