Design document

project title: Obstacle avoidance car

Sarah Mohamed

Table of contents:

Project introduction

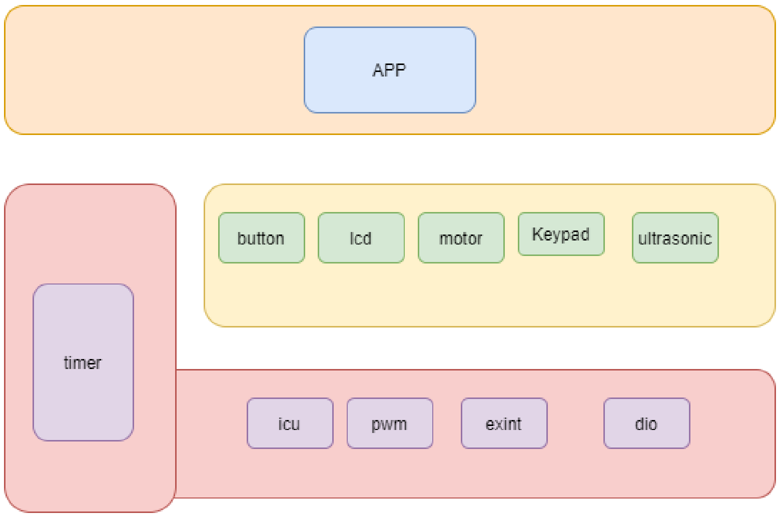
- **Description**

CARD MCU

1. The CARD MCU has two modes of operations
    1. Programming Mode
        1. The CARD MCU will enter this mode after reset
        2. For the first time only the MCU will send the following messages to the terminal
            1. “Please Enter Card PAN:” and wait for the PAN
            2. “Please Enter New PIN:” and wait for the PIN
            3. “Please Confirm New PIN:” and wait for the PIN
            4. If PIN is matched, then change to user mode
            5. If PIN is not matched, not numeric, and exceeds 4 characters, then “Wrong PIN” message is displayed and repeat from step 2
        3. For any further after resets
            1. “Please press 1 for entering user mode and 2 for programming mode: ” message is sent to the terminal and wait for a valid response, only accepts 1 or 2
        4. PAN is 16 to 19 length numeric string
        5. PIN is 4 numeric digits
        6. All data taken will be stored in the EEPROM
    2. User Mode
        1. The CARD MCU will enter this mode
            1. After completing the programming mode
            2. Or after choosing 2 in any further after resets
        2. In this mode, the CARD ECU will send a trigger signal to the ATM ECU that will make the ATM initiate its flow

High Level Design

- Layered architecture



- Modules Description

MCAL modules

GPIO

Using GPIO for initialize trigger function and apply trigger signal (rising – falling) edge to a specific pin

TIMER

Use TIMER for different delays

PWM

Control speed of motor

ICU

Calculate time of specific period

External interrupt

Handle external interrupt events

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.

Drivers Documentation Project Modules APIs

- **Keypad module APIs**

```
typedef struct {
            uint8 number_of_cols;
            uint8 number_of_rows;
            uint8 cols_first_pin;
            uint8 rows_first_pin;
            uint8 cols_port;
            uint8 rows_port;
}ST_KPD_t;


#define KPD_NO_KEY_PRESSED 12


void KPD_init(ST_KPD_t kpd);
```

Description:

used to initlaize the keypad

- **St_kpd_t: take structure to keypad rows or column**

```
void KPD_get_pressed_key(ST_KPD_t kpd,uint8*key);
```

Description:

used to determine key pressed

- **St_kpd_t: take structure to keypad rows or column**
- **Pointer : pointer to number of keys**

Drivers Documentation Project Modules APIs

- **LCD module APIs**

```
/* LCD Commands */
#define LCD_CLEAR_COMMAND             0x01
#define LCD_GO_TO_HOME                0x02
#define LCD_TWO_LINES_EIGHT_BITS_MODE  0x38
#define LCD_TWO_LINES_FOUR_BITS_MODE   0x28
#define LCD_CURSOR_OFF                0x0C
#define LCD_CURSOR_ON                 0x0E
#define LCD_SET_CURSOR_LOCATION       0x80



/*****************************************************************************************
 *                  Functions Prototypes
typedef struct {
            uint8 RS_port;
            uint8 RW_port;
            uint8 E_port;
            uint8 RS_pin;
            uint8 RW_pin;
            uint8 E_pin;
            uint8 lcd_data_port;
            uint8 lcd_data1_pin;
            uint8 lcd_data2_pin;
            uint8 lcd_data3_pin;
            uint8 lcd_data4_pin;
} ST_LCD_t;
```

void LCD_init(ST_LCD_t lcd);

Description:

used to Initialize the LCD:

 * 1. Setup the LCD pins directions by use the GPIO driver.
 * 2. Setup the LCD Data Mode 4-bits or 8-bits.

St_lCD_t: take structure to lcd port and pins

void LCD_sendCommand(ST_LCD_t lcd,uint8 command);

Description:

used Send the required command to the screen

- St_LCD_t: take structure to lcd port and pins
- command : pointer to number of keys

void LCD_displayString(ST_LCD_t lcd,const char *Str);

Description:

used to Display the required string on the screen

- St_lCD_t: take structure to lcd port and pins
- pointer : pointer to character

void LCD_moveCursor(ST_LCD_t lcd,uint8 row,uint8 col);

Description:

used Move the cursor to a specified row and column index on the screen

- St_LCD_t: take structure to lcd port and pins
- row : number or row
- Column: number of column

void LCD_displayStringRowColumn(ST_LCD_t lcd,uint8 row,uint8 col,const char *Str);

Description:

used to Display the required string in a specified row and column index on the screen

St_LCD_t: take structure to lcd port and pins

- row : number or row

void LCD_intgerToString(ST_LCD_t lcd,int data);

<u>Description</u>:

used to show decimal value on screen

- St_lCD_t: take structure to lcd port and pins
- data : show data on lcd

void LCD_clearScreen(ST_LCD_t lcd);

<u>Description</u>:

used to clear screen

- St_lCD_t: take structure to lcd port and pins

Drivers Documentation Project Modules APIs

- **Button module APIs**

```
#define BUTTON_HIGH 1
#define BUTTON_LOW 0

#define EXT_INT_BTN_PORT PORT_D
#define EXT_INT_BTN_PIN PIN2

#define BTN_2_SECOND 1500
#define BTN_MIN_SECOND 50

#define BTN_ZERO_PRESSED 10
#define BTN_ENTER_PRESSED 11
#define BTN_NO_PRESS        12

typedef struct {
            uint8 button_port;
            uint8 button_pin;
}ST_PBTN_t;

void BUTTON_init(ST_PBTN_t button);
```

Description:
used to Initialize the button
St_PBTN_t: take structure to button port and pins

```
void BUTTON_status(ST_PBTN_t button,uint8 *status);
```

Description:
used to set button status
St_PBTN_t: take structure to button port and pins
Pointer:set status of pointer

void BUTTON_read_zero_enter(ST_PBTN_t button,uint8 * value);

<u>Description</u>:

used to read value of button

- **St_PBTN_t: take structure to button port and pins**
- **Value:set value of button**

void BUTTON_enter(void);

<u>Description</u>:

used to read value of button

Drivers Documentation Project Modules APIs

- **External Interrupt:**

```
EN_EXTINT_ERROR SET_GLOBAL_INTERRUPT(EN_GLOBAL_INT
state);

/*
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 EXTINT_init(EN_EXINT_NUMBER INTx
,EN_Sence_Control INTxSense);

/*
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_CallBack(EN_EXINT_NUMBER
INTx,void(*ptrfunc)(void));
/*
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 */
```

Drivers Documentation Project Modules APIs

- **ICU module APIs**

```
void SwICU_Init(void);
Function : SwICU_Init
Description : Init pin as input and Init Interrupt
Args : Void
Return : Void
Uint16_t SwICU_GetTime(Uint16_t u16_a_TimCount);

Function : SwICU_GetTime
Description : calculate time taken from rising to
falling Edges
Args : counter of timer
Return : time taken from rising to falling Edges
```

Drivers Documentation Project Modules APIs

- **PWM module APIs**

```
void timer2_init(void);

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

void timer2_start(void);

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

void timer2_stop(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_set_pwm_normal(Uchar8_t dutycycle);

/*Description : This function calculate the on
time based on duty cycle we need .
ARGS : takes the duty cycle
return : void*/
```

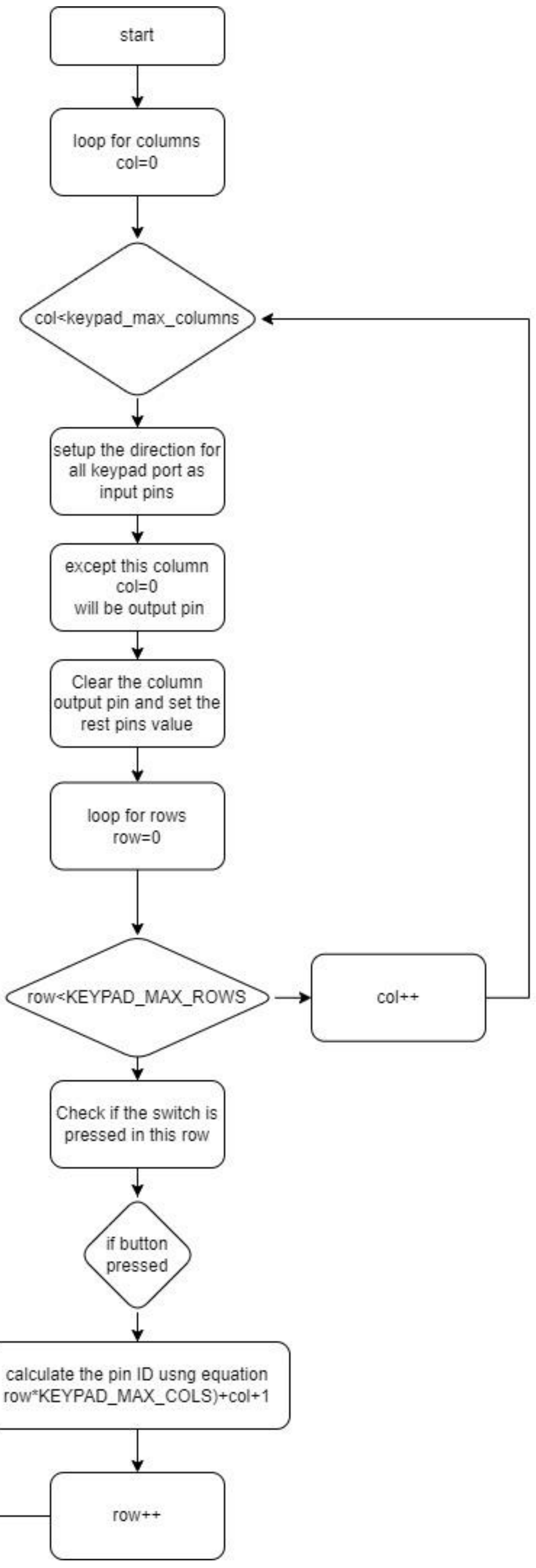
Drivers Documentation Project Modules APIs

- **ultrasonic module APIs**

```
EN_US_Error_t US_init(EN_DIO_PINS triggerPin,
EN_DIO_PINS echoPin);

/*
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_getDistance(float32 *distance);


/*
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
*/
```

GPIO APIs flowchart

uint8 KEYPAD_getPressedKey(void);


start
loop for columns col=0
col<keypad_max_columns
setup the direction for all keypad port as input pins
except this column col=0 will be output pin
Clear the column output pin and set the rest pins value
loop for rows row=0
row<KEYPAD_MAX_ROWS
col++
Check if the switch is pressed in this row
if button pressed
calculate the pin ID usng equation row*KEYPAD_MAX_COLS)+col+1
row++

Drivers Documentation Project Modules APIs

▪ GPIO module APIs

/*============= TYPE DEFINITION =============*/

```
typedef enum{
PIN_INPUT,PIN_OUTPUT
}EN_PIN_DIRECTION;

typedef enum
{
PORT_INPUT,PORT_OUTPUT=0xFF
}EN_PORT_DIRECTION;

typedef enum{
Low,High
}EN_PIN_VALUE;

typedef enum{
LOW,HIGH=0xFF
}EN_PORT_VALUE;

typedef enum{
FAILED,SUCCESS
}EN_STATE;
```

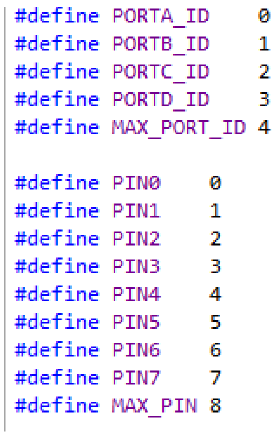
/*============= FUNCTION PROTOTYPE=============*/

EN_STATE GPIO_setPinDirection(uint8 port_num, uint8 pin_num, EN_PIN_DIRECTION direction);

Description:

Used to set specific pin direction as input or output pin

- **Port_num: determine port in which pin is connected, you have to use port_ID which defined as MACROS**
- **Pin_num: determine pin number , you have to use PIN which defined as MACROS**
- **Direction: used to determine direction of the pin, you have to use Enum EN_PIN_DIRECTION (PIN_INPUT,PIN_OUTPUT)**
- **Return : function check for the range of port_ID and PIN number Return SUCCESS if true and FAILED if out of range**



```
#define PORTA_ID    0
#define PORTB_ID    1
#define PORTC_ID    2
#define PORTD_ID    3
#define MAX_PORT_ID 4

#define PIN0    0
#define PIN1    1
#define PIN2    2
#define PIN3    3
#define PIN4    4
#define PIN5    5
#define PIN6    6
#define PIN7    7
#define MAX_PIN 8
```

EN_STATE GPIO_checkstate(uint8 port_num,uint8 pin_num);

Description:

Used to check for the range of port_ID and pin range

- **Port_num: determine port in which pin is connected, you have to use port_ID which defined as MACROS**
- **Pin_num: determine pin number , you have to use PIN which defined as MACROS**
- **Return : function check for the range of port_ID Return SUCCESS if true and FAILED if out of range**

Drivers Documentation Project Modules APIs

- **GPIO module APIs**

EN_STATE GPIO_writePin(uint8 port_num, uint8 pin_num, EN_PIN_VALUE value);

Description:

used to write high or low to specific pin

- Port_num: determine port in which pin is connected, you have to use port_ID which defined as MACROS
- Pin_num: determine pin number , you have to use PIN which defined as MACROS
- Value: used to determine direction of the pin, you have to use Enum EN_PIN_VALUE (Low,High)
- Return : function check for the range of port_ID and PIN number Return SUCCESS if true and FAILED if out of range

EN_STATE GPIO_readPin(uint8 port_num, uint8 pin_num,uint8* value);

Description:

used to read specific pin value

- Port_num: determine port in which pin is connected, you have to use port_ID which defined as MACROS
- Pin_num: determine pin number , you have to use PIN which defined as MACROS
- Value: the address to variable of the return reading (High, Low)
- Return : function check for the range of port_ID and PIN number Return SUCCESS if true and FAILED if out of range

EN_STATE GPIO_togglePin(uint8 port_num, uint8 pin_num);

Description:

used to toggle the output state of the pin

- Port_num: determine port in which pin is connected, you have to use port_ID which defined as MACROS
- Pin_num: determine pin number , you have to use PIN which defined as MACROS
- Return : function check for the range of port_ID and PIN number Return SUCCESS if true and FAILED if out of range

EN_STATE GPIO_setPortDirection(uint8 port_num, EN_PORT_DIRECTION direction);

Description:

used to determine port direction

- Port_num: determine port ,you have to use port_ID which defined as MACROS
- Direction: used to determine direction of the pin, you have to use Enum EN_PORT_DIRECTION (PORT_INPUT,PORT_OUTPUT)
- Return : function check for the range of port_ID Return SUCCESS if true and FAILED if out of range

Drivers Documentation Project Modules APIs

- **GPIO module APIs**

EN_STATE GPIO_writePort(uint8 port_num, uint8 value);
Description:
used to write high/low to specific port
- Port_num: determine port ,you have to use port_ID which defined as MACROS
- Value: used to determine direction of the port, you have to use Enum EN_PORT_VALUE [LOW,HIGH]
- Return : function check for the range of port_ID Return SUCCESS if true and FAILED if out of range

EN_STATE GPIO_readPort(uint8 port_num,uint8* value);
Description:
used to read the value of specific port
- Port_num: determine port ,you have to use port_ID which defined as MACROS
- Value: the address to variable of the return reading (High, Low)
- Return : function check for the range of port_ID Return SUCCESS if true and FAILED if out of range

Drivers Documentation Project Modules APIs

▪ Timer0_delay module APIs (TIMER_0.h)

```
/*============= TYPE DEFINITION =============*/
typedef struct{
   float delay;
   uint16 prescaler;
   uint8 init_value;
   float NO_OF_OV;
}ST_timer0_config;
```

Description:
the structure is used to implement delay object, to define delay variable:

```
/*============= MACRO DEFINITION =============*/
#define TCCR0            (*((volatile uint8*)0x53))
#define TCNT0            (*((volatile uint8*)0x52))
#define OCR0             (*((volatile uint8*)0x5C))
#define TIFR             (*((volatile uint8*)0x58))
#define TIMSK            (*((volatile uint8*)0x59))
//TCCR0 timer counter control register
#define CS00  0
#define CS01  1
#define CS02  2
#define WGM01 3
#define COM00 4
#define COM01 5
#define WGM00 6
#define FOC0  7
//TIMSK interrupt mask register
#define TOIE0  0
#define OCIE0  1
#define TOIE1  2
#define OCIE1B 3
#define OCIE1A 4
#define TICIE1 5
#define TOIE2  6
#define OCIE2  7
//TIFR interrupt flag register
#define TOV0  0
#define OCF0  1
#define TOV1  2
#define OCF1B 3
#define OCF1A 4
#define ICF1  5
#define TOV2  6
#define OCF2  7
```

Drivers Documentation Project Modules APIs

- **Timer0_delay module APIs** **(TIMER0_Utilities.h)**

```
/*============MACRO DEFINITION ============*/
#define max_count 256
#define min_count  1
#define init_value(T_max,T_delay,tick)  (((float)T_max-T_delay)/tick)

//pre_scaler values for TIMER0
#define N0    0
#define N1    1
#define N8    8
#define N64   64
#define N256  256
#define N1024 1024

//T_max in (ms) delay for each pre_scaler
#define Tmax_N1    0.26F
#define Tmax_N8    2.05F
#define Tmax_N64   16.38F
#define Tmax_N256  65.54F
#define Tmax_N1024 262.14F

//T_min in (ms) delay for each pre_scaler
#define Tmin_N1    0.001F
#define Tmin_N8    0.008F
#define Tmin_N64   0.064F
#define Tmin_N256  0.256F
#define Tmin_N1024 1.024F
```

Timer0_delay module APIs (TIMER_0.h)

```
/*============ FUNCTION PROTOTYPE ============*/
void Timer0_Delay(float delay);
```

Description:

- used to apply delay using polling technique
- it convert number of overflows to integer number to implement the required delay correctly
- example: if number of overflows=3.8
- mean perform 3 overflows and calculate the remaining time to complete the delay

Drivers Documentation Project Modules APIs

- **Timer0_delay module APIs** **(TIMER0_Utilities.h)**

void Timer0_event(uint16 delay,void(*g_ptr)(void));

Description:

- used to apply time out delay and run function if a period of time has passed
- Delay: delay time
- g_ptr: pointer to function which is called when time has passed

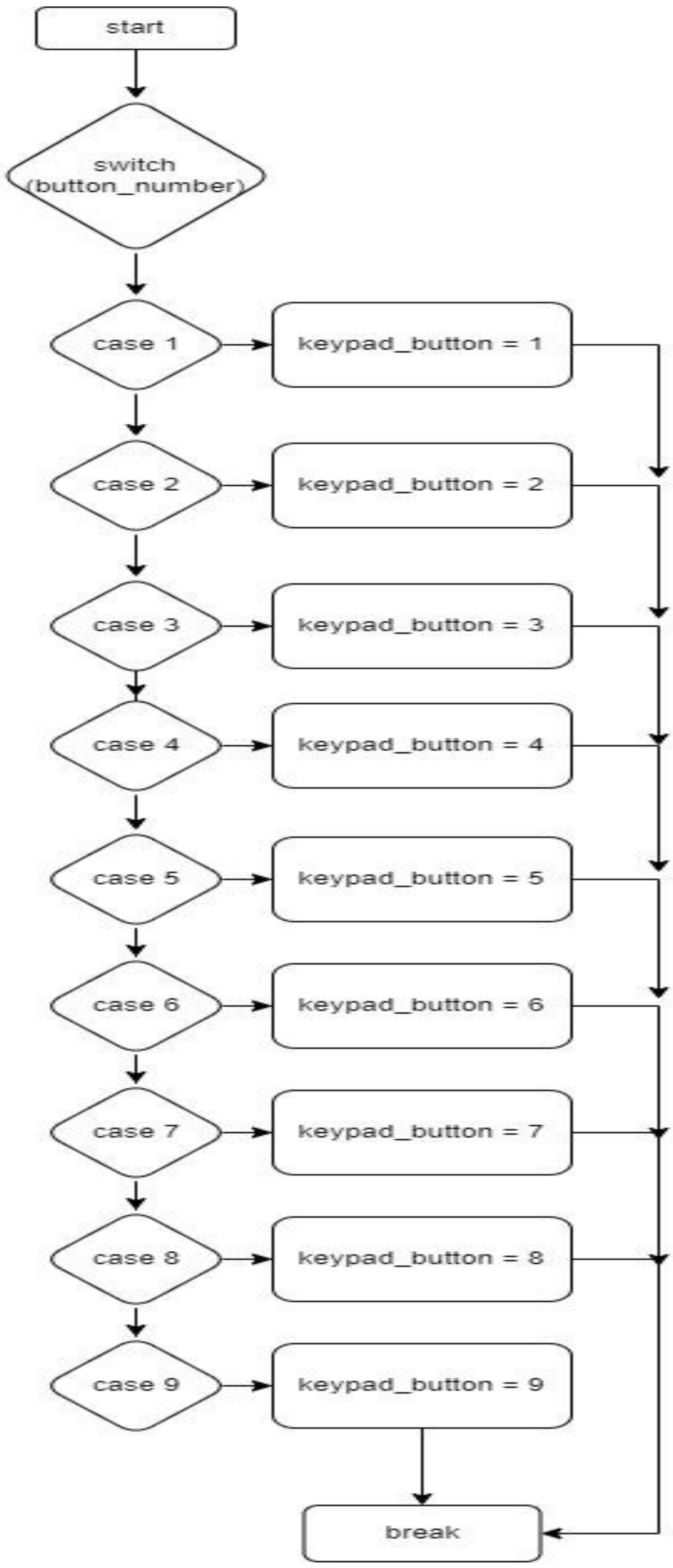
Drivers Documentation Project Modules APIs

- **motor module APIs**

```
en_MotorError_t DCM_Init(st_Motor_t *pst_a_Motor);

/**
* \brief initialize motor pins
* \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 start 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);


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

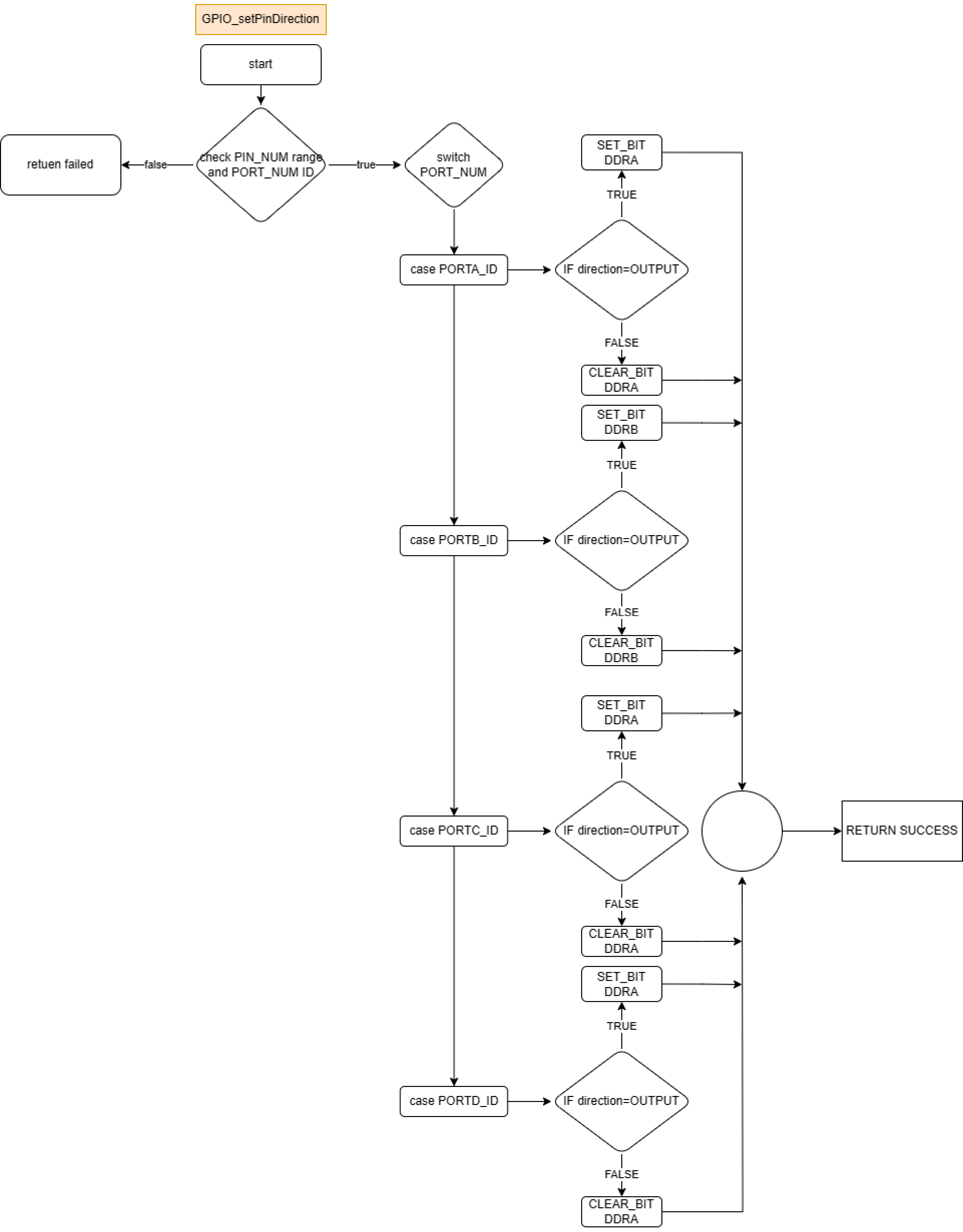
GPIO APIs flowchart

static uint8 KEYPAD_3x8_adjustKeyNumber(uint8 button_number;(



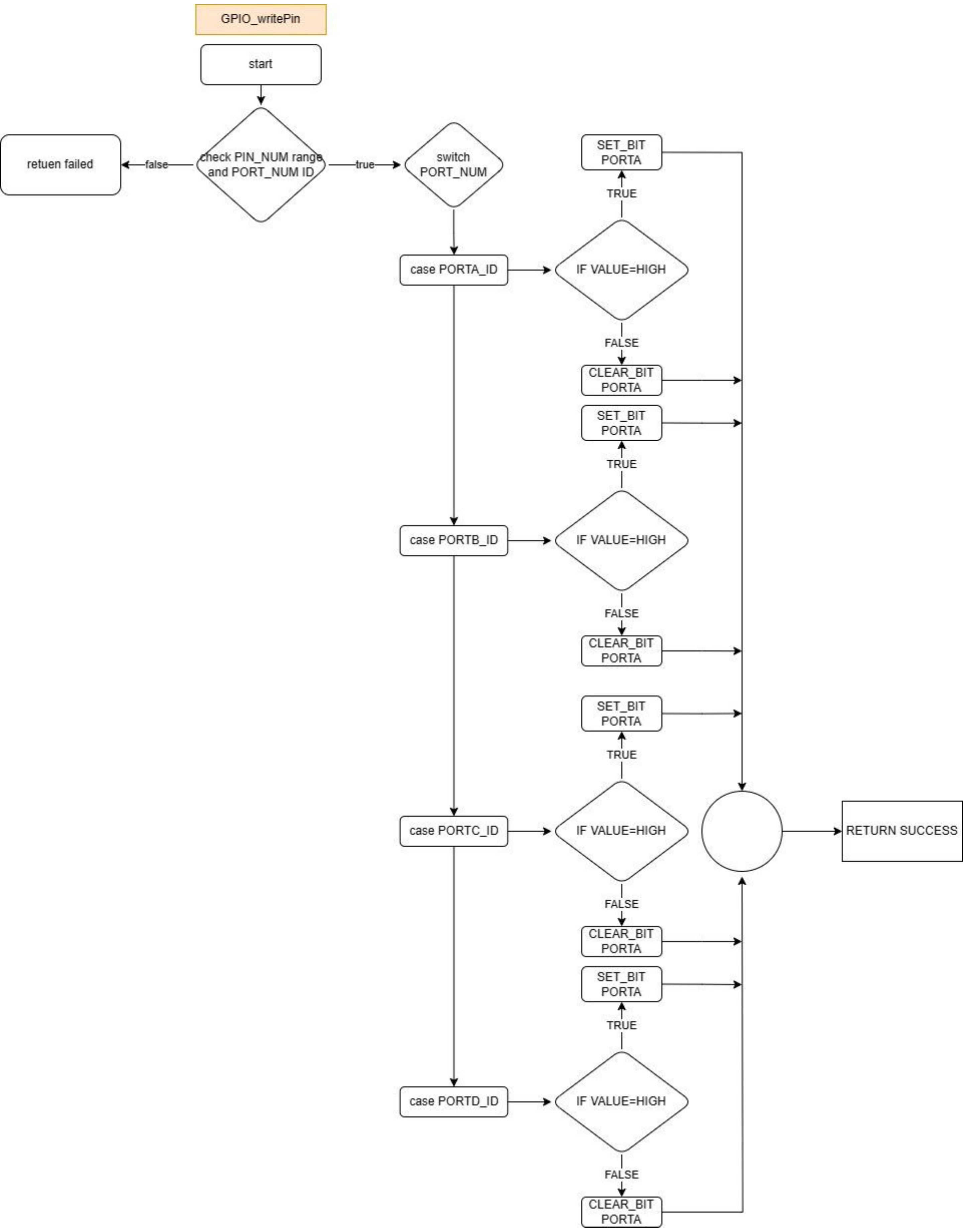
GPIO APIs flowchart

EN_STATE GPIO_setPinDirection(uint8 port_num, uint8 pin_num, EN_PIN_DIRECTION direction);



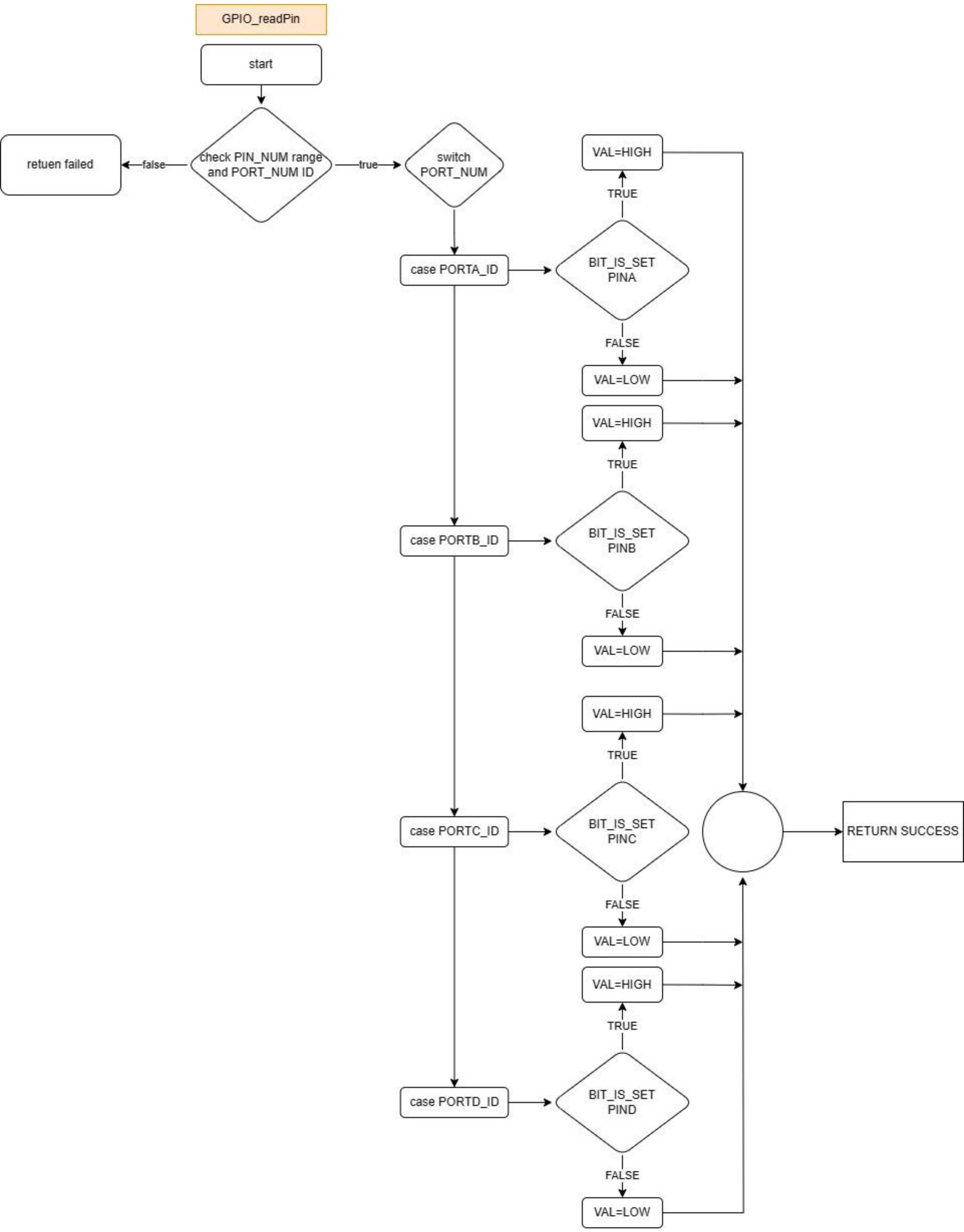
GPIO APIs flowchart

EN_STATE GPIO_writePin(uint8 port_num, uint8 pin_num, EN_PIN_VALUE value);



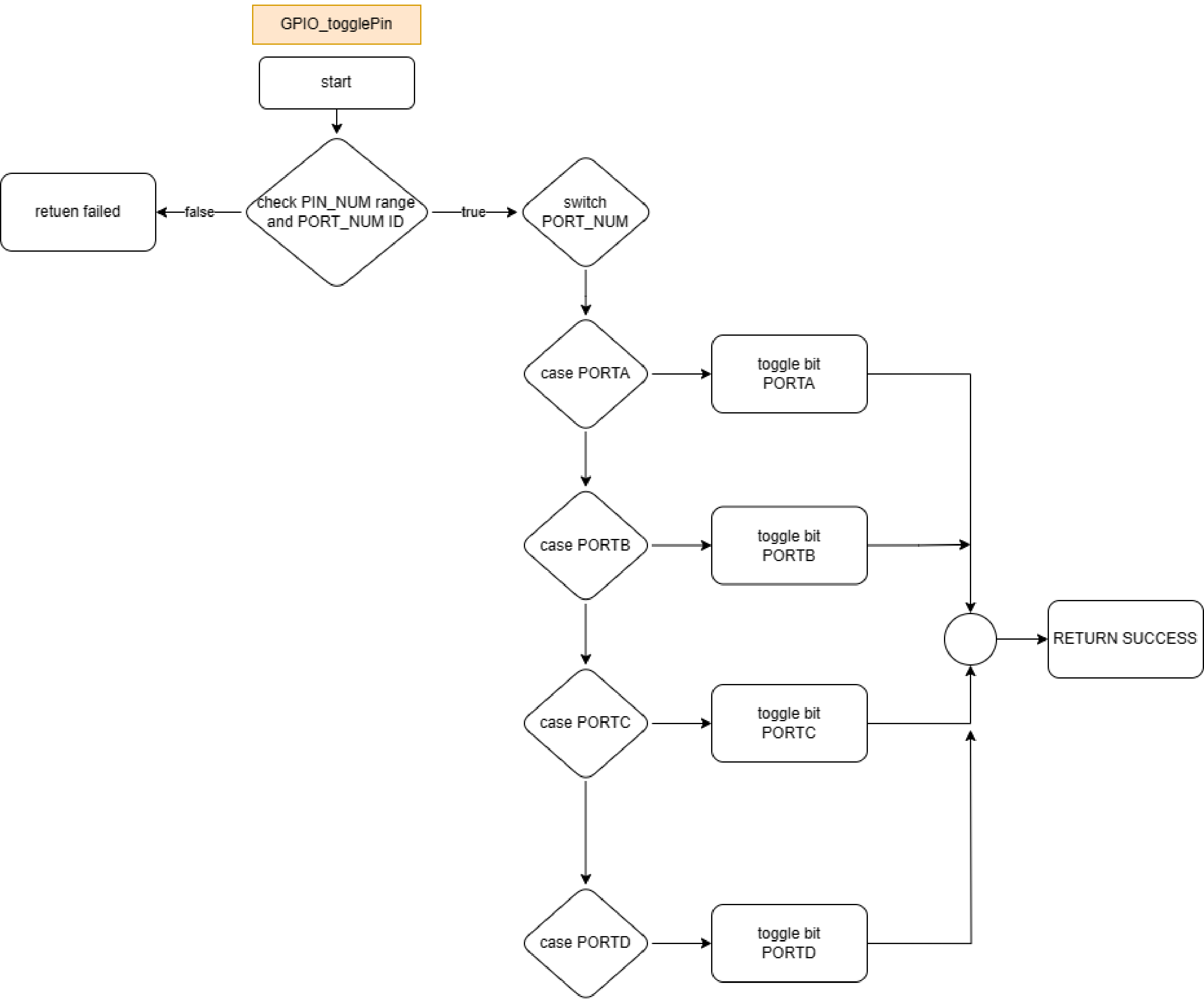
GPIO APIs flowchart

EN_STATE GPIO_readPin(uint8 port_num, uint8 pin_num,uint8* value);



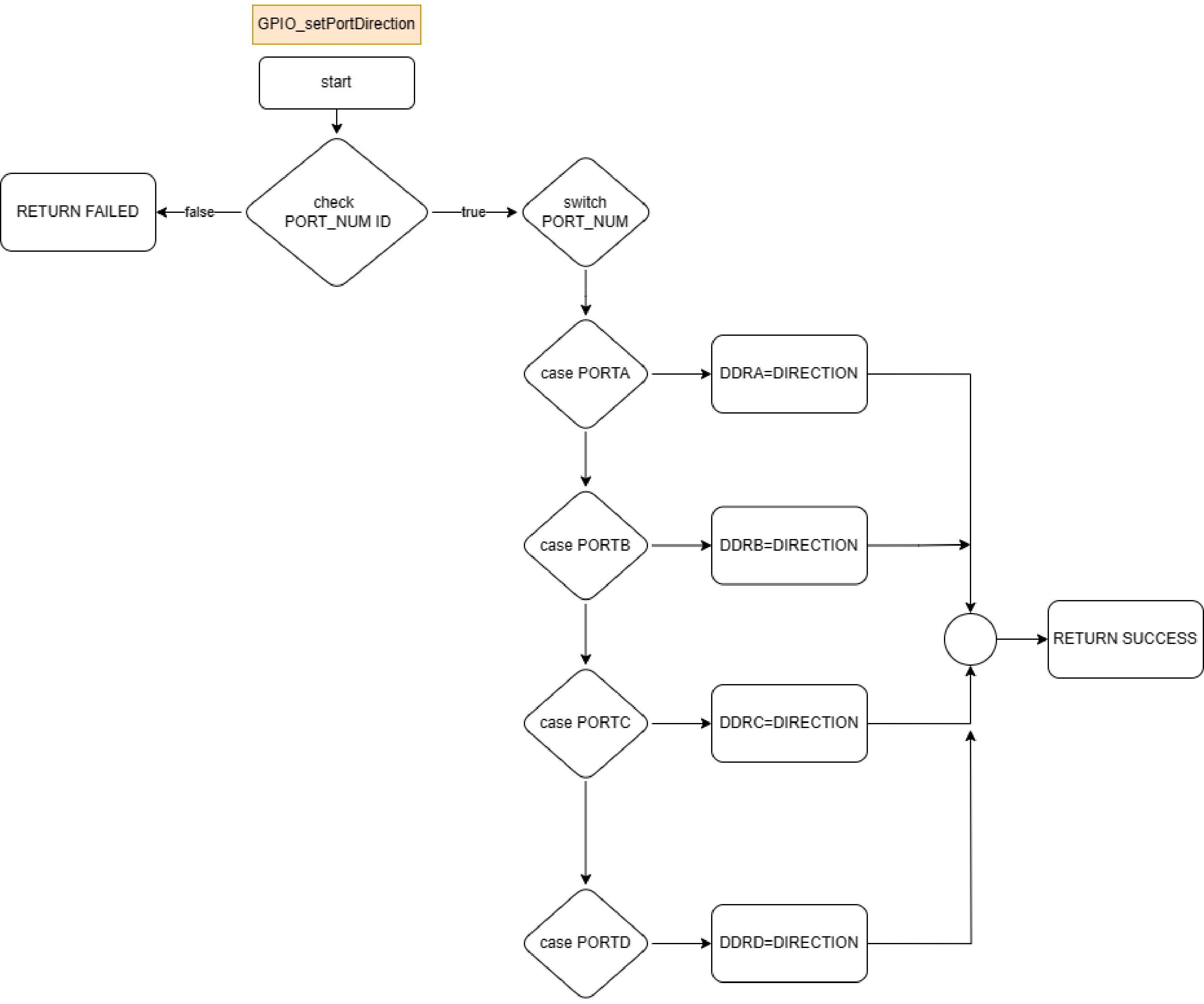
GPIO APIs flowchart

EN_STATE GPIO_togglePin(uint8 port_num, uint8 pin_num);



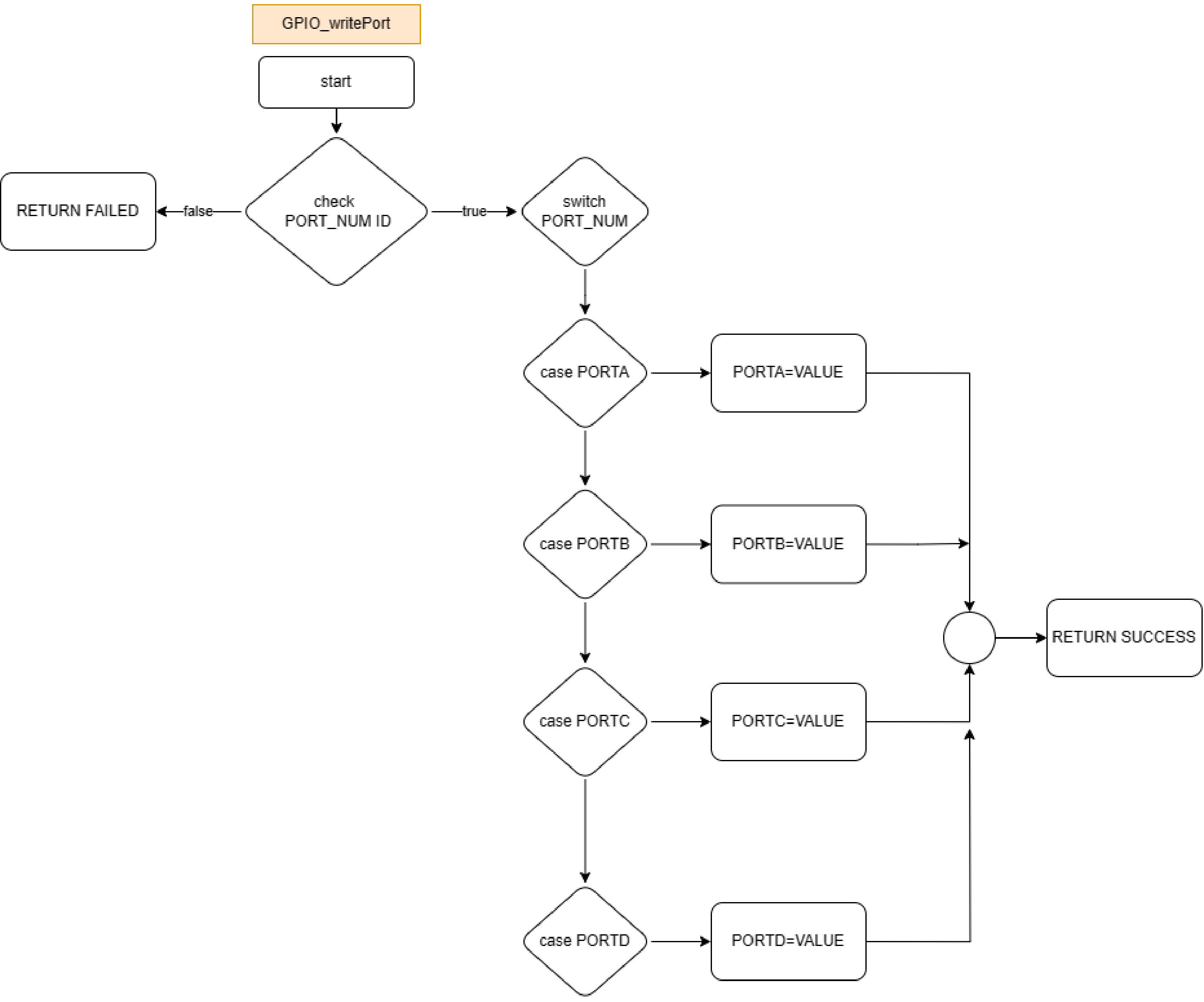
GPIO APIs flowchart

EN_STATE GPIO_setPortDirection(uint8 port_num, EN_PORT_DIRECTION direction);



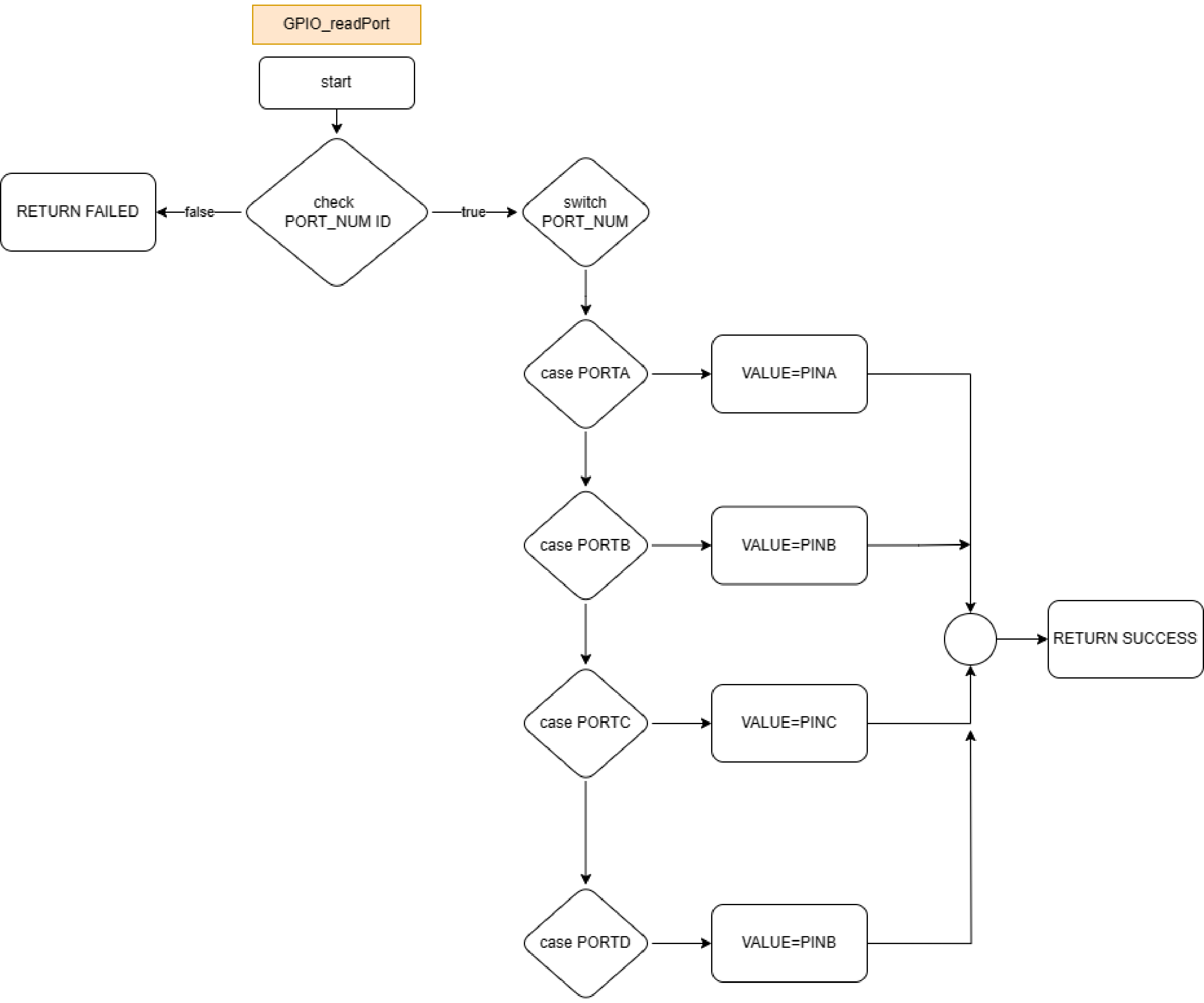
GPIO APIs flowchart

EN_STATE GPIO_writePort(uint8 port_num, uint8 value);

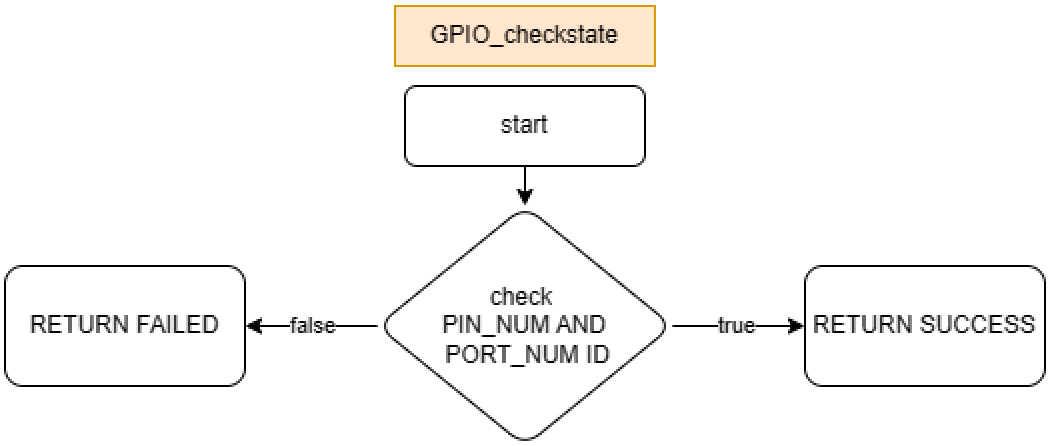


GPIO APIs flowchart

EN_STATE GPIO_readPort(uint8 port_num,uint8* value);

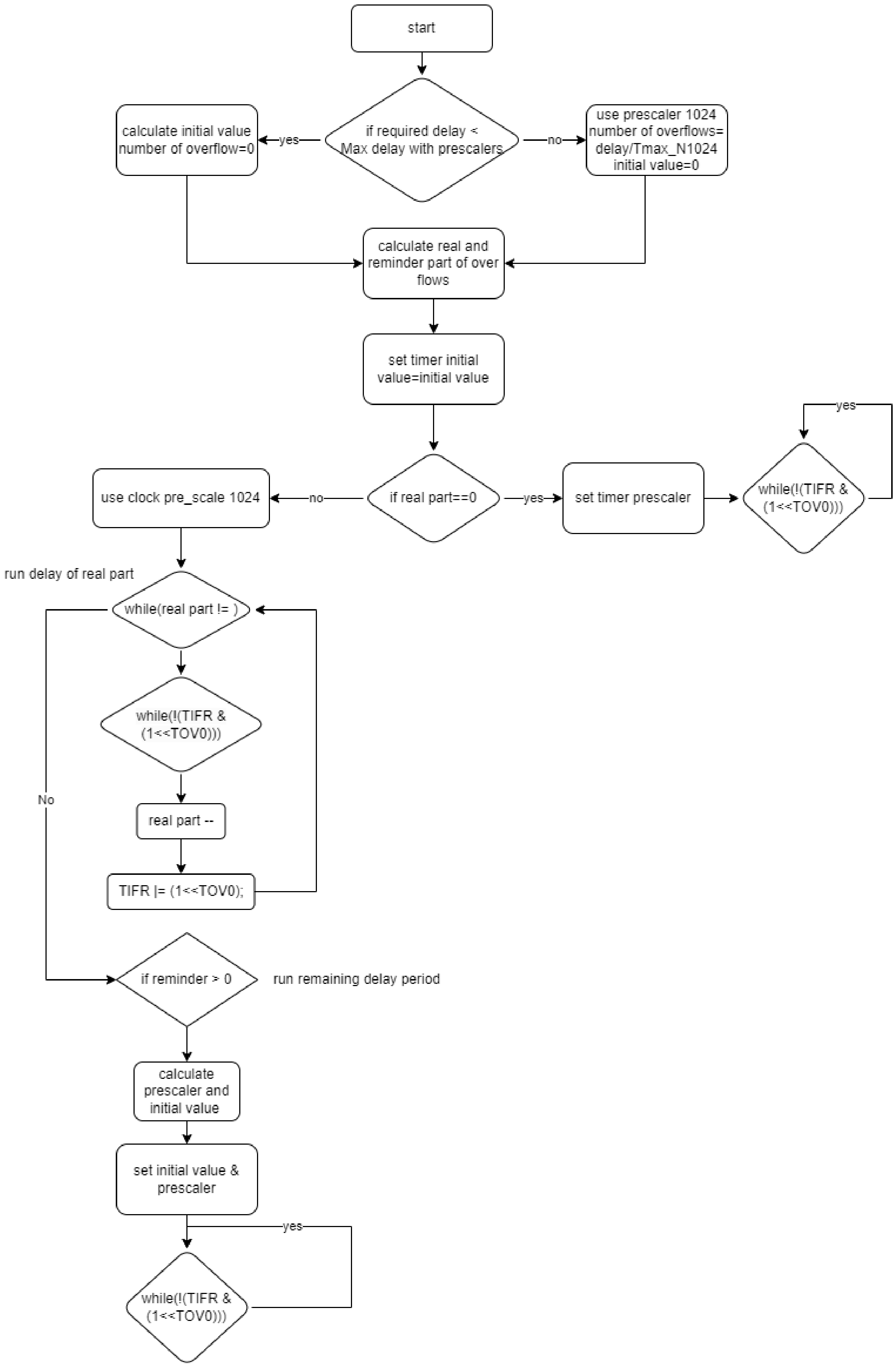


EN_STATE GPIO_checkstate(uint8 port_num,uint8 pin_num);



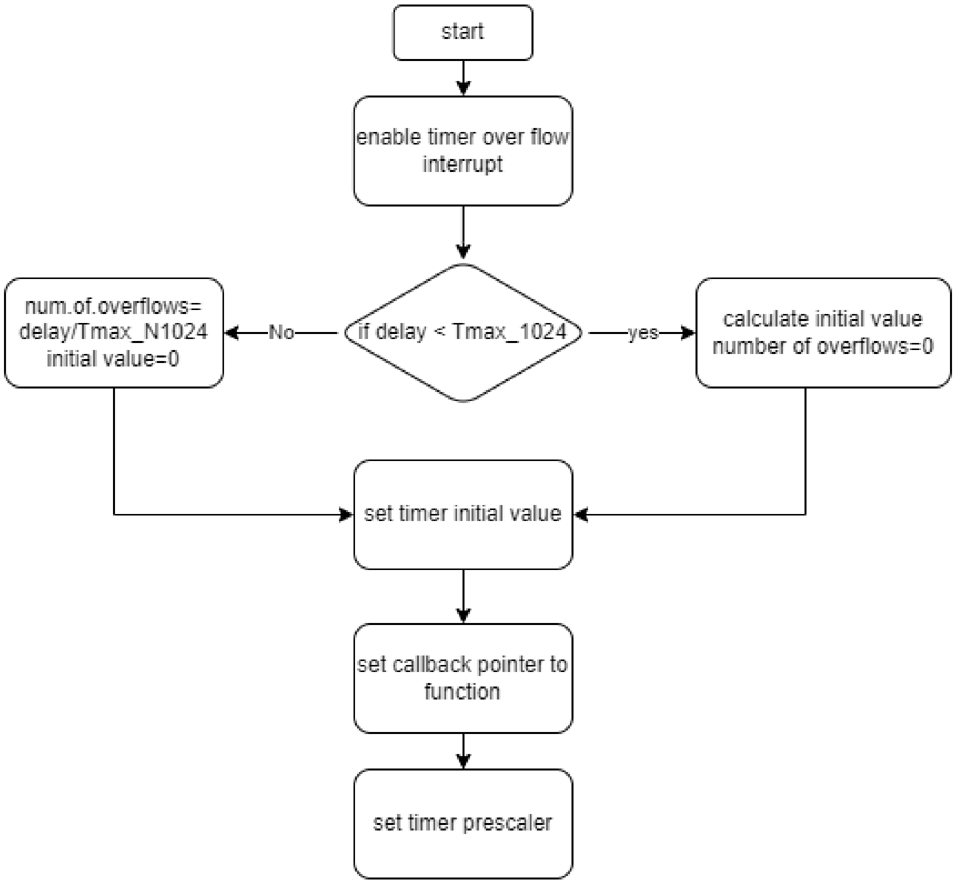
Timer0 APIs flowchart

void Timer0_Delay(float delay);


start
if required delay < Max delay with prescalers
yes
calculate initial value number of overflow=0
no
use prescaler 1024 number of overflows= delay/Tmax_N1024 initial value=0
calculate real and reminder part of over flows
set timer initial value=initial value
if real part==0
yes
set timer prescaler
while(!(TIFR & (1<<TOV0)))
yes
no
use clock pre_scale 1024
run delay of real part
while(real part !=)
while(!(TIFR & (1<<TOV0)))
real part --
TIFR |= (1<<TOV0);
No
if reminder > 0
run remaining delay period
calculate prescaler and initial value
set initial value & prescaler
yes
while(!(TIFR & (1<<TOV0)))

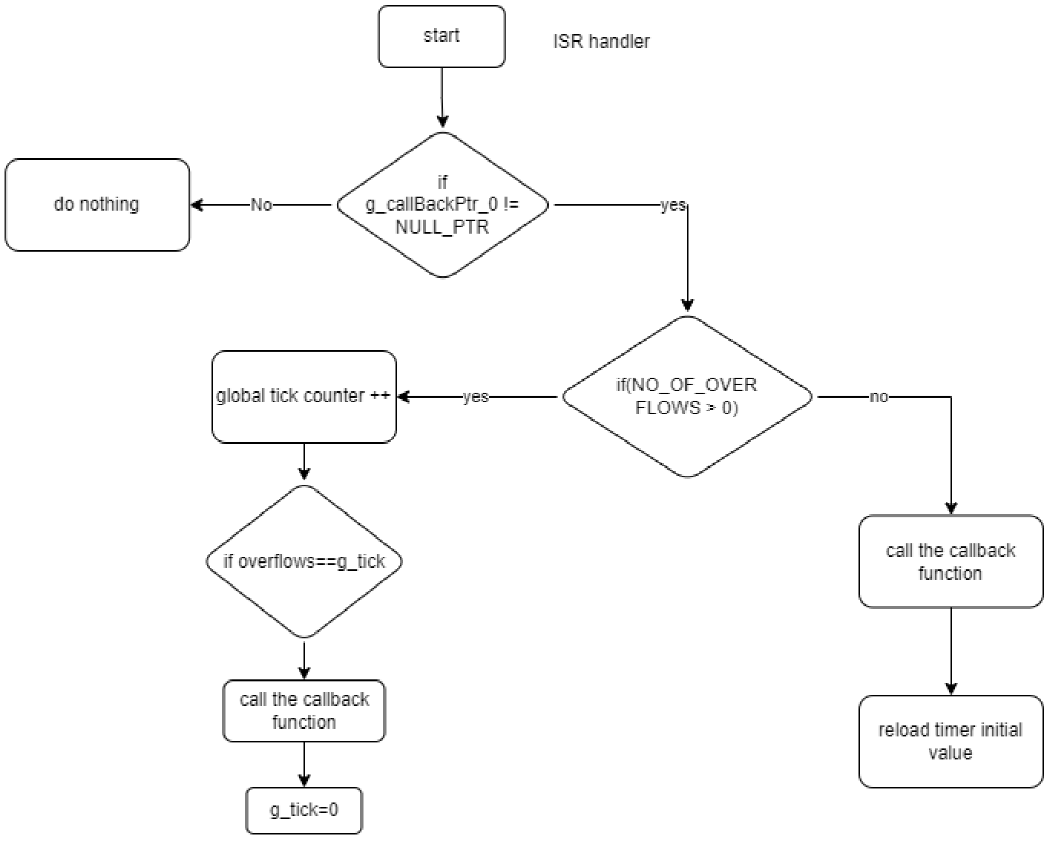
Timer0 APIs flowchart

void Timer0_event(uint16 delay,void(*g_ptr)(void));



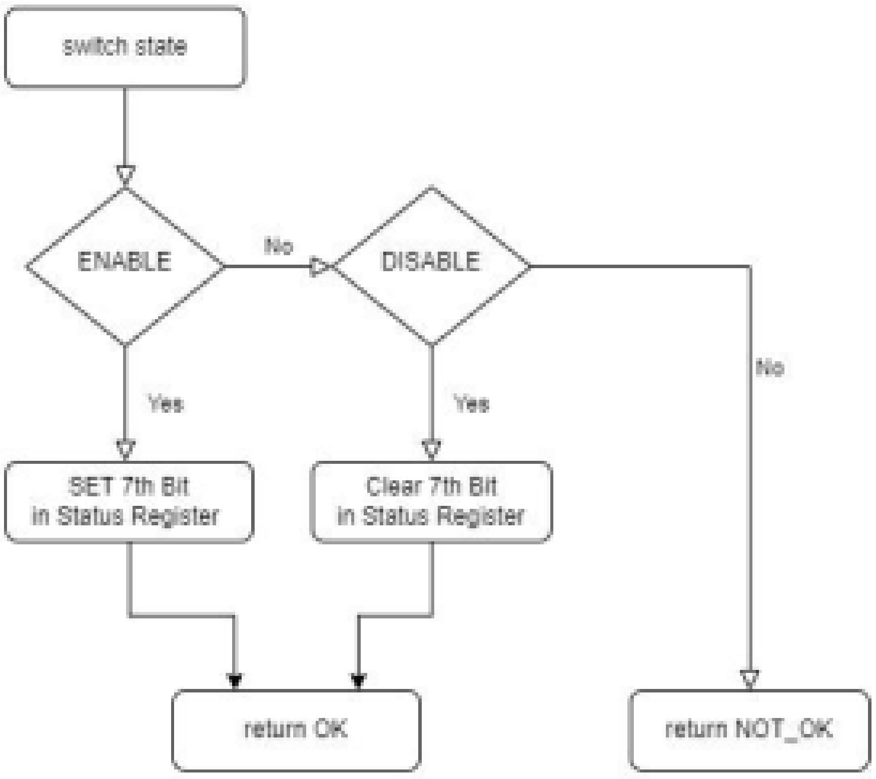
start
enable timer over flow interrupt
if delay < Tmax_1024
No
yes
num.of.overflows= delay/Tmax_N1024 initial value=0
calculate initial value number of overflows=0
set timer initial value
set callback pointer to function
set timer prescaler

ISR (TIMER0_OVF_vect)



interrupt APIs flowchart

Set global interrupt

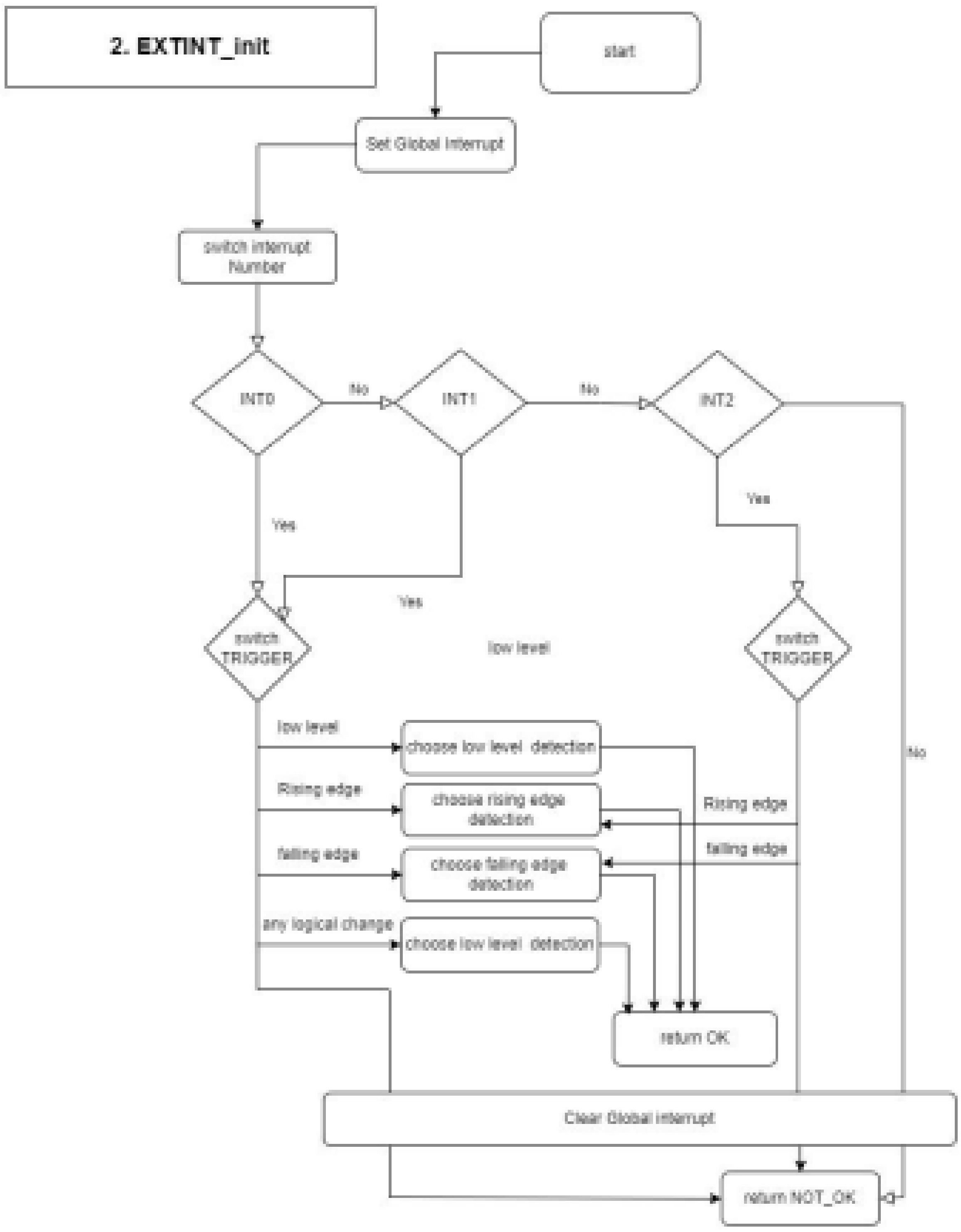

switch state
ENABLE
No
DISABLE
Yes
Yes
No
SET 7th Bit in Status Register
Clear 7th Bit in Status Register
return OK
return NOT_OK

interrupt APIs flowchart

Set global interrupt

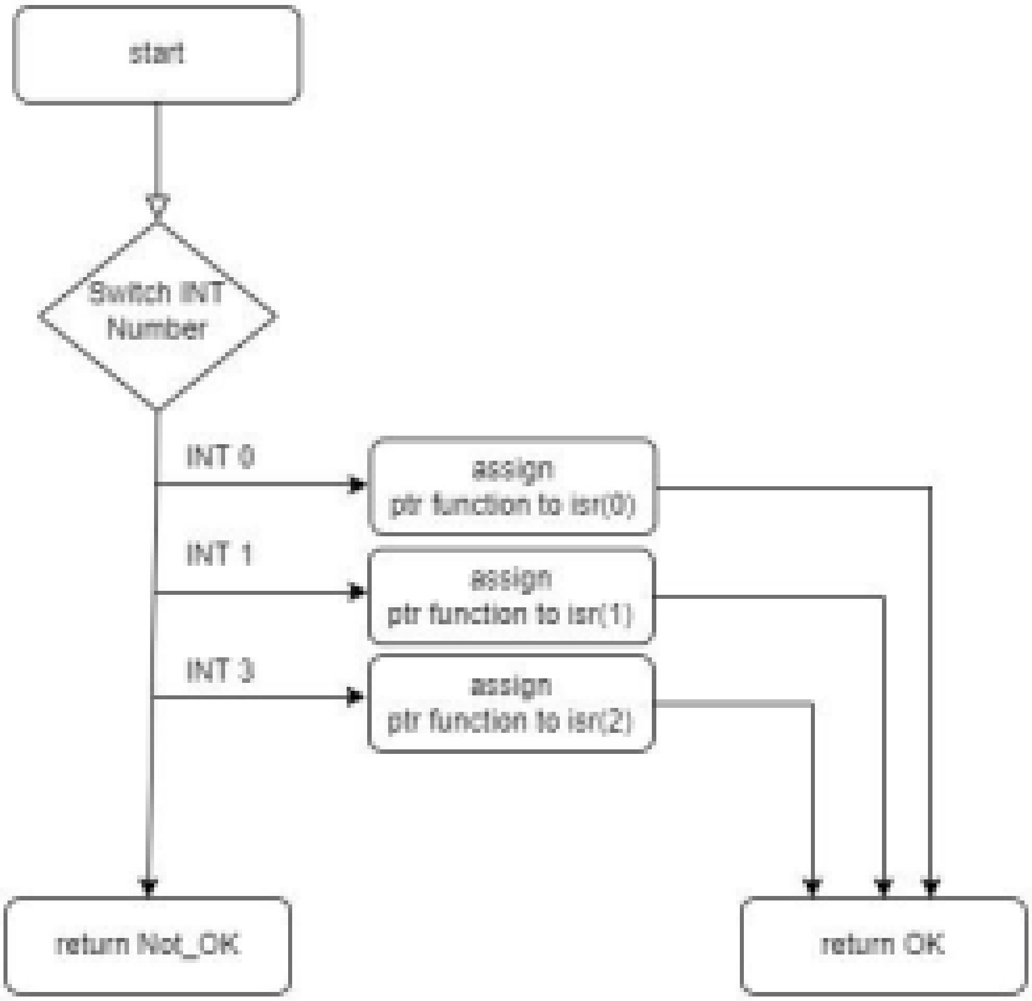
interrupt APIs flowchart

Interrupt init



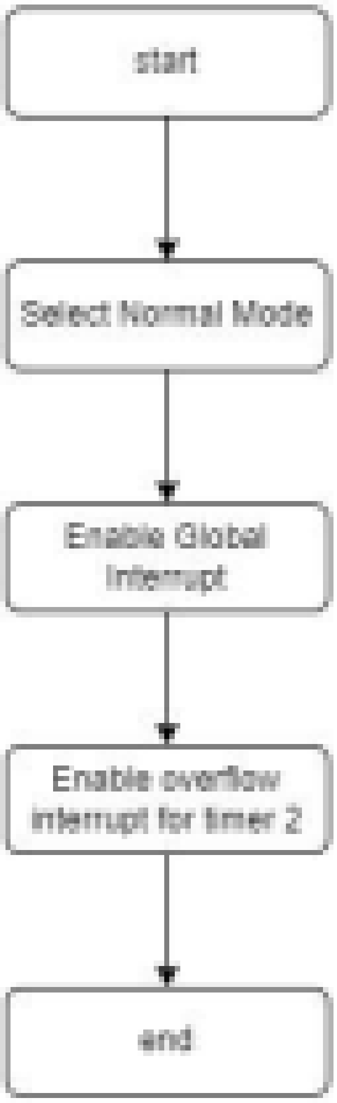
interrupt APIs flowchart

3. EXTINT_CALLBACK


start
Switch INT Number
INT 0
INT 1
INT 3
assign ptr function to isr(0)
assign ptr function to isr(1)
assign ptr function to isr(2)
return Not_OK
return OK

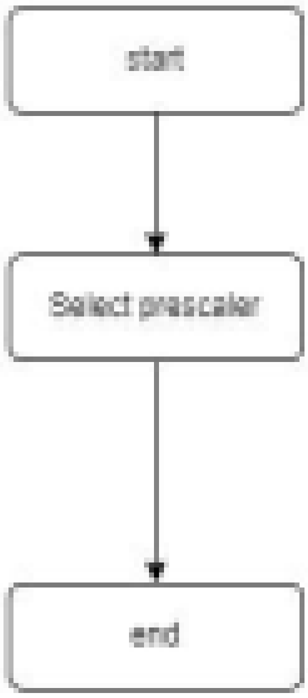
PWM APIs flowchart

timer init

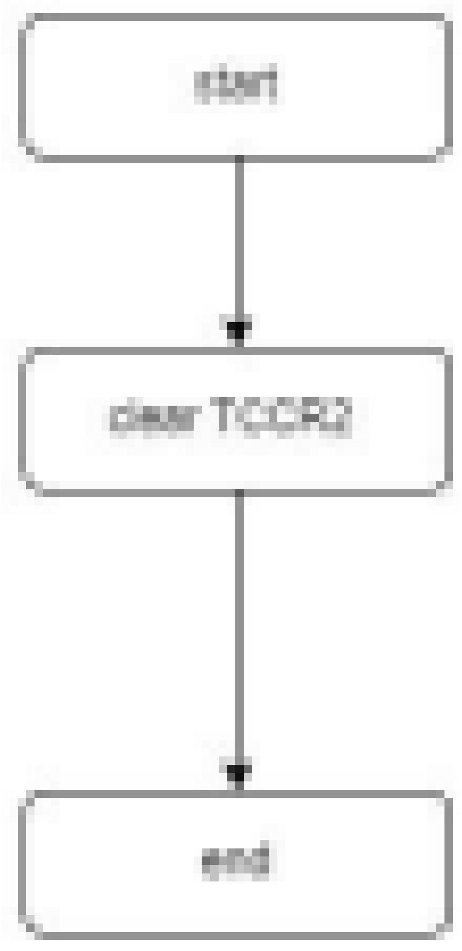


PWM APIs flowchart

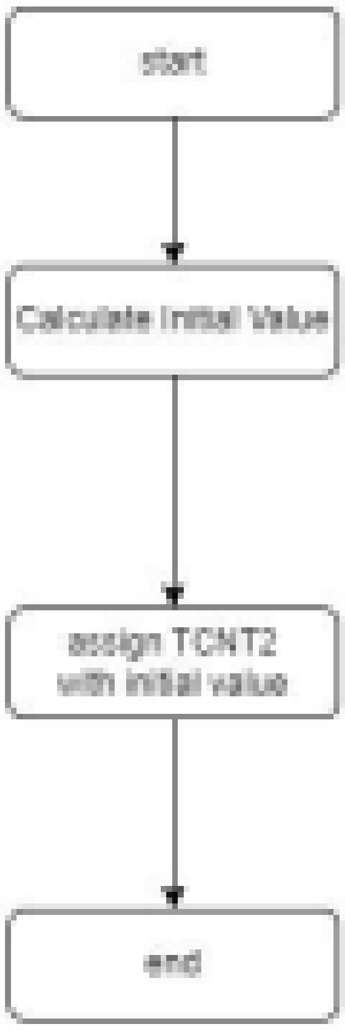
timer start



PWM APIs flowchart

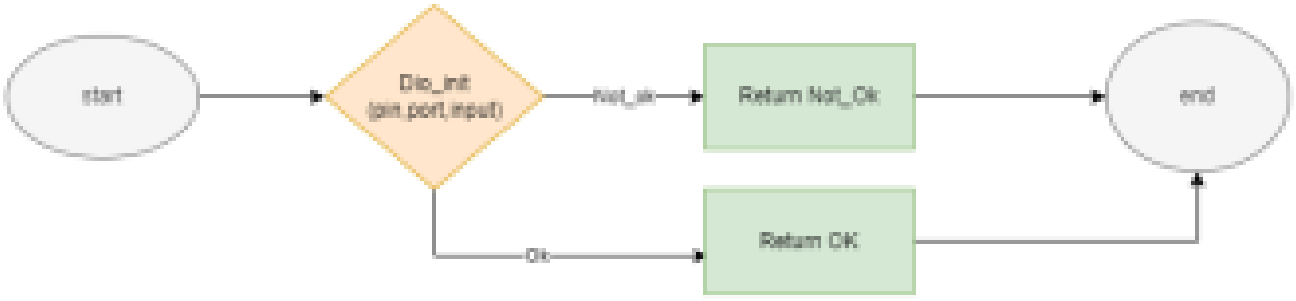


PWM APIs flowchart

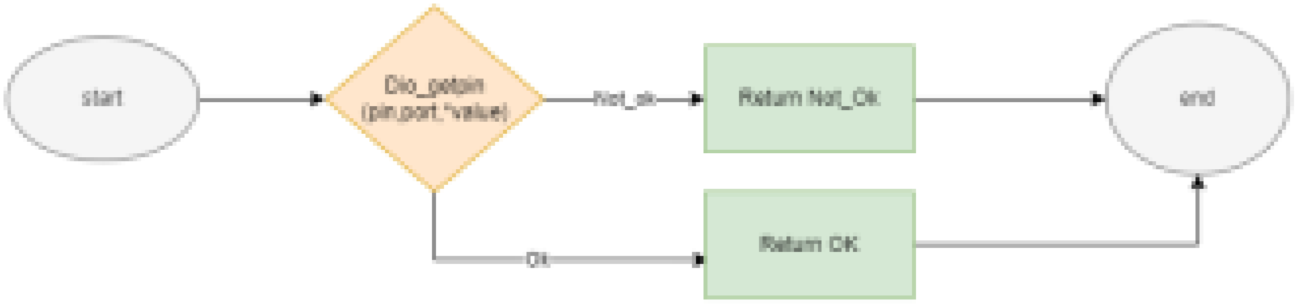


Button APIs flowchart

Button_init (pinNumber , portNumber)

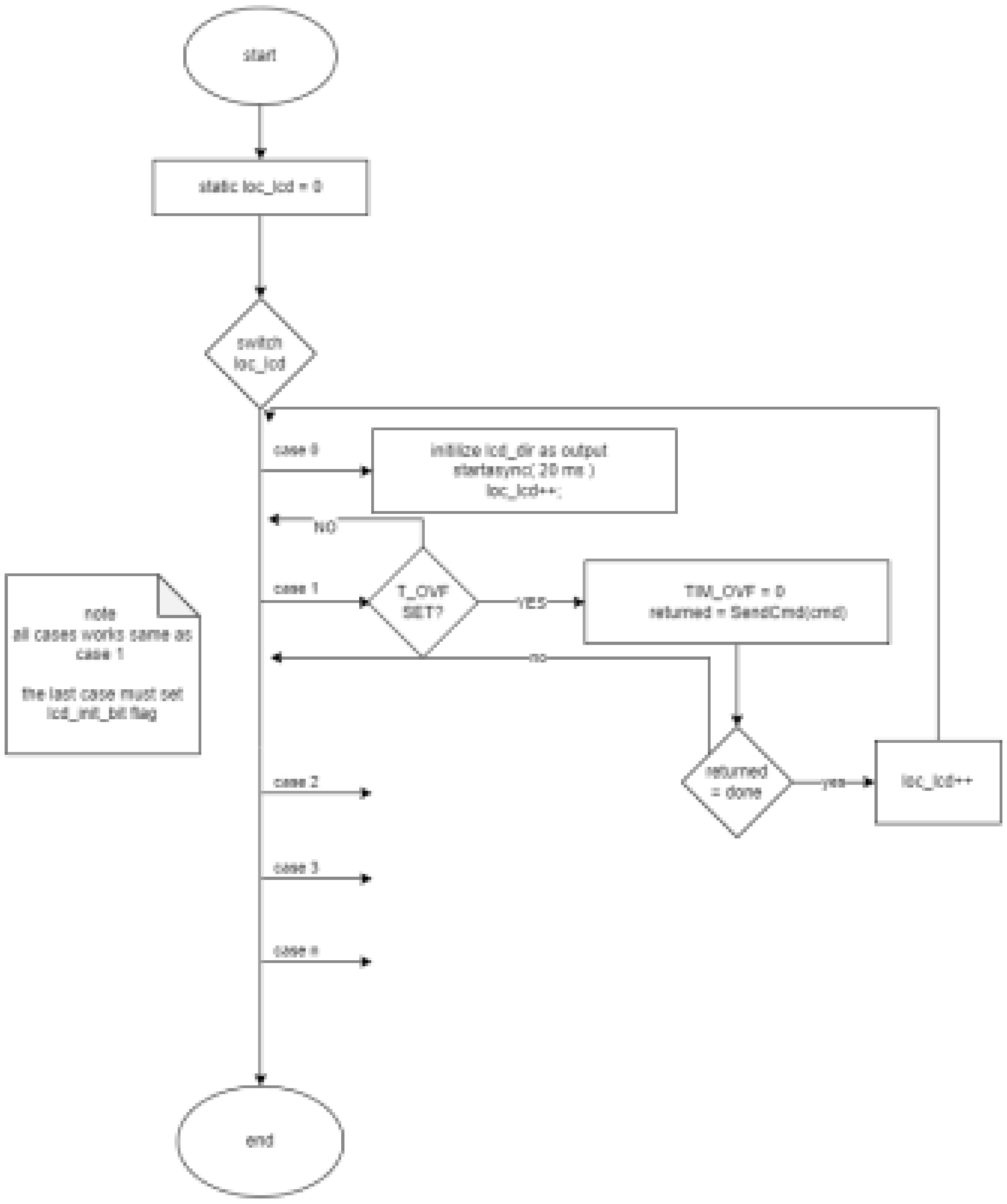


Button_read(pinNumber , portNumber, *value)

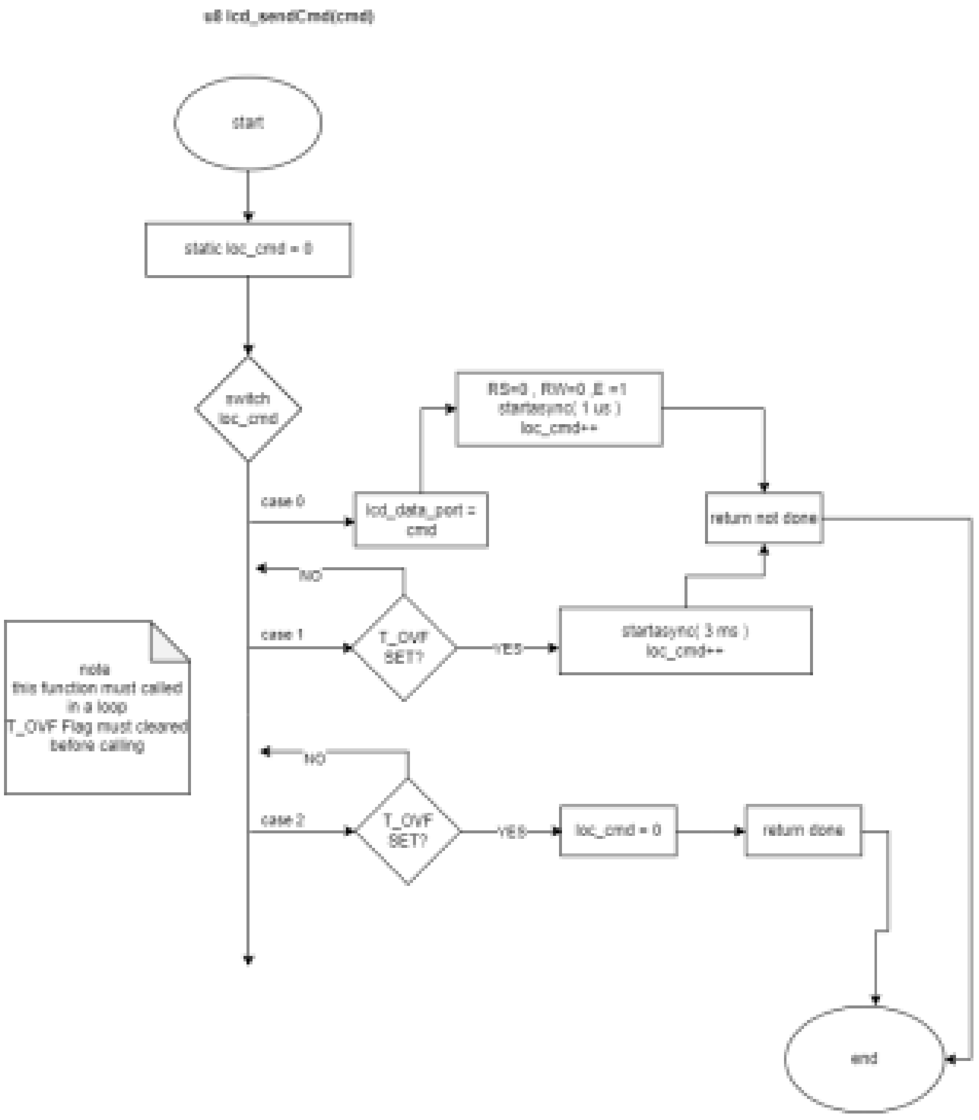


LCD APIs flowchart

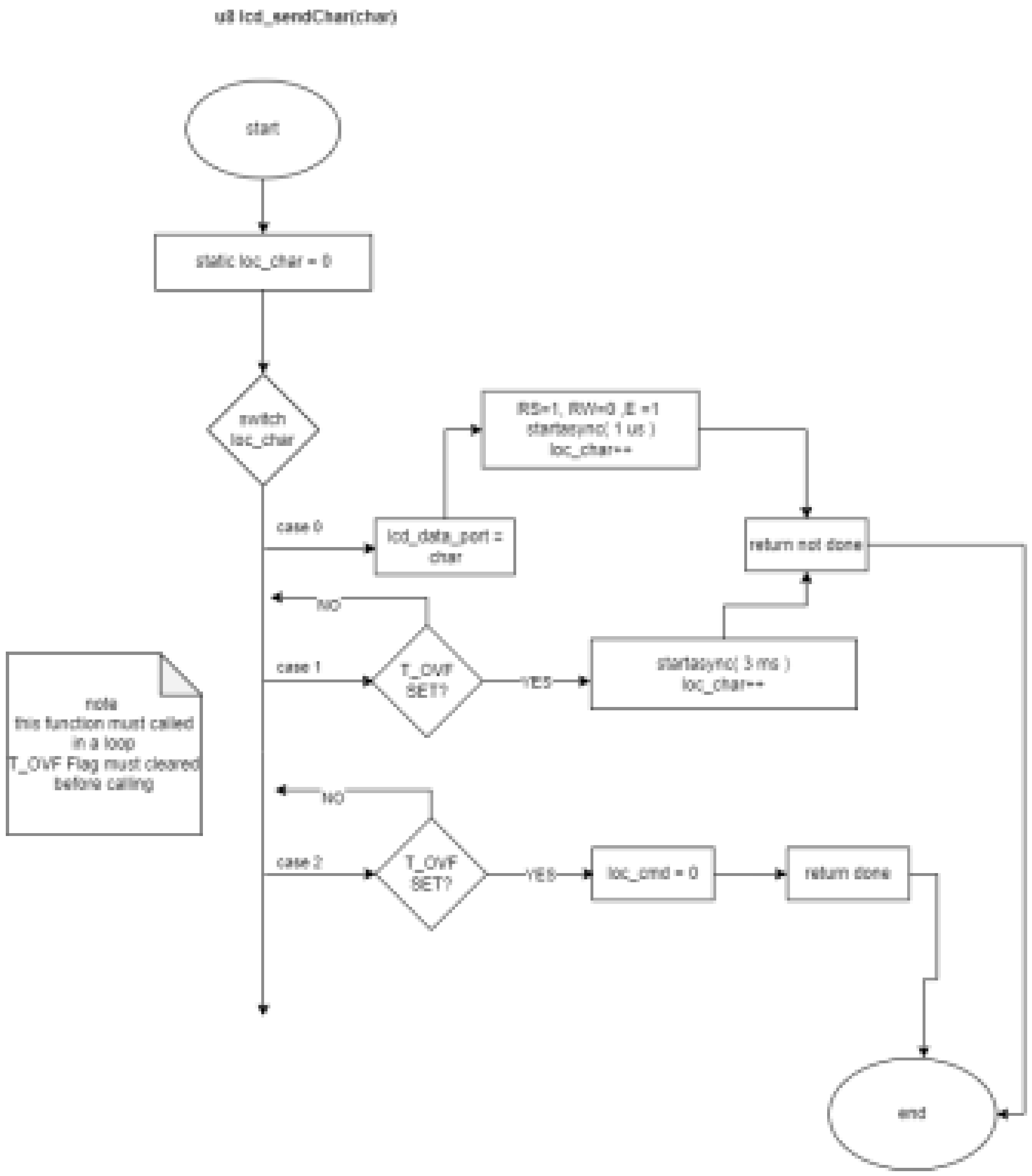
LCD_init()



LCD APIs flowchart

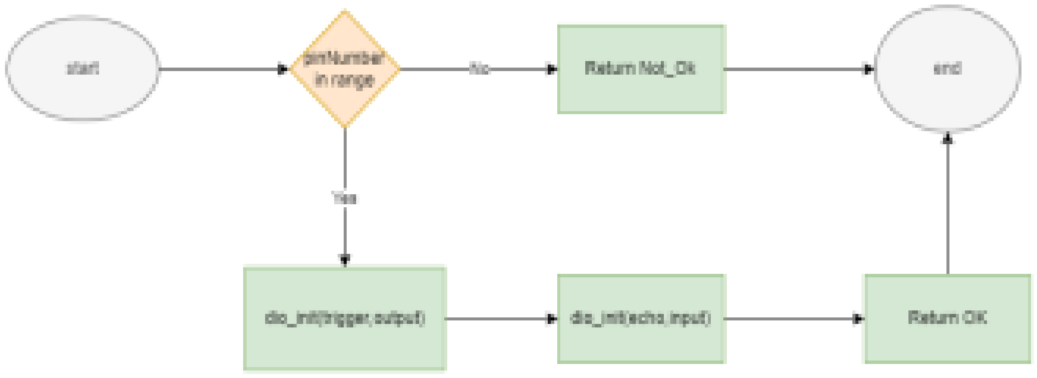


LCD APIs flowchart

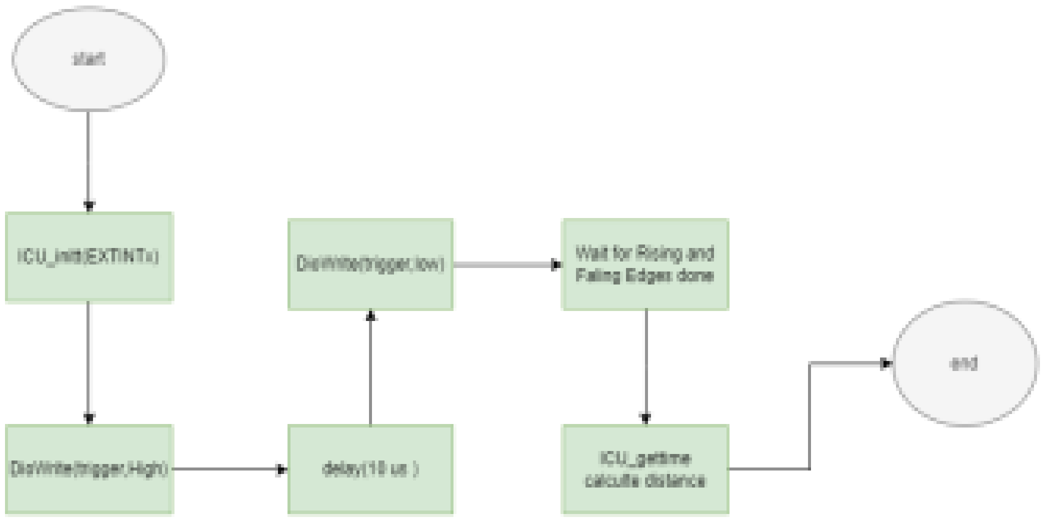


Ultrasonic APIs flowchart

US_init (echoPin , triggerPin)

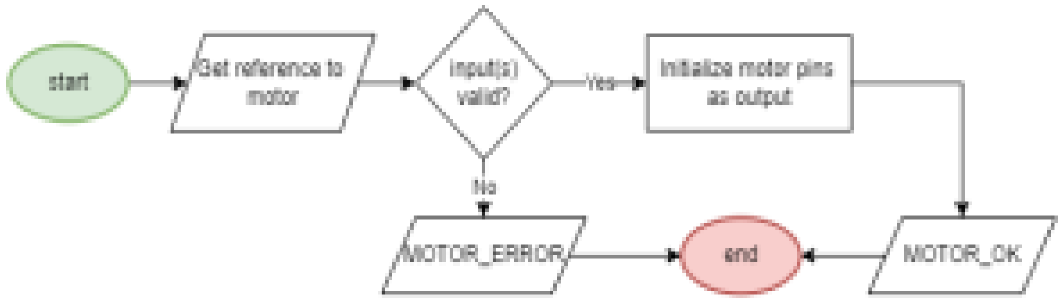


US_getDistance (triggerPin,EXTINTx,float32 *distance)

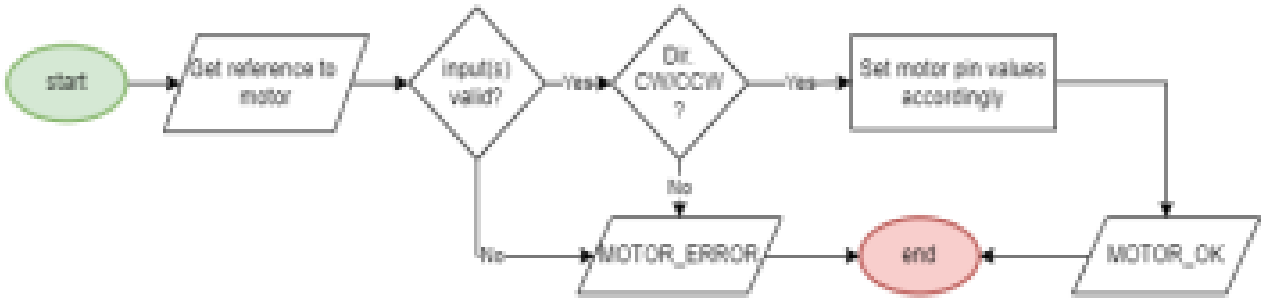


Motor APIs flowchart

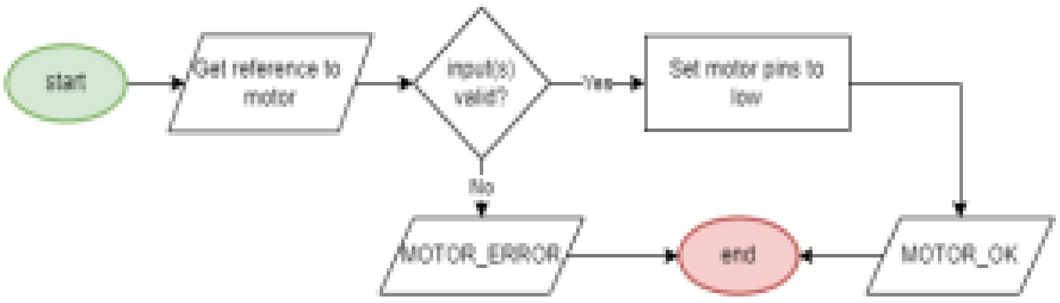
DCM_Init



DCM_Start

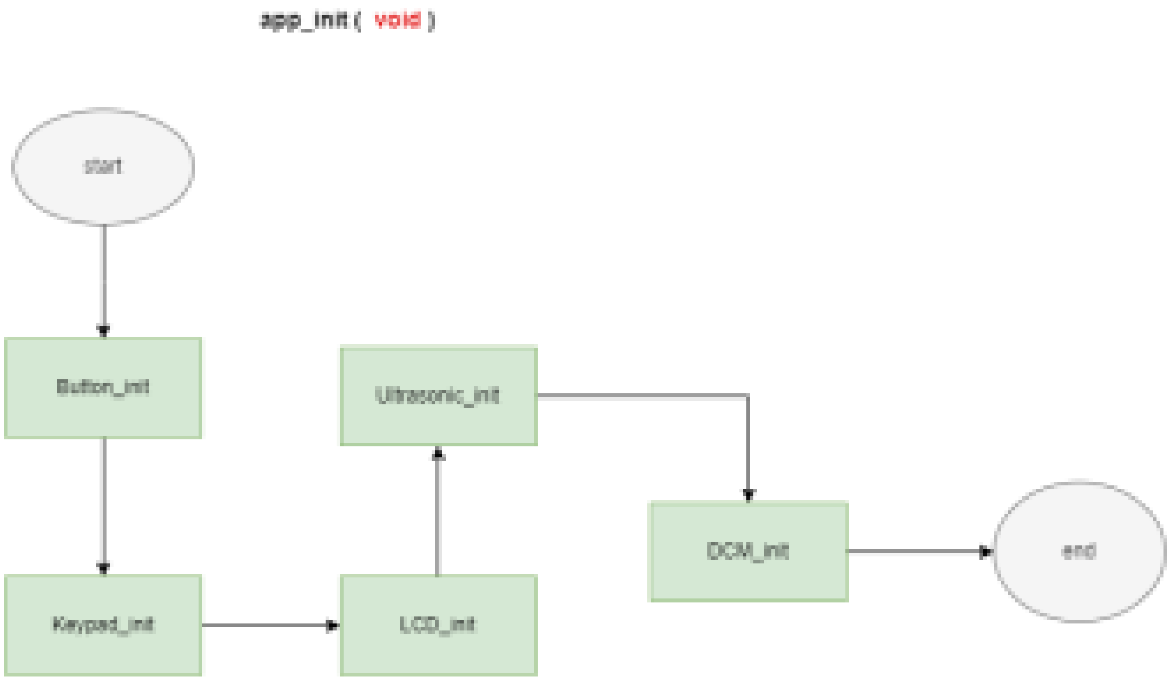


DCM_Stop



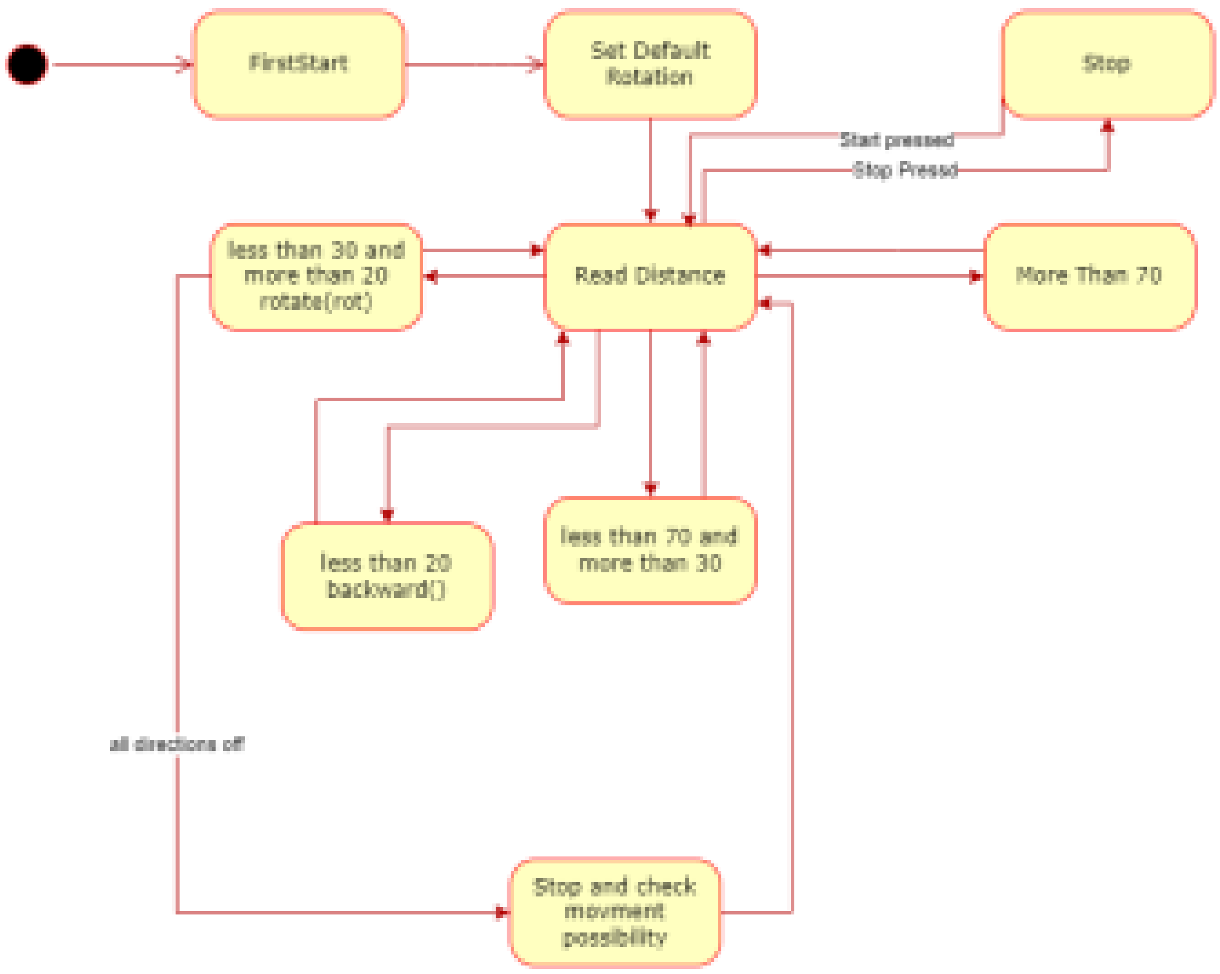
Application layer flowchart

- APP_Init



Application layer flowchart

- 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