Obstacle Avoidance Robot

By / Sharpel Malak

2023

Contents

INTRODUCTION

An Obstacle Avoidance Robot is an intelligent robot, which can automatically sense and overcome obstacles on its path. It contains of a Microcontroller to process the data, and Ultrasonic sensors to detect the obstacles on its path. Obstacle avoidance is one of the most important aspects of mobile robotics.

High Level Design

01) Layered Architecture

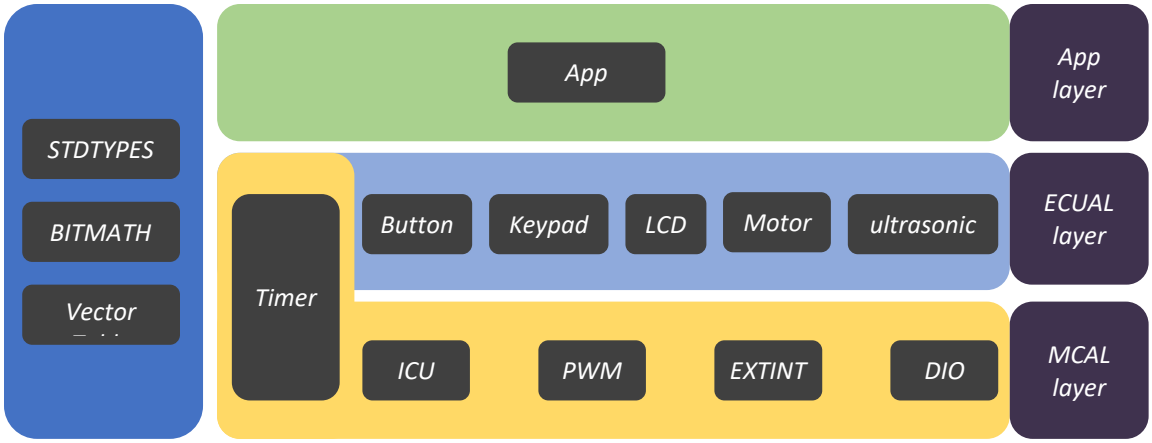


Figure 1: Layered Architecture

02) Modules Description

MCAL Layer:

- **DIO:** Controls GPIO Pins on MCU**.**
- **External Interrupt:** Handle External Interrupts events.
- **Timer:** Delay Generator it could be Sync or Async.
- **PWM:** Controls the Speed of Motors.
- **ICU**: Used to calculate time of specific Period.

HAL Layer:

- **Button:** Handle Dealing with the Button (rotation Button)
- **Keypad :** Handle Dealing with Two Buttons (Start and Stop Buttons)
- **LCD:** Display State of the Robot and all other data.
- **Ultrasonic:** By helping of ICU we can calculate Distance throw it.
- **Motor:** Controls Movement Direction and start or stop the robot.

Service Layer:

- **STD_Types:** Contains all the standard types used by all the layers.
- **BIT_Math:** Provides bit-wise operations.
- **Vect_table:** Contains all interrupt vectors and provides macros for dealing with general interrupt.

Application Layer:

- Contains the main logic of the project.

03) Drivers' Documentation

MCAL Layer:

- **DIO:**

```
/*
Description : This function initialize PIN and set it's direction
ARGS        : take PIN Number and PORT Number and Direction (INPUT,OUTPUT)
return      : return DIO_OK if the PIN initializes correctly, DIO_NOT_OK otherwise
*/
EN_DIO_ERROR DIO_init(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber,EN_DIO_DIRECTION
direction);

/*
Description : This function write on PIN and set it's level
ARGS        : take PIN Number and PORT Number and level (LOW,HIGH)
return      : return DIO_OK if the PIN level sets correctly, DIO_NOT_OK otherwise
*/
EN_DIO_ERROR DIO_write(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber,EN_DIO_LEVEL level);

/*
Description : This function toggles PIN level
ARGS        : take PIN Number and PORT Number
return      : return DIO_OK if the PIN toggles correctly, DIO_NOT_OK otherwise
*/
EN_DIO_ERROR DIO_toggle(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber);

/*
Description : This function reads PIN level and store it in the variable
ARGS        : take PIN Number and PORT Number and pointer to the variable
return      : return DIO_OK if the PIN value stored correctly , DIO_NOT_OK otherwise
*/
EN_DIO_ERROR DIO_read(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber, uint8_t * value);
```

- **External Interrupt:**

```
/*
Description : This function initializes the GLOBAL_INTERRUPT
ARGS        : takes the state ( ENABLE OR DISABLE )
return      : return EXTINT_OK if the PIN initializes correctly, EXTINT_NOT_OK
otherwise
*/
EN_EXTINT_ERROR SET_GLOBAL_INTERRUPT(EN_GLOBAL_INT state);


/*
Description : This function initializes the external interrupt number and it's
detecting type
ARGS        : takes the EXINT_NUMBER( INT0,INT1 OR INT2) and sense control.
return      : return EXTINT_OK if the EXINT_NUMBER initializes correctly,
EXTINT_NOT_OK otherwise
*/
EN_EXTINT_ERROR EXTINT_init(EN_EXINT_NUMBER INTx ,EN_Sence_Control INTxSense);

/*
Description : This function takes the external interrupt number and initialize call
back function.
ARGS        : takes the EXINT_NUMBER( INT0,INT1 OR INT2) and pointer to the function
we want to execute.
return      : return EXTINT_OK if the EXINT_NUMBER initializes correctly,
EXTINT_NOT_OK otherwise
*/
EN_EXTINT_ERROR EXTINT_CallBack(EN_EXINT_NUMBER INTx,void(*ptrfunc)(void));
```

- **Timer:**

```
/************************************************************************
   Function    : timer0_init
   Description : this function selects Normal mode and enable peripheral interrupt
   Args        : Void
   Return      : Void
************************************************************************/
void timer0_init(void);

/************************************************************************
   Function    : timer0_start
   Description : this function selects prescaller to start the timer
   Args        : Void
   Return      : Void
************************************************************************/

void timer0_start(void);

/************************************************************************
   Function    : timer0_stop
   Description : this function selects no prescaller to stop the timer
   Args        : Void
   Return      : Void
************************************************************************/

void timer0_stop(void);

/************************************************************************
   Function    : timer0_set_delay
   Description : this function calculate total ticks and init TCNT reg and NumOvf
   Args        : delay in milli seconds
   Return      : Void
************************************************************************/

void timer0_set_delay(uint32_t delay_ms);

/************************************************************************
   Function    : TIMER_INT_CallBack
   Description : assign ptr to function we want to execute when ISR Fired
   Args        : ptr to function
   Return      : Void
************************************************************************/

void TIMER_INT_CallBack(void(*ptrfunc)(void));
```

- **ICU :**

```
/*******************************************************************************
   Function    : SwICU_Init
   Description : Init pin as input and Init Interrupt
   Args        : Void
   Return      : Void
*******************************************************************************/
void SwICU_Init(void);

/*******************************************************************************
   Function    : SwICU_GetTime
   Description : calculate time taken from rising to falling Edges
   Args        : counter of timer
   Return      : time taken from rising to falling Edges
*******************************************************************************/
Uint16_t SwICU_GetTime(Uint16_t u16_a_TimCount);
```

- **PWM :**

```
/*Description : This function selects the normal mode and enables the GLOBAL_INTERRUPT
and overflow interrupt for timer2
ARGS        : void
return      : void*/
void timer2_init(void);

/*Description : This function selects the prescaler (clk/1024). the timer start
counting once we call this function.
ARGS        : void
return      : void*/
void timer2_start(void);


/*Description : This function selects the no clock source option. the timer stop
counting once we call this function.
ARGS        : void
return      : void*/
void timer2_stop(void);

/*Description : This function calculate the on time based on duty cycle we need .
ARGS        : takes the duty cycle
return      : void*/
void timer2_set_pwm_normal(Uchar8_t dutycycle);
```

HAL Layer:

- **Button**

```
/*
Description : This function initialize PIN and set it's direction as Input
ARGS        : take PIN Number and PORT Number
return      : return BTN_OK if the PIN initializes correctly, BTN_NOT_OK otherwise
*/

EN_BTN_Error_t Button_init(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber);

/*
Description : This function Read PIN value and store it in variable
ARGS        : take PIN Number and PORT Number and the address of the variable
return      : return BTN_OK if the PIN read correctly, BTN_NOT_OK otherwise
*/
EN_BTN_Error_t Button_read(EN_DIO_PINS pinNumber,EN_DIO_PORTS portNumber,uint8_t
*value);
```

- **Keypad :**

```
/*************************************************************************************
 Name : KEYPAD_init()
 Description : Initializes keypad pins(Rows are outputs & Columns are inputs).
 ARGS : void
 return : void
 **********************/
void KEYPAD_init(void);


/*************************************************************************************
 Name : KEYPAD_GetButton
 Description : This Function loops over other three functions (Checks (R1,R2,R3)).
 ARGS : void
 return : the pressed key or Nothing pressed
**********************/
EN_KEYPAD_BTNS KEYPAD_GetButton(void);

/*************************************************************************************
 Name : KEYPAD_checkR1 ,  KEYPAD_checkR2, KEYPAD_checkR3
 Description : functions are checking the entire row if it pressed or not.
 ARGS : void
 return : the pressed key or Nothing pressed

*************************************************************************************
**********************/

EN_KEYPAD_BTNS KEYPAD_checkR1(void);
EN_KEYPAD_BTNS KEYPAD_checkR2(void);
EN_KEYPAD_BTNS KEYPAD_checkR3(void);
```

- **LCD :**

```
/*
Description : This function initialize LCD
ARGS        : void
return      : void
*/
void LCD_init(void);

/*
Description : This function Send CMD To LCD
ARGS        : cmd
return      : void
*/

void LCD_sendCommand(uint8_t cmd);

/*
Description : This function Send Character To LCD
ARGS        : charData
return      : void
*/

void LCD_sendChar(uint8_t charData);

/*
Description : This function Send string To LCD
ARGS        : pointer to string
return      : void
*/

void LCD_sendString(uint8_t * strData);
```

- **Ultrasonic :**

```
/*
Description : This function initialize Trigger and Echo PINs and set there direction
ARGS        : take trigger and echo PINs Number
return      : return US_OK if the PINs initialize correctly, US_NOT_OK otherwise
*/

EN_US_Error_t US_init(EN_DIO_PINS triggerPin, EN_DIO_PINS echoPin);

/*
Description : This function calls init icu and and geticu(time)
ARGS        : pointer to store distance in it
return      : return US_OK if the distance stored, US_NOT_OK otherwise
*/

EN_US_Error_t US_getDistance(float32 *distance);
```

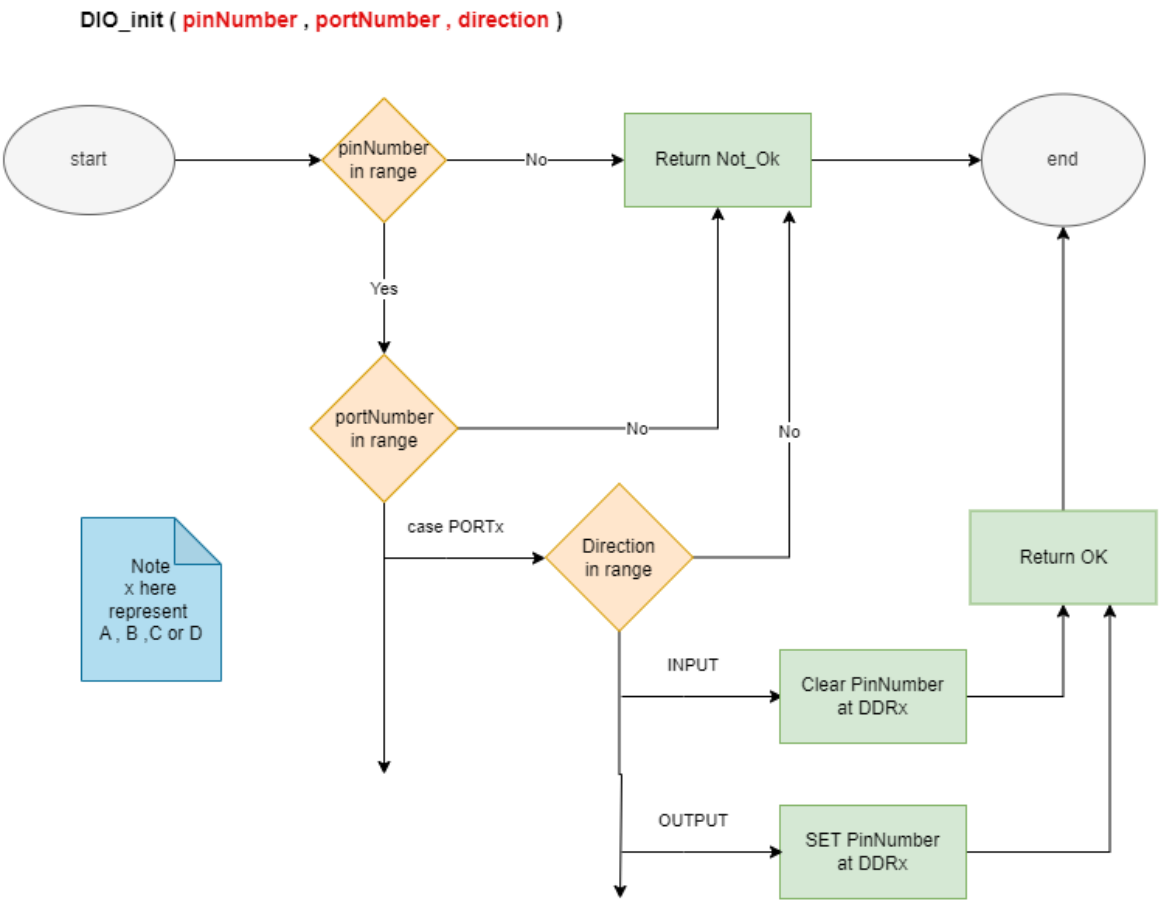
- **Motor**

```
/**
 * \brief initialize motor pins
 * \param pst_a_Motor reference to desired motor
 * \return en_MotorError_t
 */
en_MotorError_t DCM_Init(st_Motor_t *pst_a_Motor);

/**
 * \brief Function to start the given motor
 * \param pst_a_Motor reference to desired motor
 * \return en_MotorError_t
 */
en_MotorError_t DCM_Start(st_Motor_t *pst_a_Motor);


/**
 * \brief Function to stop the given motor
 * \param pst_a_Motor reference to desired motor
 * \return en_MotorError_t
 */

en_MotorError_t DCM_Stop(st_Motor_t *pst_a_Motor);
```

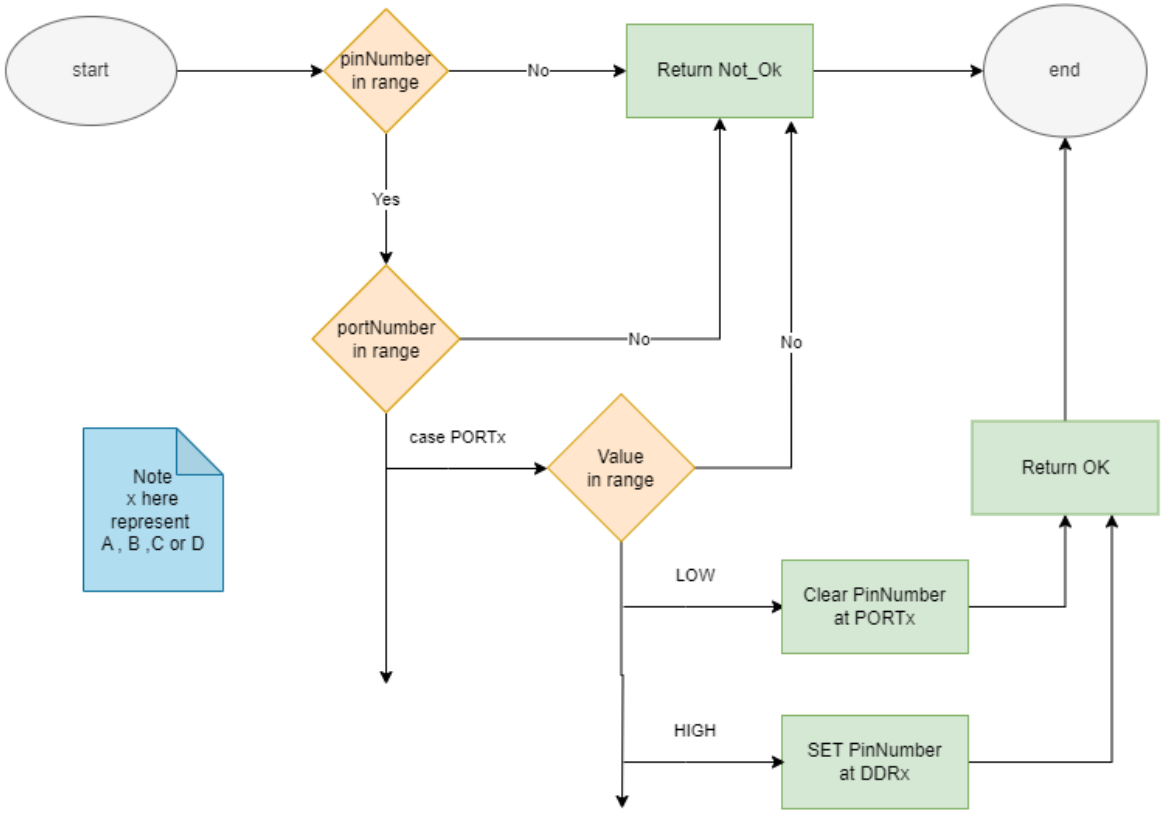
Low Level Design

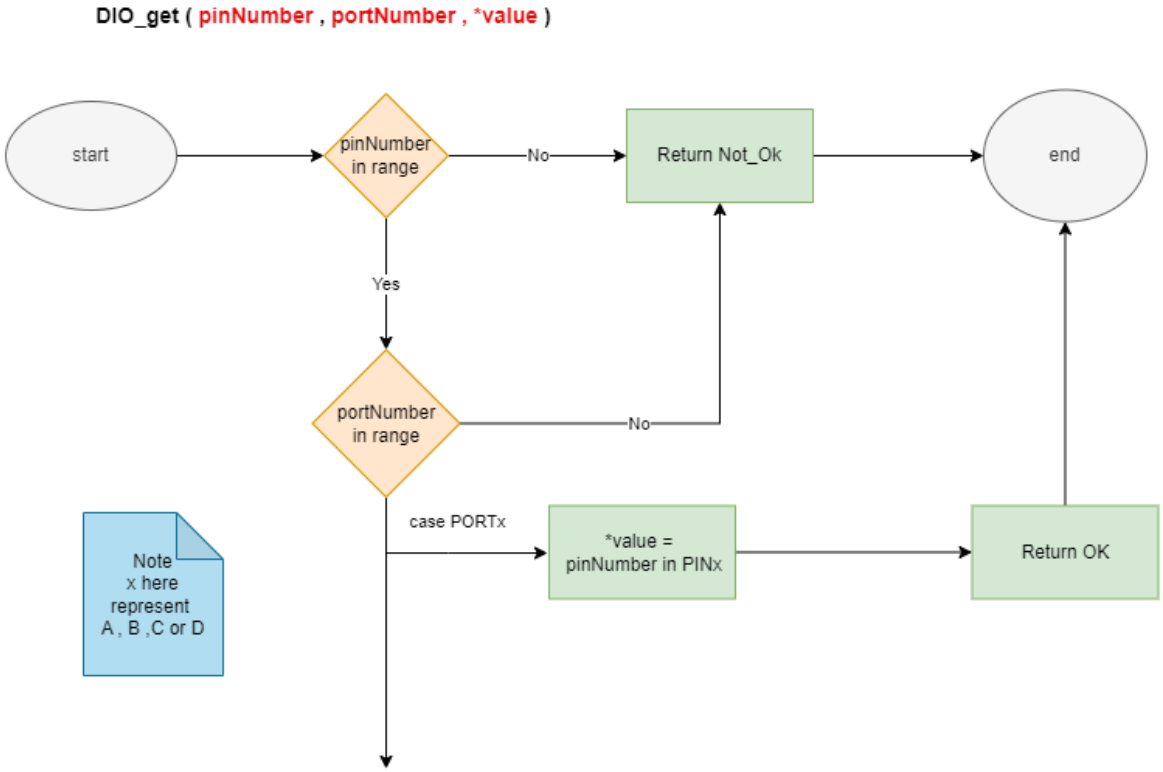
MCAL Layer:

- **DIO:**

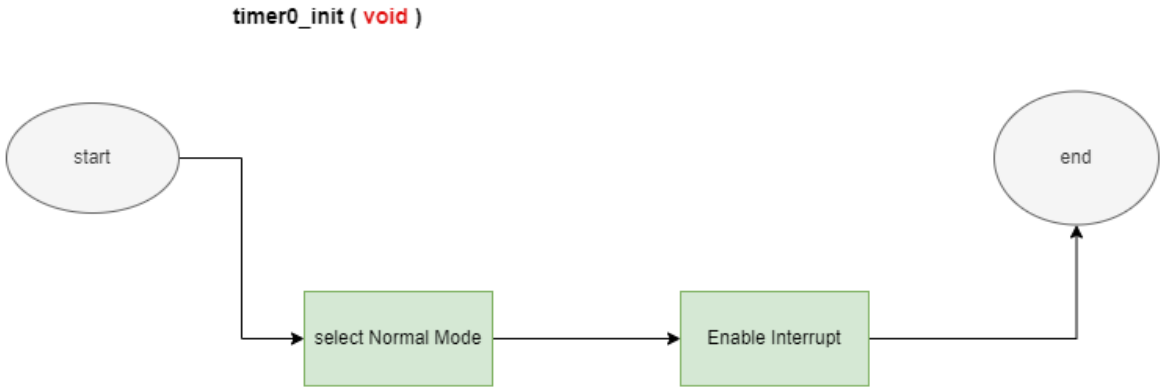


DIO_write (pinNumber , portNumber , Value)

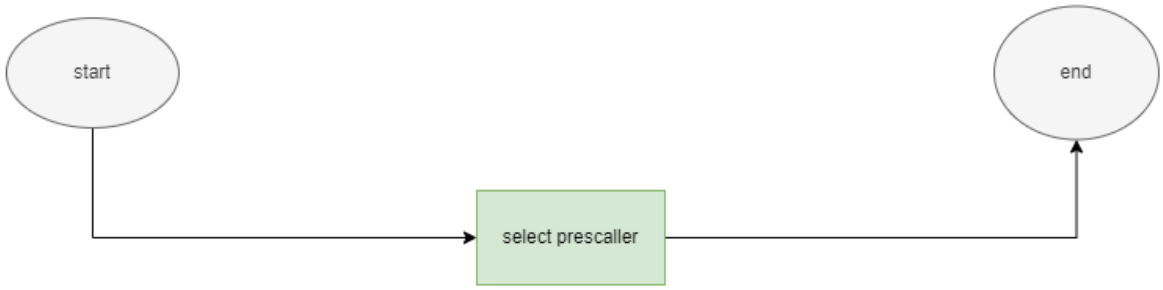



DIO_get (pinNumber , portNumber , *value)
start
pinNumber in range
No
Return Not_Ok
end
Yes
portNumber in range
No
case PORTx
*value = pinNumber in PINx
Return OK
Note x here represent A , B ,C or D

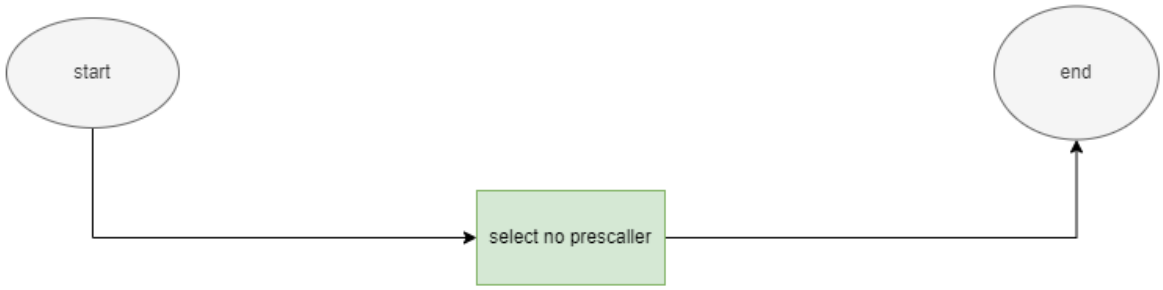
- **Timer :**


timer0_init (void)
start
select Normal Mode
Enable Interrupt
end

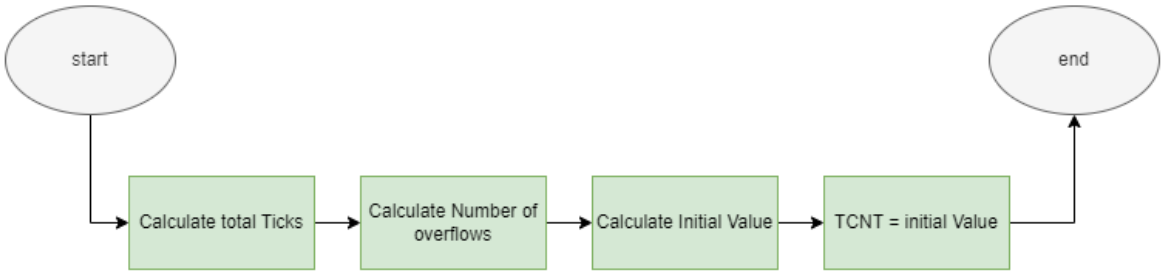
timer0_start (void)



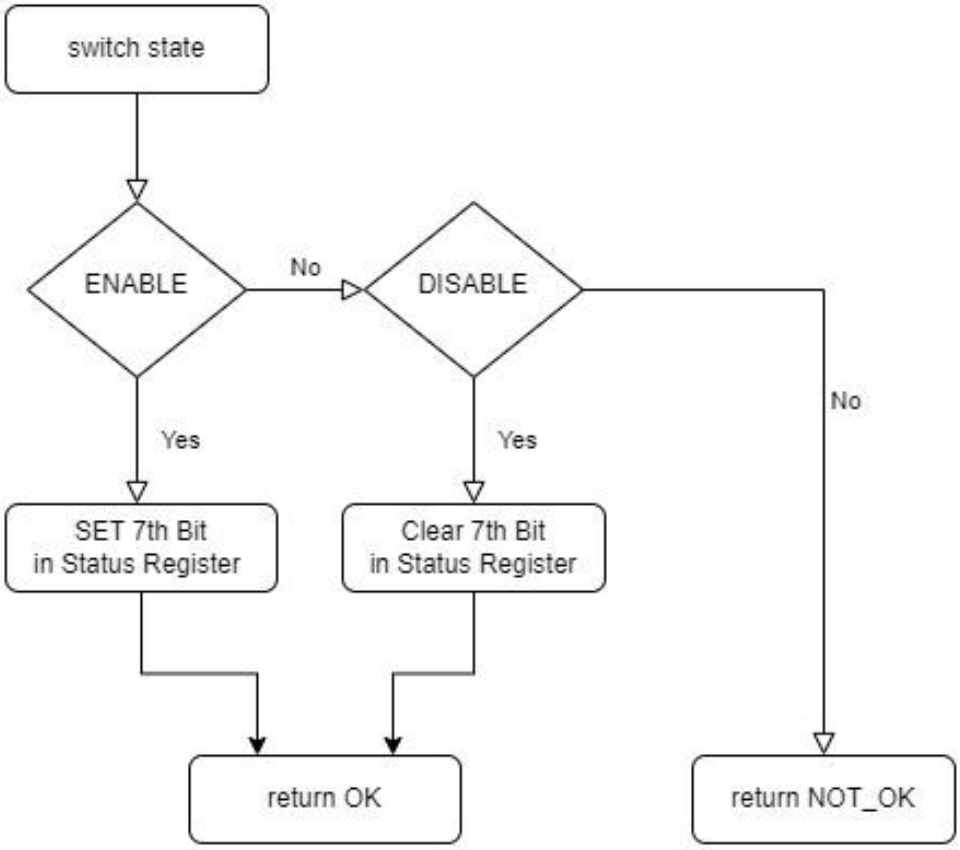
timer0_stop (void)

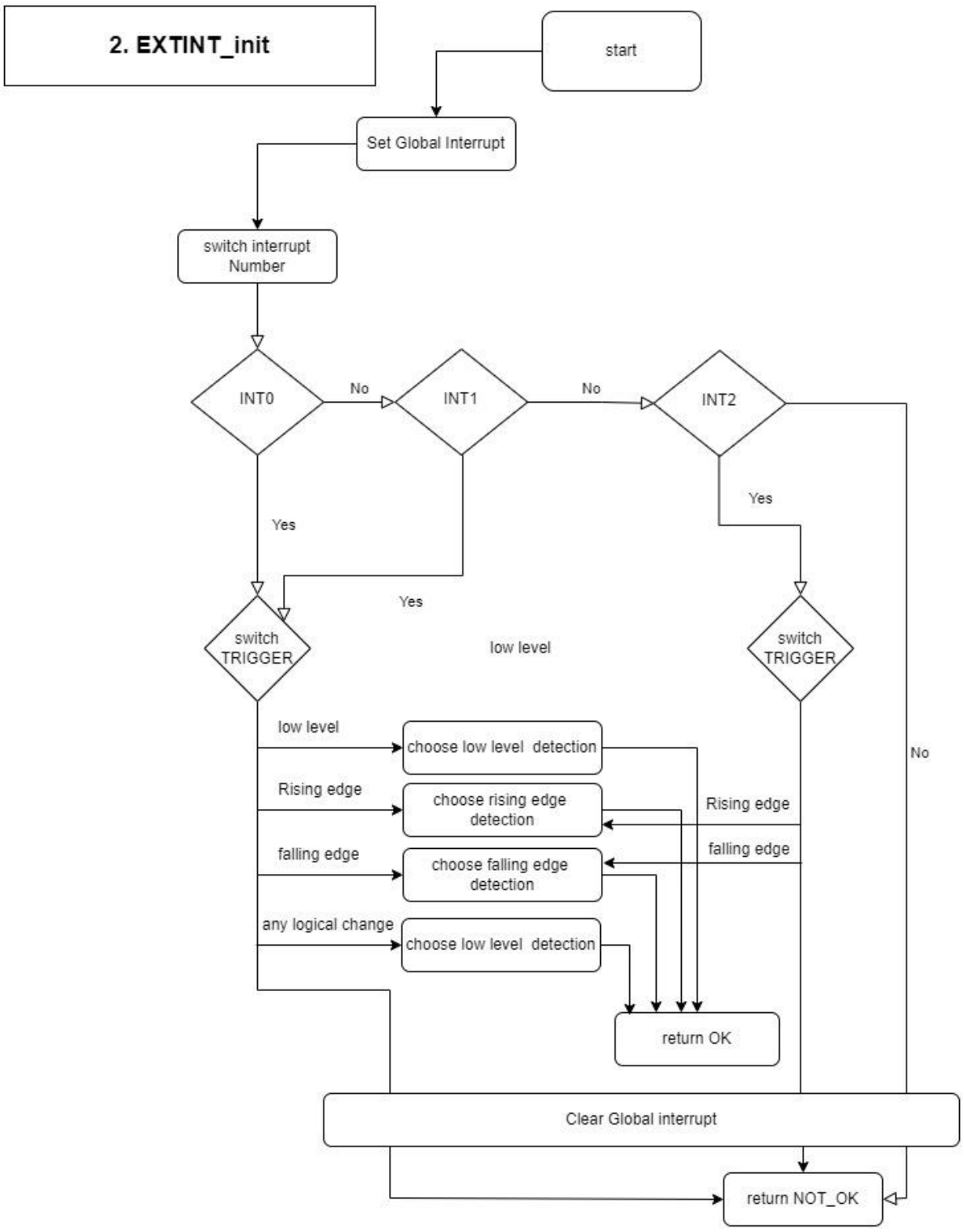


timer0_set_delay (delay)

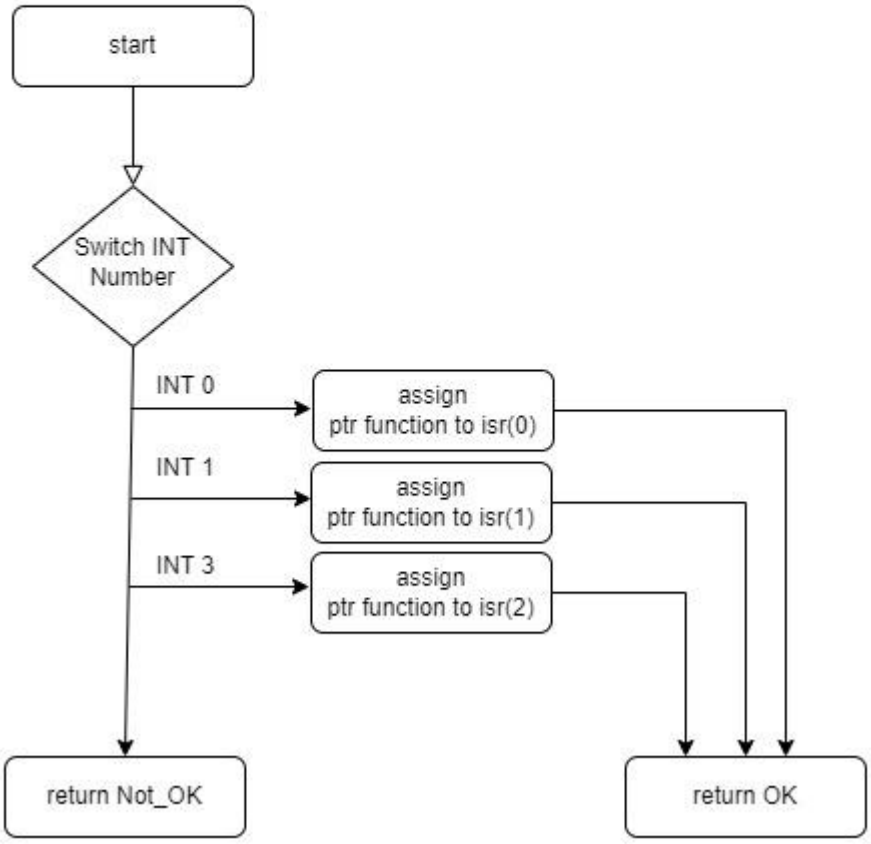


- **External Interrupt:**



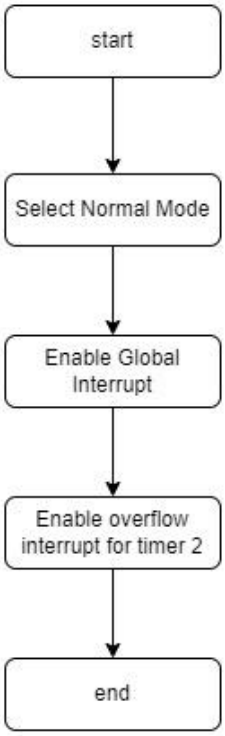

2. EXTINT_init
start
Set Global Interrupt
switch interrupt Number
INT0
No
INT1
No
INT2
Yes
Yes
Yes
switch TRIGGER
low level
switch TRIGGER
low level
choose low level detection
Rising edge
choose rising edge detection
Rising edge
falling edge
choose falling edge detection
falling edge
any logical change
choose low level detection
No
return OK
Clear Global interrupt
return NOT_OK

3. EXTINT_CALLBACK

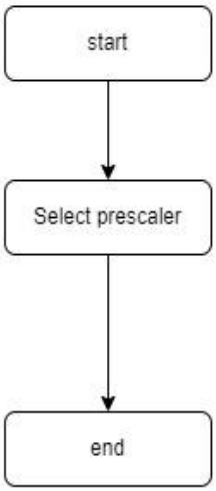


- **PWM :**

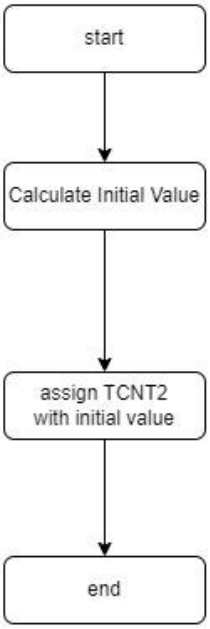
1. TIMER2_init



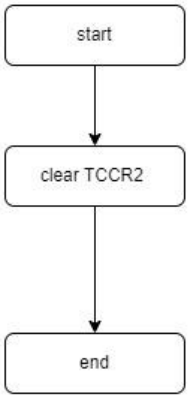
1. TIMER2_start



3. TIMER2_set_pwm

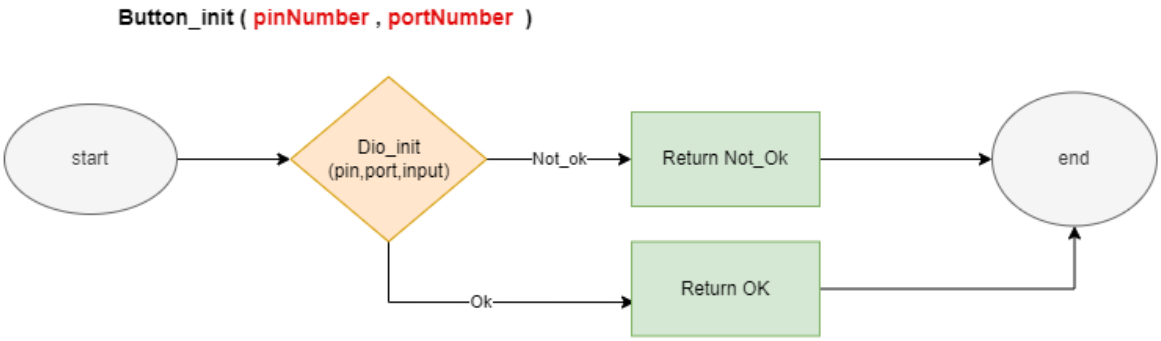


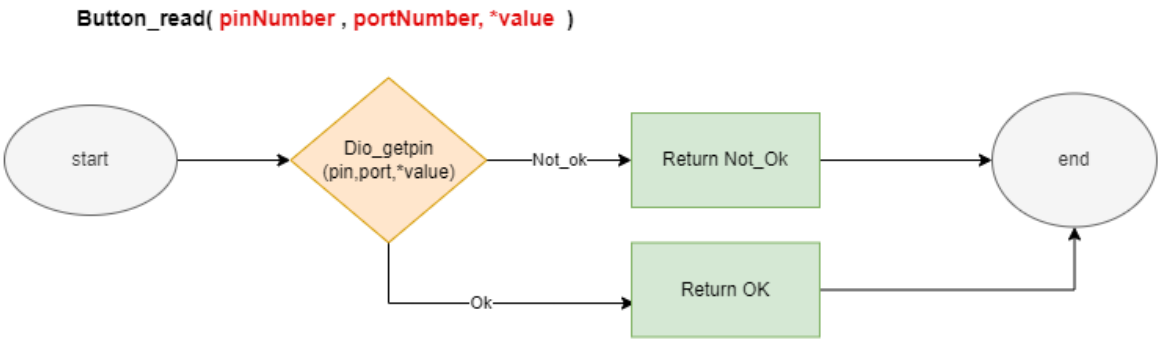
3. TIMER2_stop



HAL Layer:

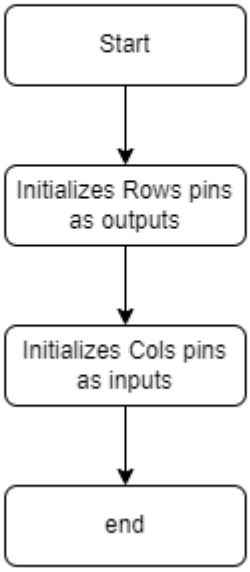
- **Button:**



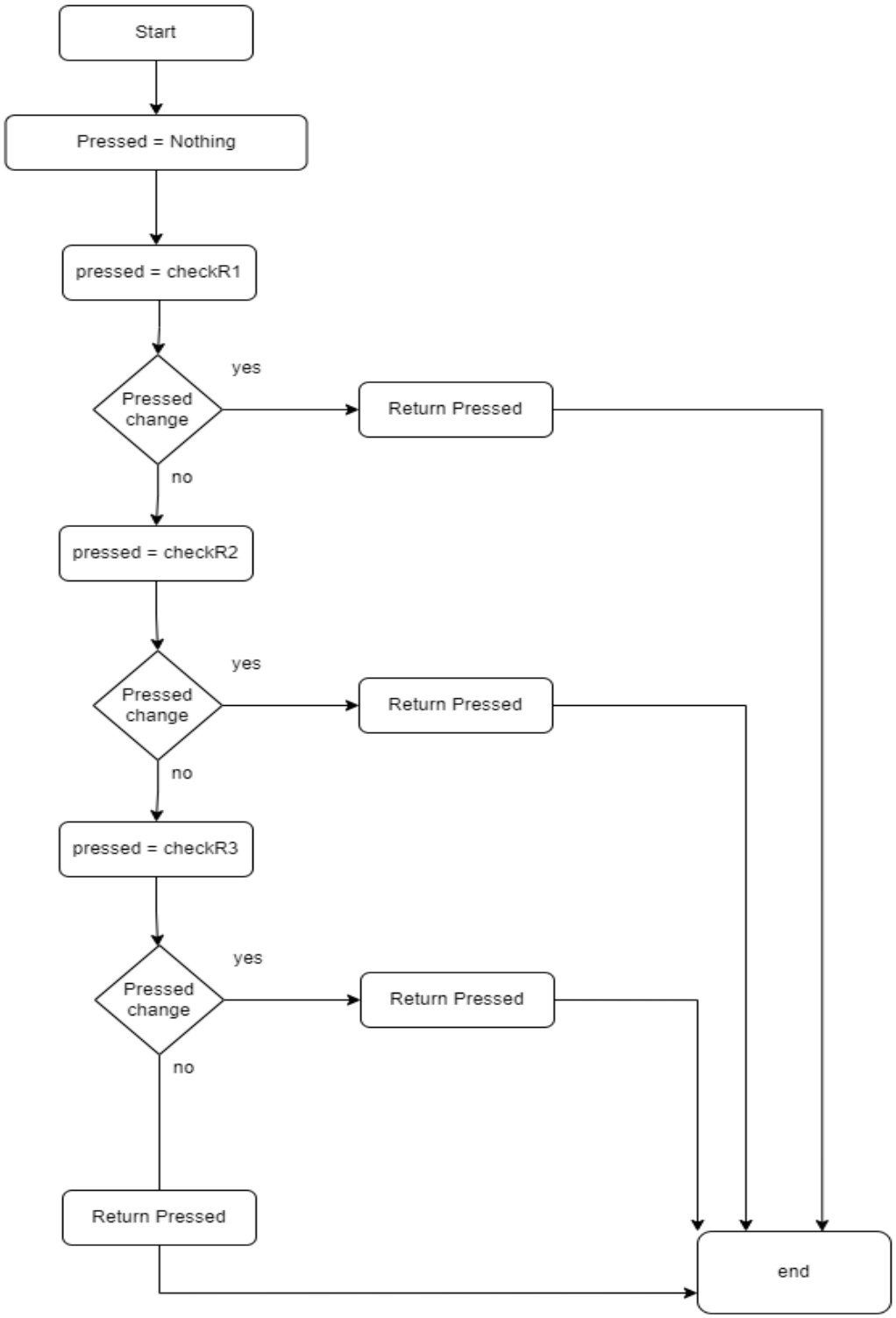


- **Keypad:**

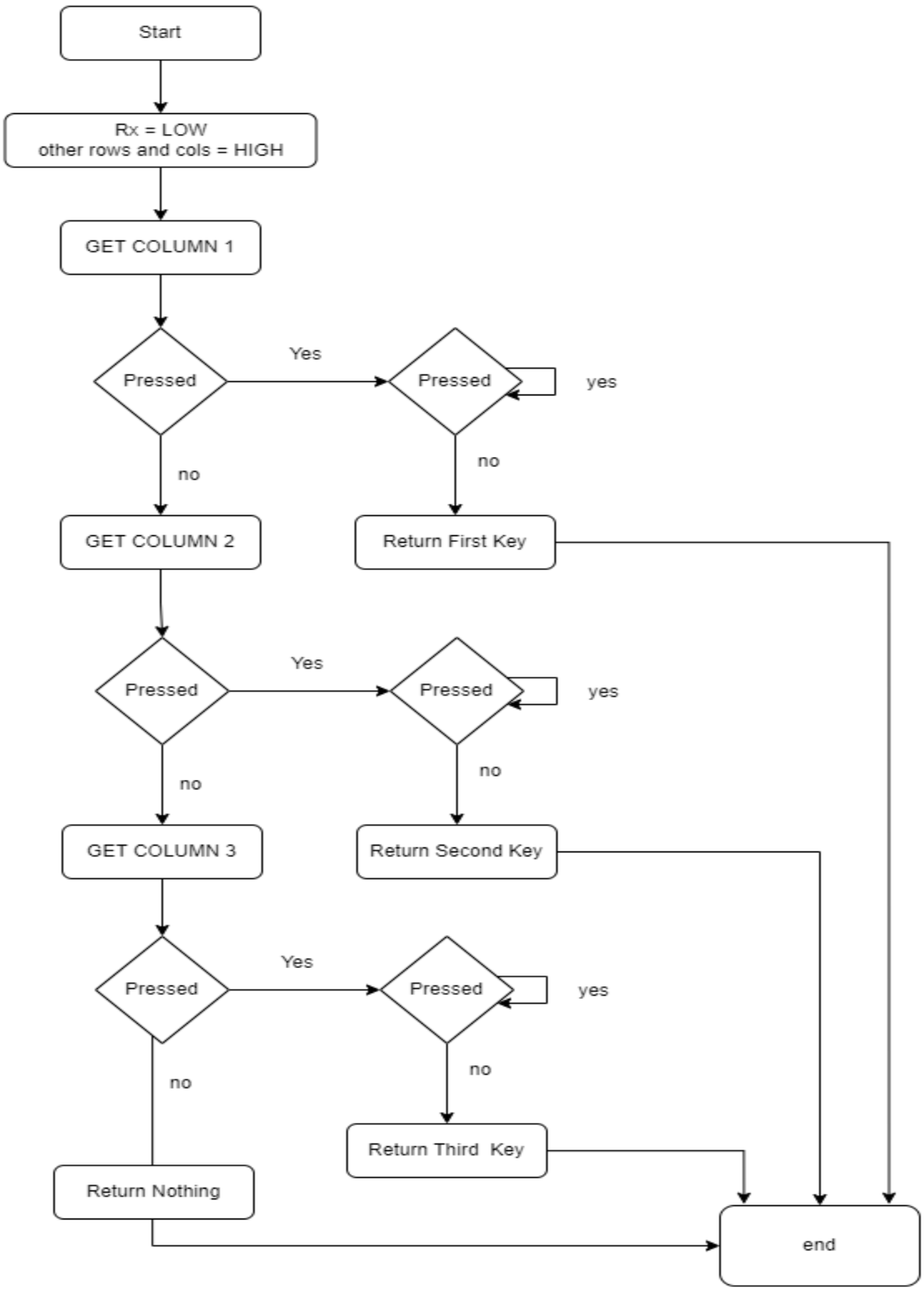
KEYPAD_init(void)



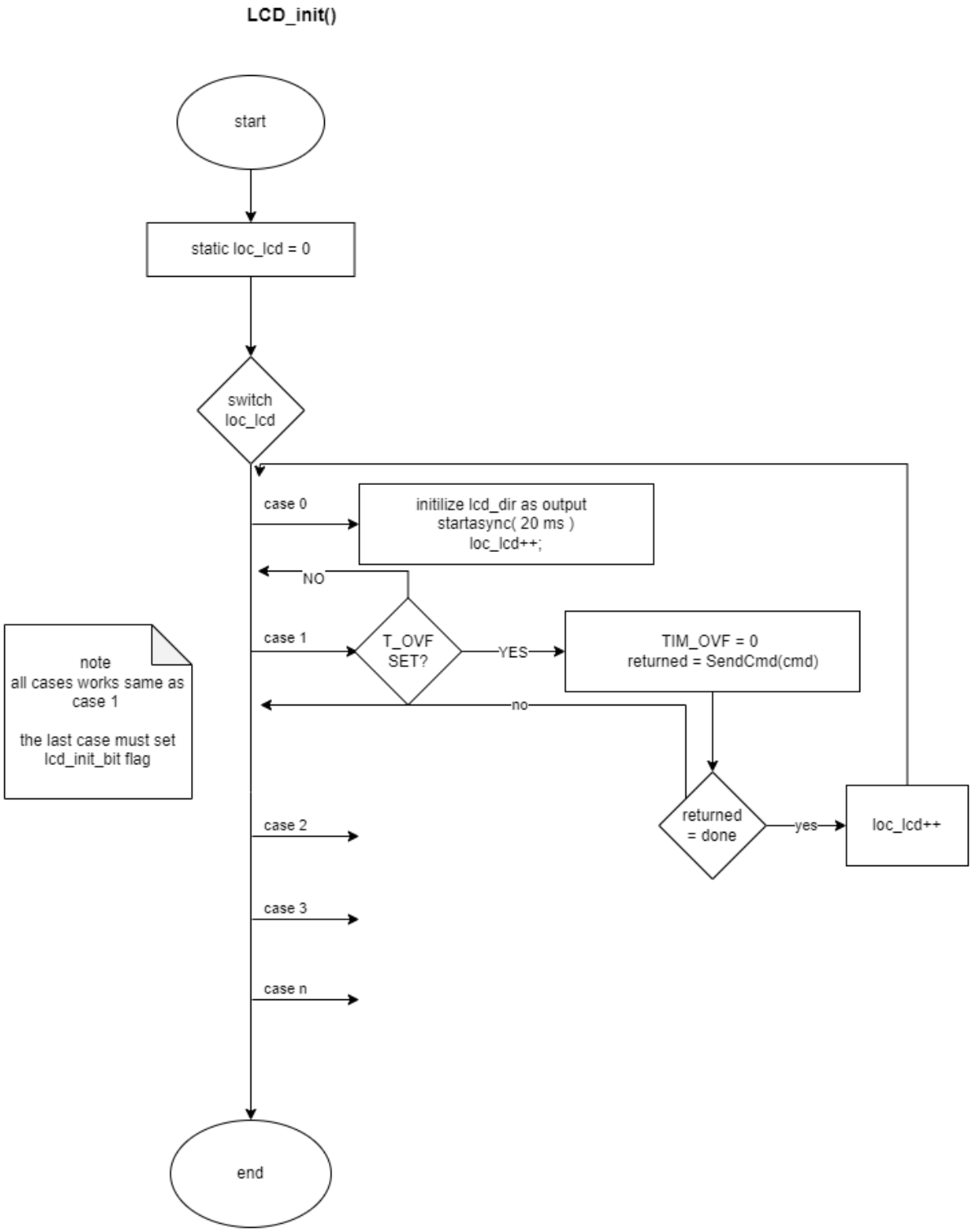
GetButton(void)

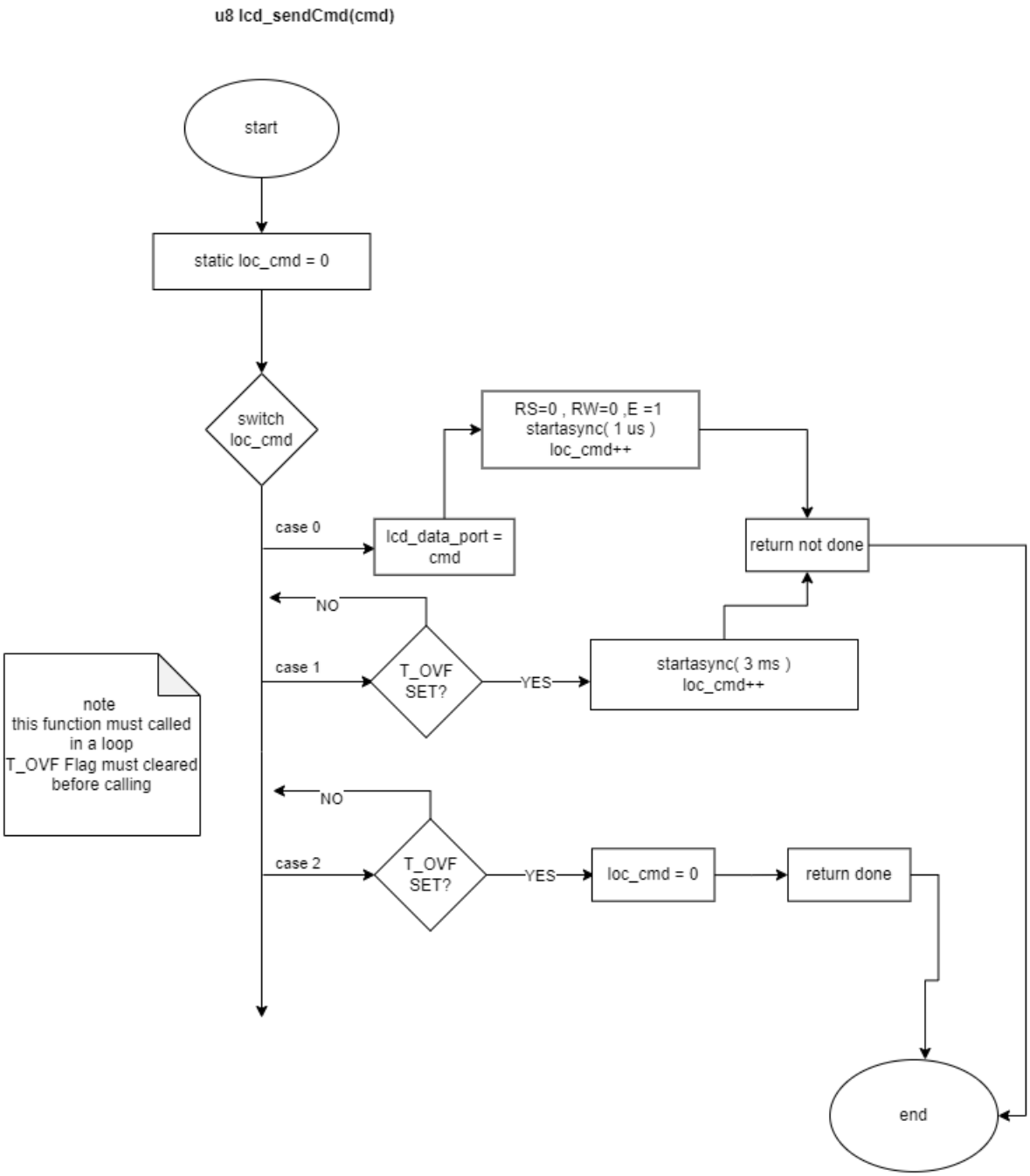


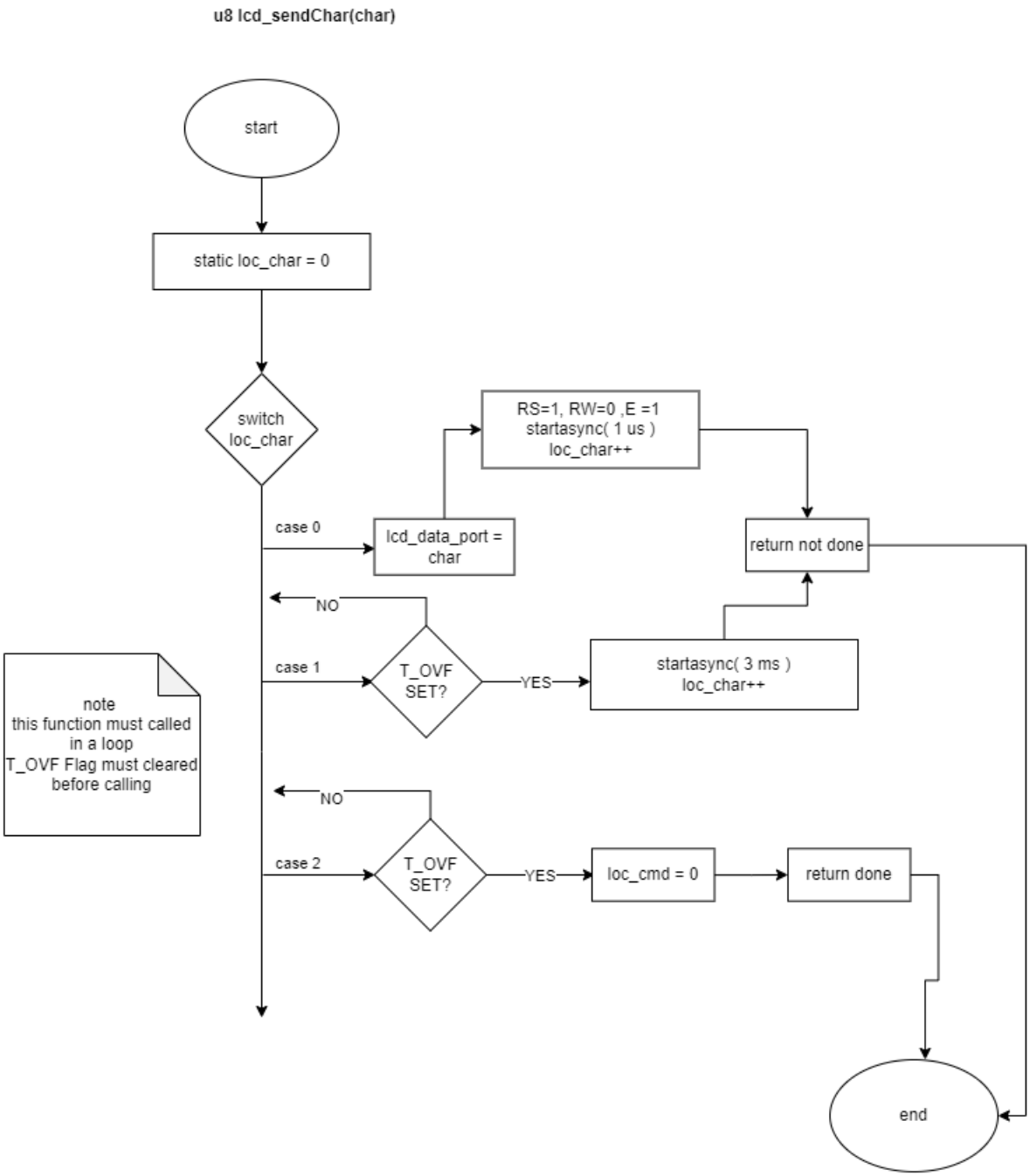
KEYPAD_CheckRx(void)
x here (1.2.3)



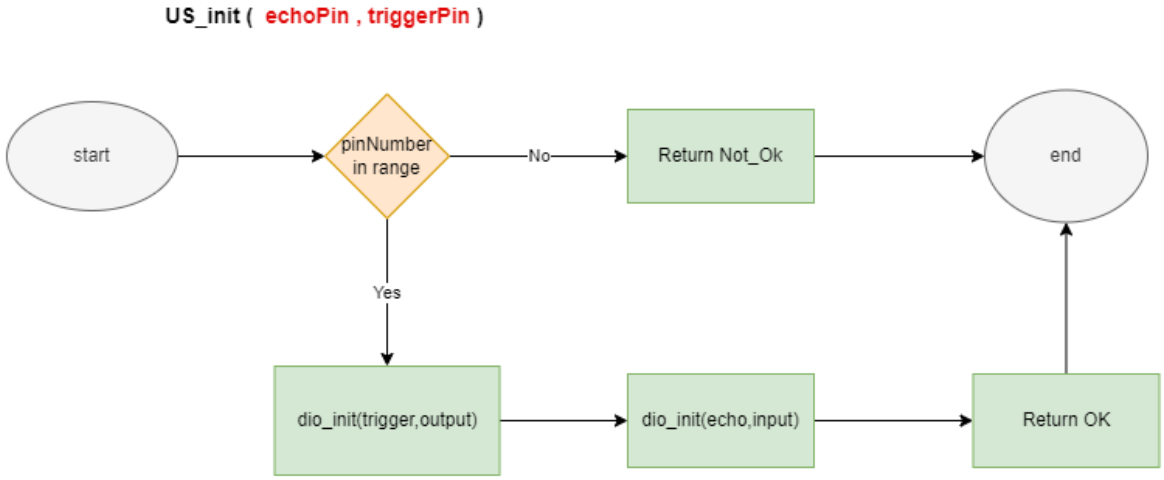
- **LCD :**

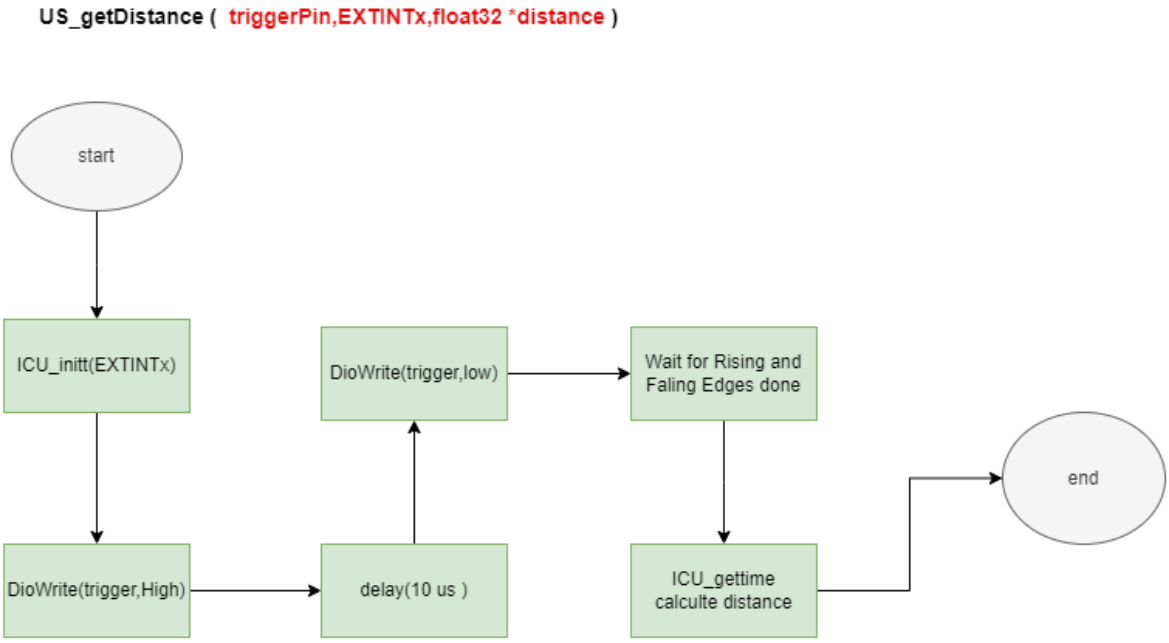

LCD_init()
start
static loc_lcd = 0
switch
loc_lcd
case 0
initilize lcd_dir as output
startasync(20 ms)
loc_lcd++;
NO
case 1
T_OVF
SET?
YES
TIM_OVF = 0
returned = SendCmd(cmd)
no
note
all cases works same as
case 1
the last case must set
lcd_init_bit flag
returned
= done
yes
loc_lcd++
case 2
case 3
case n
end


u8 lcd_sendCmd(cmd)
start
static loc_cmd = 0
switch loc_cmd
case 0
lcd_data_port = cmd
RS=0 , RW=0 ,E =1
startasync(1 us)
loc_cmd++
return not done
case 1
NO
T_OVF SET?
YES
startasync(3 ms)
loc_cmd++
case 2
NO
T_OVF SET?
YES
loc_cmd = 0
return done
end
note
this function must called in a loop
T_OVF Flag must cleared before calling


u8 lcd_sendChar(char)
start
static loc_char = 0
switch loc_char
case 0
lcd_data_port = char
RS=1, RW=0 ,E =1
startasync(1 us)
loc_char++
return not done
case 1
NO
T_OVF SET?
YES
startasync(3 ms)
loc_char++
note
this function must called in a loop
T_OVF Flag must cleared before calling
case 2
NO
T_OVF SET?
YES
loc_cmd = 0
return done
end

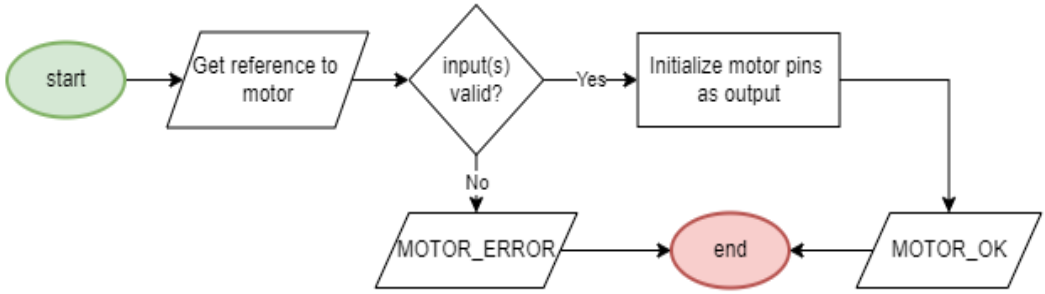
- **Ultrasonic :**


US_init (echoPin , triggerPin)
start
pinNumber in range
No
Return Not_Ok
end
Yes
dio_init(trigger,output)
dio_init(echo,input)
Return OK

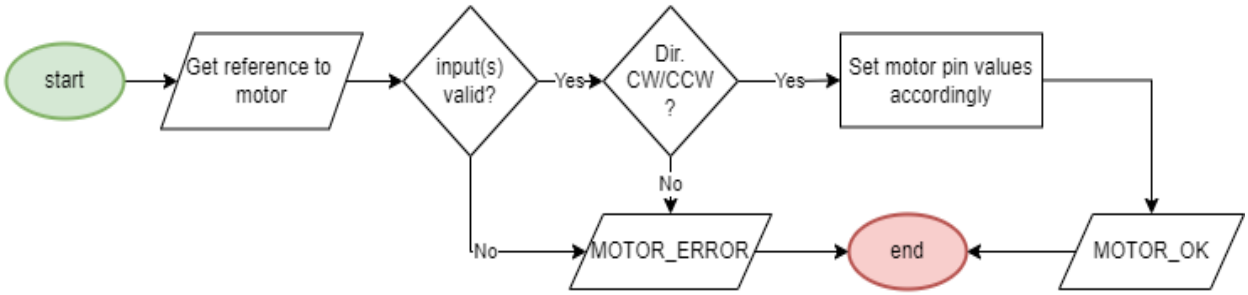

US_getDistance (triggerPin,EXTINTx,float32 *distance)
start
ICU_initt(EXTINTx)
DioWrite(trigger,low)
Wait for Rising and Faling Edges done
end
DioWrite(trigger,High)
delay(10 us)
ICU_gettime calculte distance

- **Motor**

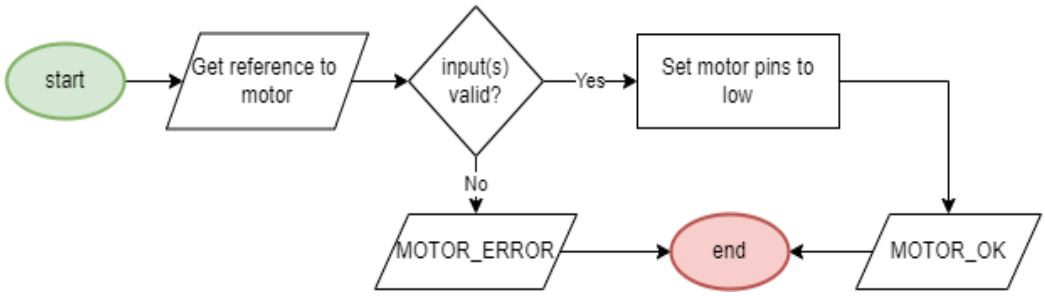
DCM_Init



DCM_Start

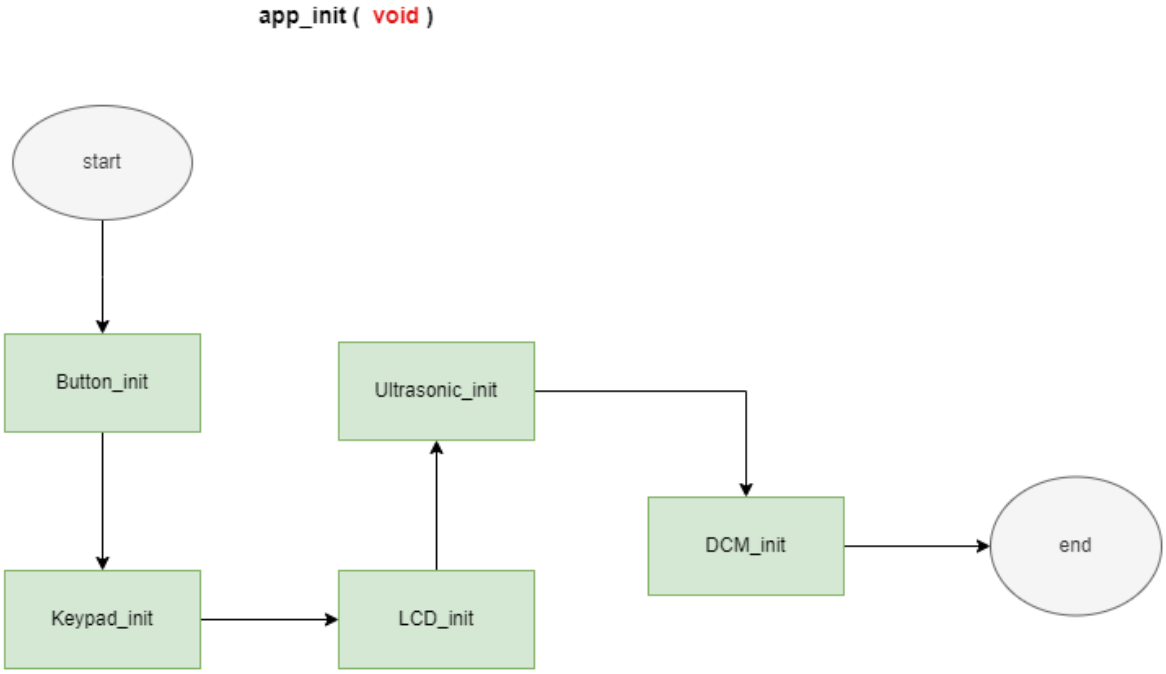


DCM_Stop

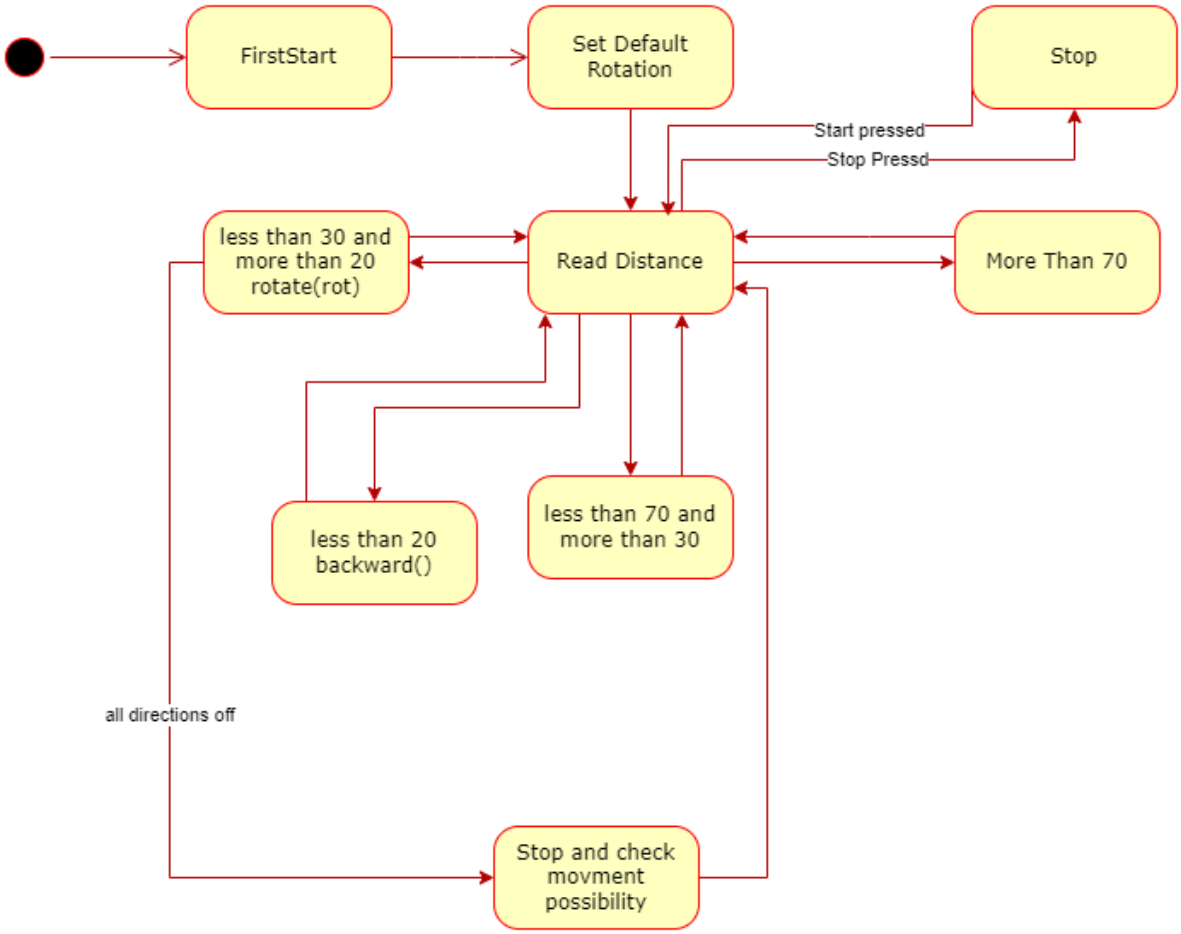


Application Layer:

- APP_Init



- APP_Start


FirstStart
Set Default Rotation
Stop
Start pressed
Stop Pressd
less than 30 and more than 20 rotate(rot)
Read Distance
More Than 70
less than 20 backward()
less than 70 and more than 30
all directions off
Stop and check movment possibility