Introduction:

This system is designed to control the temperature in a room. It includes a temperature sensor, a display, a keypad, and a buzzer (LED in Simulation). The user can set a target temperature using the keypad, and the system will turn on the buzzer if the temperature goes above the target.

System Architecture:

1. The system is implemented using a microcontroller. The following components are included in the system:

2. LM35 Temperature Sensor: This is a temperature sensor that measures the temperature of the room and outputs an analog voltage proportional to the temperature.

3. LCD Display: This is a 2x16 character LCD display that is used to show the current temperature, target temperature, and system status.

4. Keypad: This is a 3x3 keypad that is used to set the target temperature.

5. Buzzer (LED in the Simulation): This is a buzzer that is used to indicate when the temperature is above the target temperature.

6. Microcontroller: The system is implemented using a microcontroller that reads the temperature from the LM35, displays the temperature on the LCD, and controls the buzzer based on the temperature.

use cases for the system:

- Adjusting the target temperature: The user can use the keypad to adjust the target temperature. By pressing the "4" button, the user can enter the set mode, which allows them to adjust the temperature using the "1" and "2" buttons to increase or decrease the temperature, respectively. The "3" button is used to confirm the desired temperature and return to the main screen.

- Resetting the temperature: If the user wants to reset the temperature to the default value, they can press the "5" button, which triggers the reset mode. The system will display a message indicating that the temperature has been reset, and the default temperature will be set as the target temperature.

- Temperature monitoring: The system continuously monitors the temperature using the LM35 temperature sensor. If the current temperature is lower than the target temperature, the system turns off the buzzer. If the temperature exceeds the target temperature, the system turns on the buzzer to alert the user.

- Displaying the current and target temperature: The system displays the current temperature and the target temperature on the LCD screen. The user can easily check the temperature status at any time.

- Start-up message: When the system is started, it displays a welcome message on the LCD screen. The message indicates that the system is ready to use and displays the default target temperature.

- Error handling: The system handles errors caused by incorrect user inputs. If the user presses an invalid button, the system will display an error message on the LCD screen and return to the main screen.

System Functions:

AppInit(): This function initializes the system by initializing the LCD display, ADC (analog-to-digital converter), and the buzzer. It also displays the welcome message, the default temperature, and the current temperature on the LCD.

AppStart(): This function is called repeatedly in a loop. It reads the temperature from the LM35, updates the LCD display with the current temperature, target temperature, and system status (whether the buzzer is on or off), and checks for keypad inputs to set or reset the target temperature.

setMode(): This function is called when the user wants to set the target temperature. It displays the minimum, maximum, and current target temperature on the LCD and waits for the user to input a new

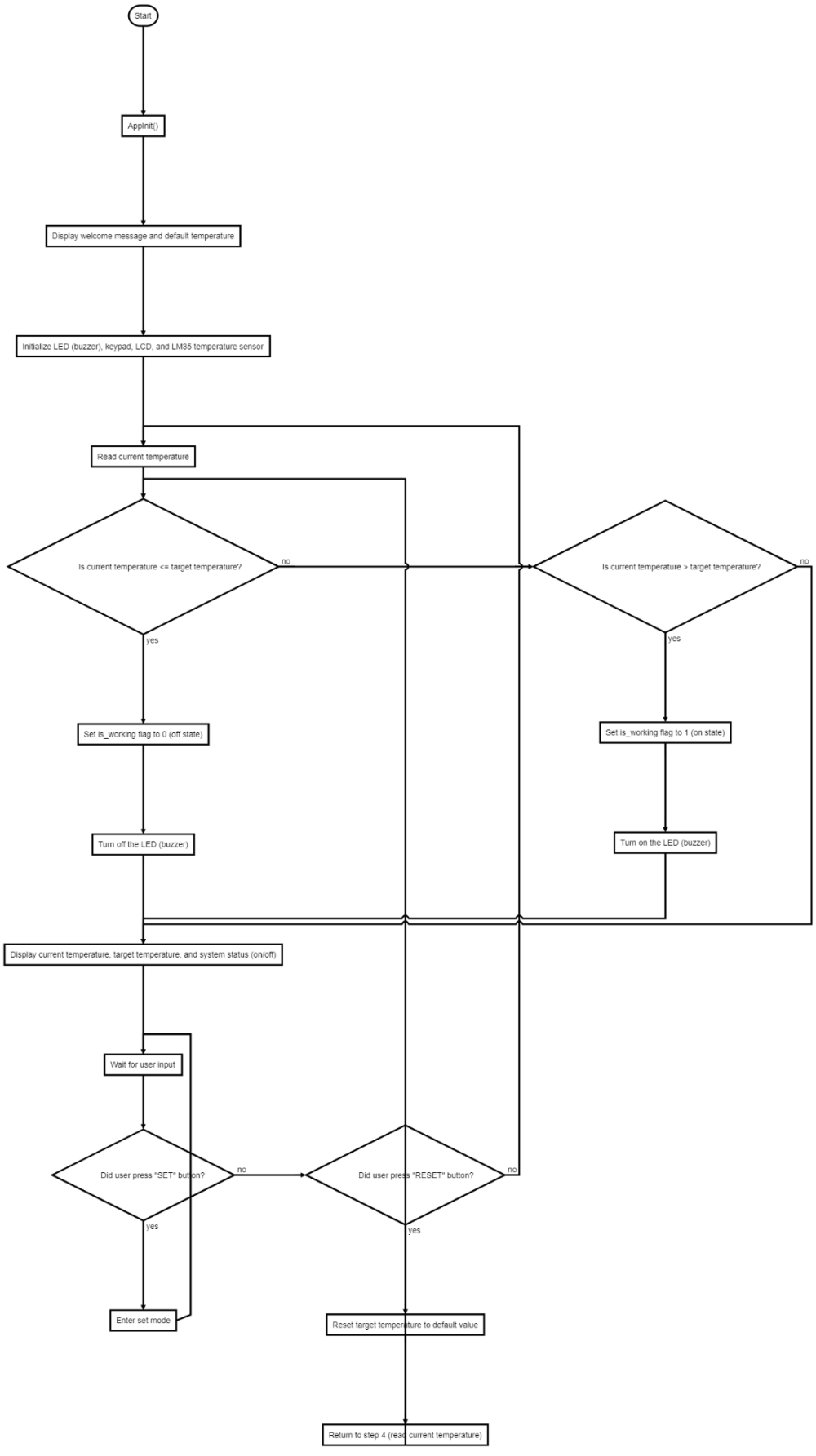
target temperature using the keypad. If the user inputs a new target temperature, the function returns and the new target temperature is set.

resetMode(): This function resets the target temperature to the default temperature and displays a message on the LCD indicating that the temperature has been reset.

buzzer_digitalwrite(): This function is used to turn on or off the buzzer (LED in the original description) based on the current temperature and target temperature. If the current temperature is above the target temperature, the buzzer is turned on. Otherwise, the buzzer is turned off.

Global Variables:

- current_temp: This variable holds the current temperature read from the LM35.
- target_temp: This variable holds the target temperature set by the user.
- buffer: This variable is used to temporarily store the new target temperature input by the user through the keypad.
- is_working: This variable is used to indicate whether the buzzer (LED in the original description) should be turned on or off based on the current temperature and target temperature.


Start
AppInit()
Display welcome message and default temperature
Initialize LED (buzzer), keypad, LCD, and LM35 temperature sensor
Read current temperature
Is current temperature <= target temperature?
no
yes
Set is_working flag to 0 (off state)
Turn off the LED (buzzer)
Is current temperature > target temperature?
no
yes
Set is_working flag to 1 (on state)
Turn on the LED (buzzer)
Display current temperature, target temperature, and system status (on/off)
Wait for user input
Did user press "SET" button?
no
yes
Enter set mode
Did user press "RESET" button?
no
yes
Reset target temperature to default value
Return to step 4 (read current temperature)

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**

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**

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**

Project Modules APIs

- Keypad driver APIs

This is a keypad static configuration, before using driver you have to configure the macros according to keypad specifications and hardware connections

```
/*====Keypad configurations for number of rows and columns==== */
#define KEYPAD_NUM_COLS                 3
#define KEYPAD_NUM_ROWS                 3

/*=============  Keypad Port Configurations =============*/
#define KEYPAD_PORT_ID                      PORTB_ID
#define KEYPAD_FIRST_ROW_PIN_ID             PIN3
#define KEYPAD_FIRST_COLUMN_PIN_ID          PIN0

/*===========  Keypad button logic configurations ===========  */
#define KEYPAD_BUTTON_PRESSED           LOGIC_LOW

/*=============  FUNCTION PROTOTYPE=============*/
uint8 KEYPAD_getPressedKey(void);
```

Description:

Used to get the pressed key from the keypad

```
static uint8 KEYPAD_3x3_adjustKeyNumber(uint8 button_number);
```

Description:

Before code compilation, modify the function to determine keypad map for each key location

```
static uint8 KEYPAD_3x3_adjustKeyNumber(uint8 button_number)
{
    uint8 keypad_button = 0;
    switch(button_number)
    {
        case 1: keypad_button = 1;
                break;
        case 2: keypad_button = 2;
                break;
        case 3: keypad_button = 3;
                break;
        case 4: keypad_button = 4;
                break;
        case 5: keypad_button = 5;
                break;
        case 6: keypad_button = 6;
                break;
        case 7: keypad_button = 7;
                break;
        case 8: keypad_button = 8;
                break;
        case 9: keypad_button = 9;
                break;
        default: keypad_button = button_number;
                break;
    }
    return keypad_button;
```

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
```

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

Project Modules APIs

- Timer0 delay module APIs (TIMER0_Utilities.h)

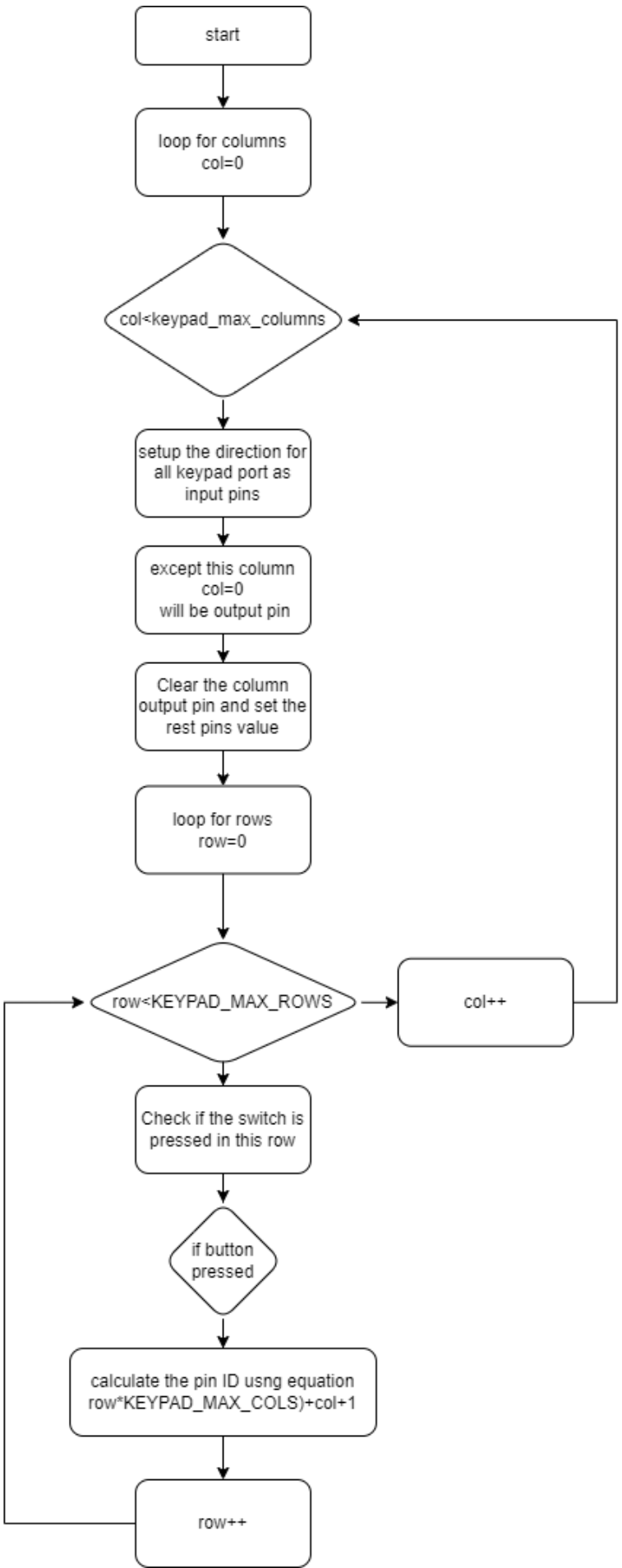
void Timer0_event(uint16 delay,void(*g_ptr)(void));

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

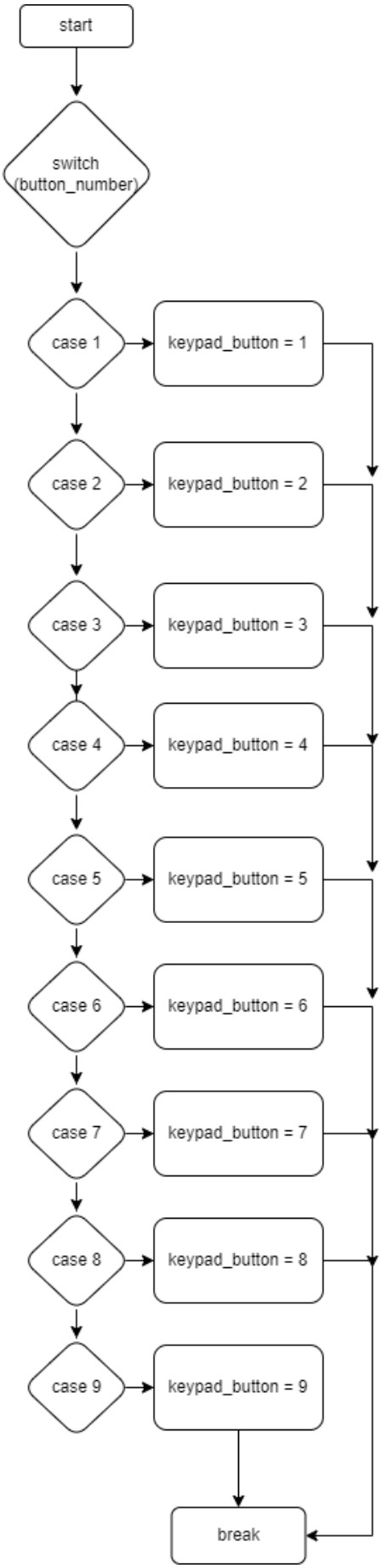
Keypad APIs flowchart

uint8 KEYPAD_getPressedKey(void);



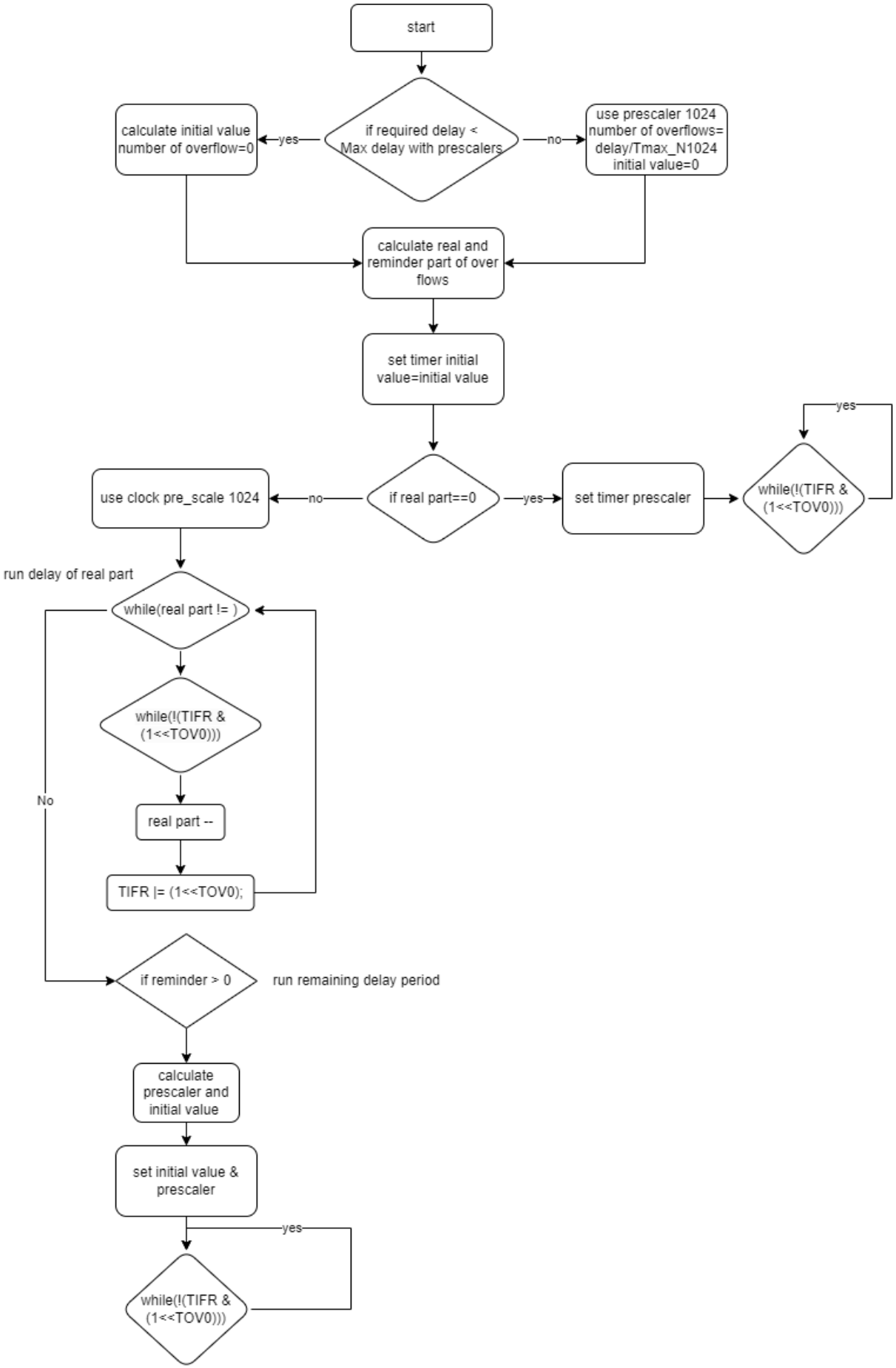
Keypad APIs flowchart

static uint8 KEYPAD_3x3_adjustKeyNumber(uint8 button_number);



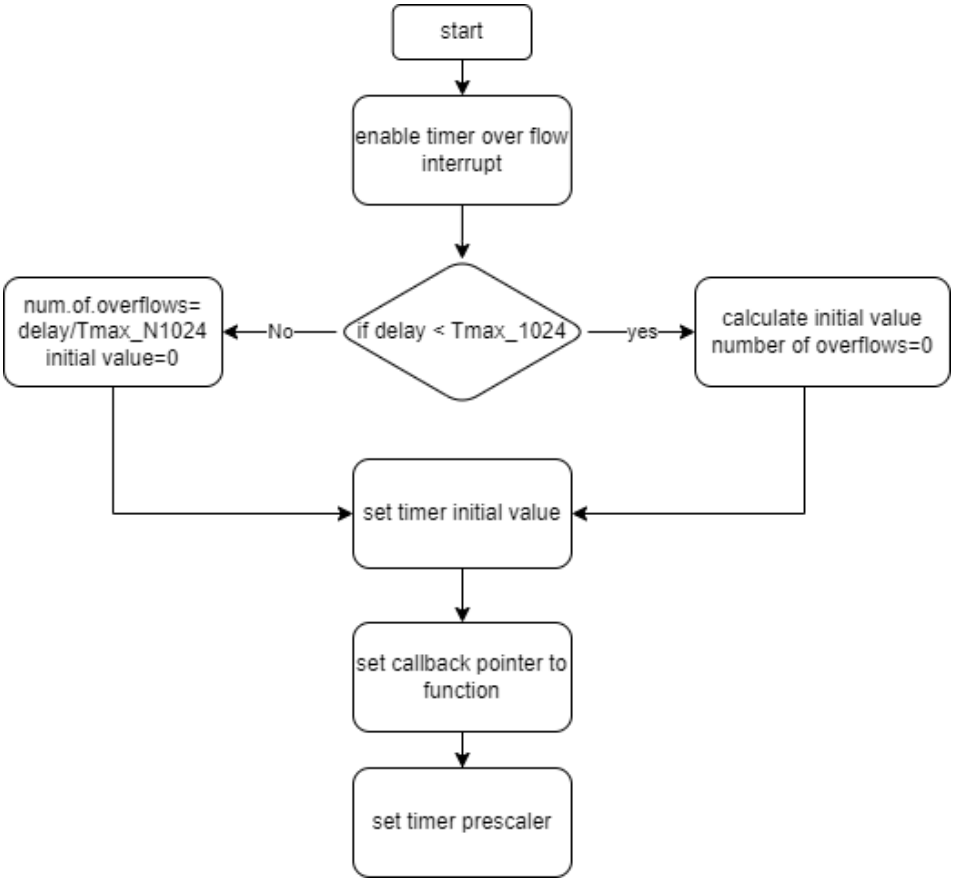
Timer0 APIs flowchart

void Timer0_Delay(float delay);

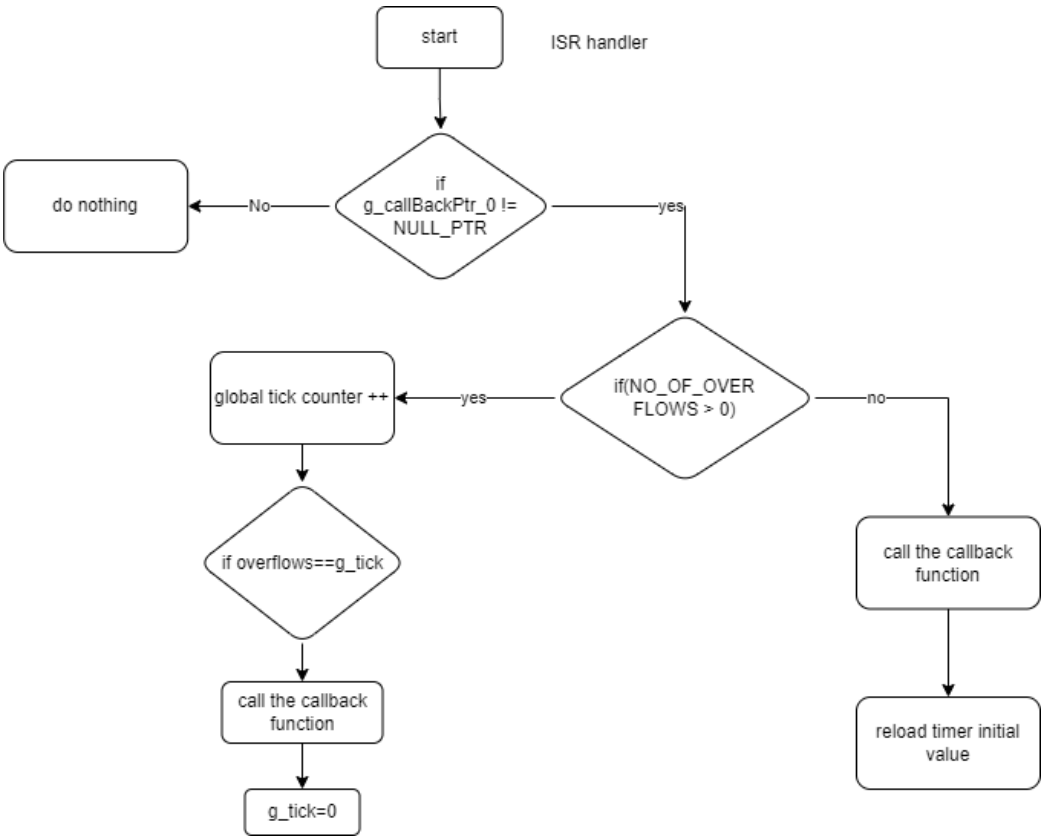


Timer0 APIs flowchart

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



ISR (TIMER0_OVF_vect)



High level design

Module description

LCD module

The LCD module is responsible for the initialization of the pins connected to the LCD and controls the LCD itself.

LCD driver documentation

```
/**
 * @brief Initializes the pins connected to the LCD display, clears the display,
 * and sets the cursor to the home position.
 */
void LCD_init(void);

/**
 * @brief Sends a command to the LCD display.
 * @param[in] lcd_command The specified command to be sent to the LCD.
 */
void LCD_sendCommand(uint8 lcd_command);

/**
 * @brief Displays the specified character on the LCD display.
 * @param[in] character The selected character to be displayed.
 */
void LCD_displayCharacter(uint8 character);

/**
 * @brief Displays a whole string on the LCD.
 * @param[in] string Address of the string to be displayed on the LCD.
 */
void LCD_displayString(uint8 *string);

/**
 * @brief Stores a custom character in the CGRAM.
 * @param custom_character[in] Address of the bitmap representation of the
character to be created.
 * @param cgram_location[in] Location in CGRAM to write the custom character in.
 */
```

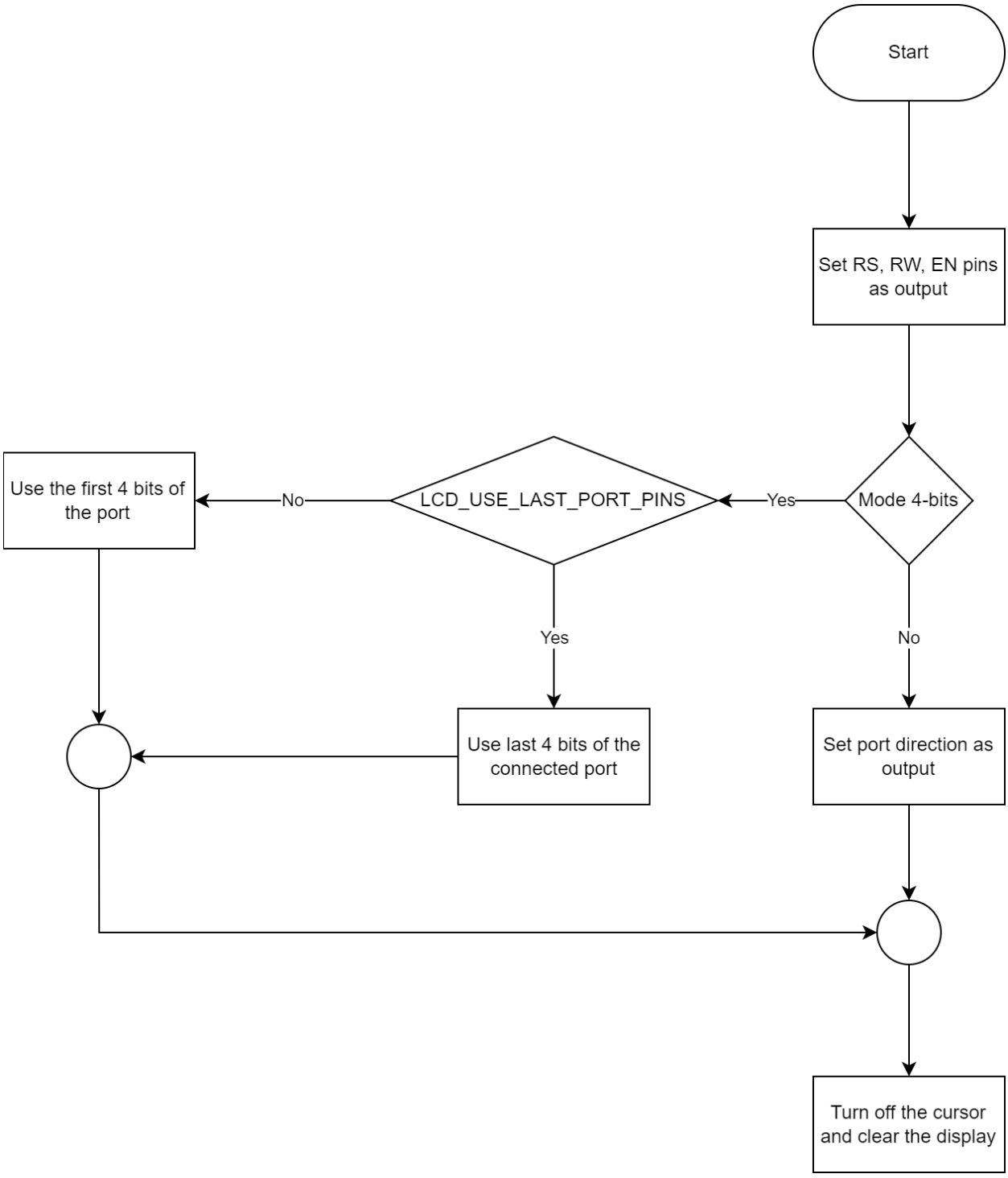
```c
void LCD_createCustomCharacter(uint8 *custom_character, uint8 cgram_location);

/**
 * @brief Moves the cursor to a specific row and column on the LCD display.
 * @param row The specified row, either row 0 or row 1.
 * @param col The specified column from 0 to 15.
 */
void LCD_moveCursor(uint8 row,  uint8 col);
```

Low level Design

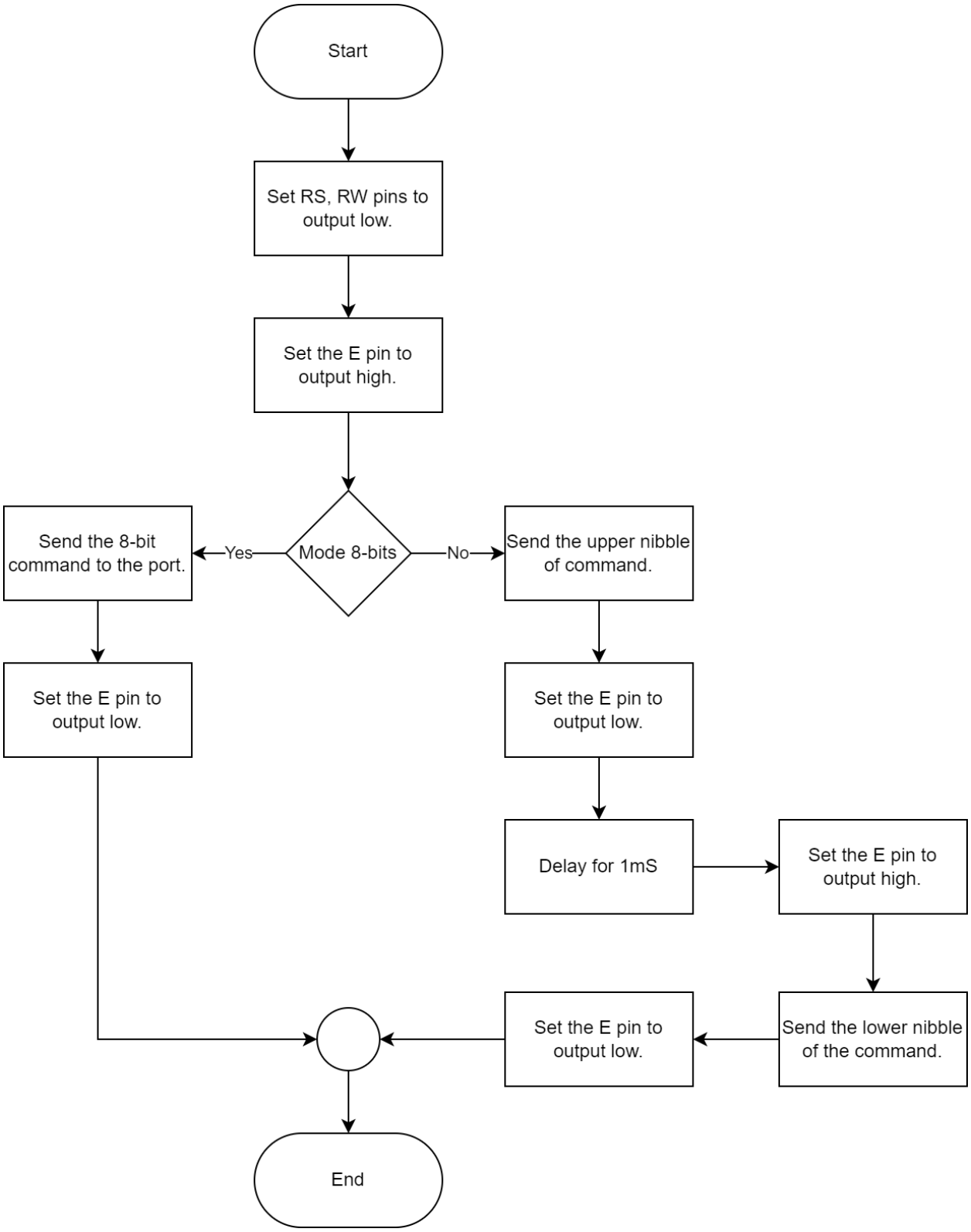
LCD module

1. void LCD_init(void);
2. void LCD_sendCommand(uint8 lcd_command);
3. void LCD_displayCharacter(uint8 character);
4. void LCD_displayString(uint8 *string);
5. void LCD_createCustomCharacter(uint8 *custom_character, uint8 cgram_location);
6. void LCD_moveCursor(uint8 row, uint8 col);

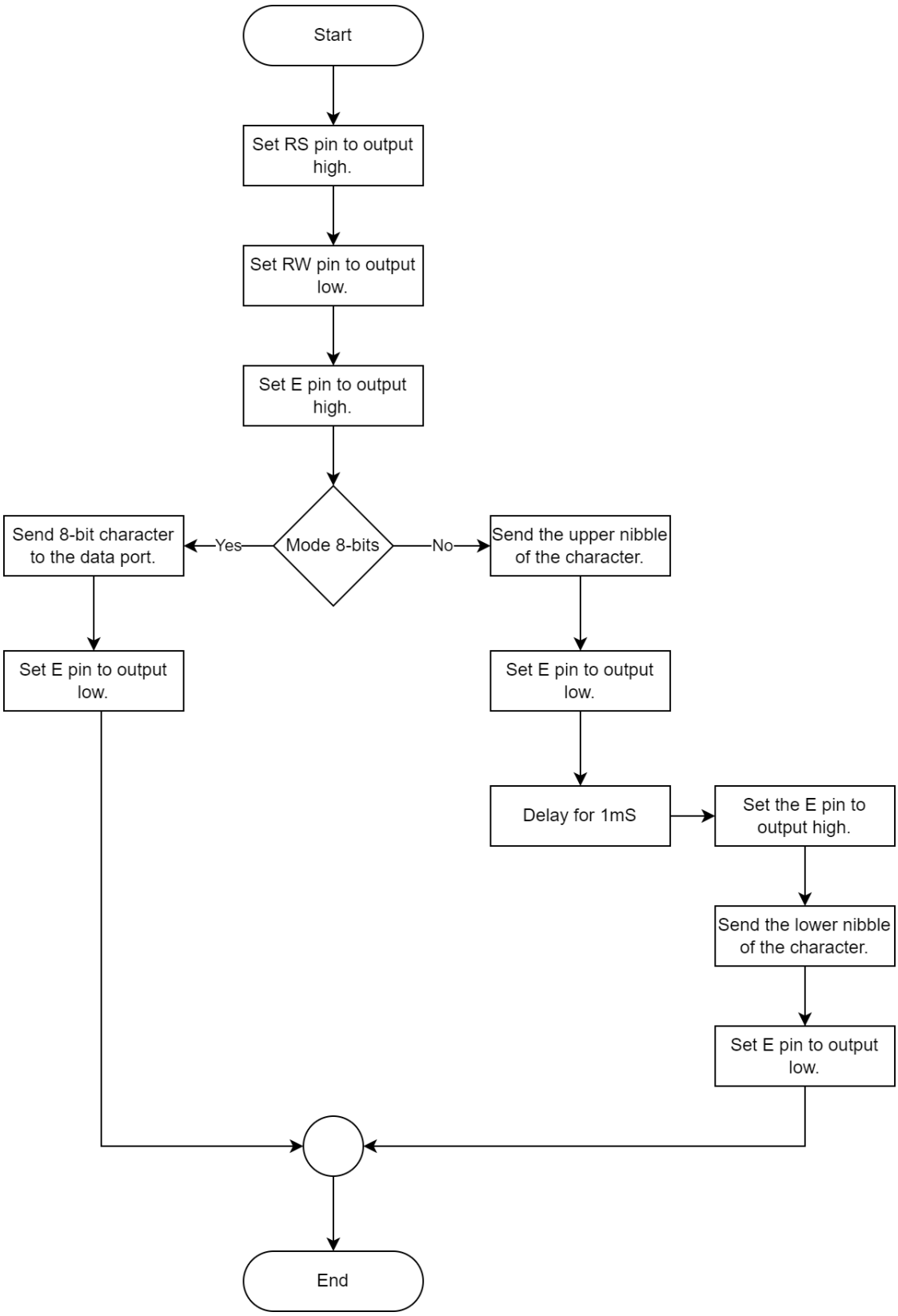
void LCD_init(void);



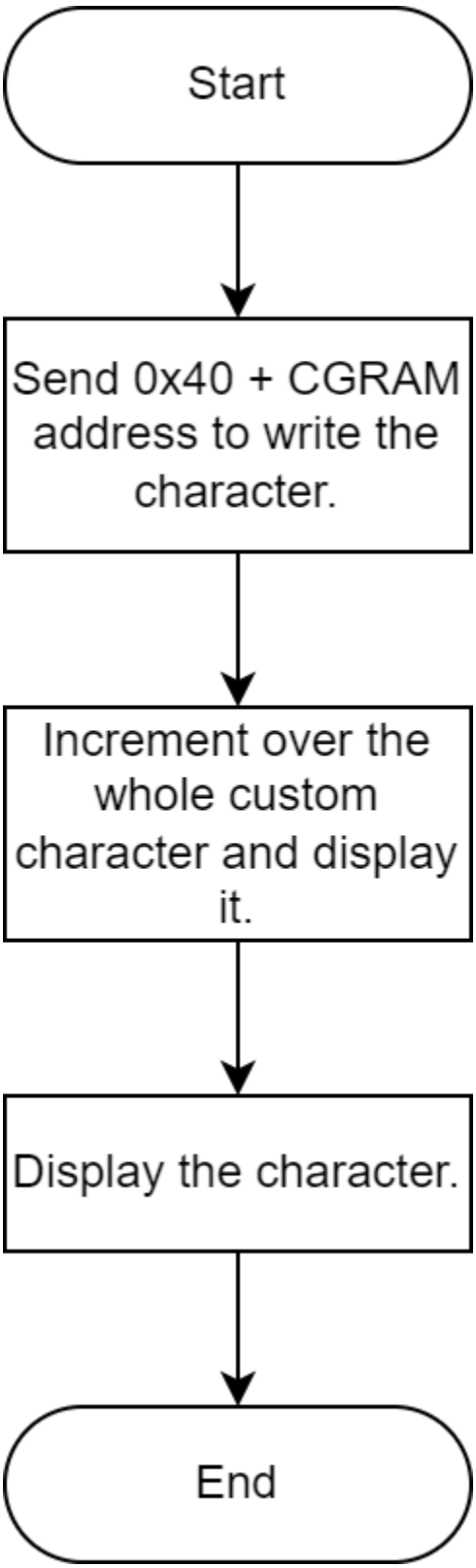
void LCD_sendCommand(uint8 lcd_command);



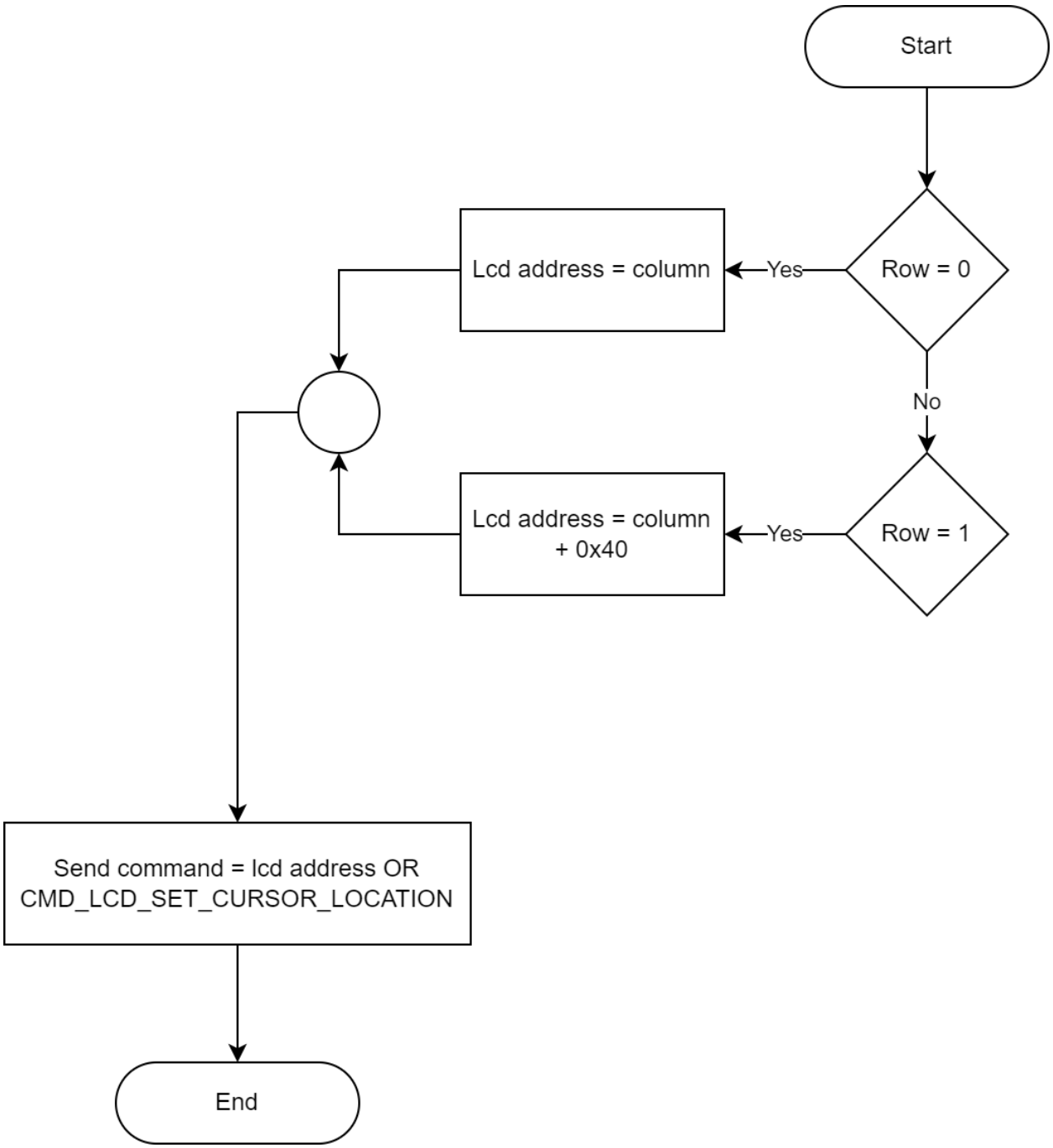
void LCD_displayCharacter(uint8 character);



void LCD_createCustomCharacter(uint8 *custom_character, uint8 cgram_location);



void LCD_moveCursor(uint8 row, uint8 col);



High level design

Module description

Temperature sensor

This module is responsible for reading the ADC value and converting into to a temperature value.
The module depends on the following equation to calculate the current temperature being read by the ADC:

$$\frac{Current\ ADC\ value * Sensor\ Max\ Rated\ Temp * ADC\ Reference\ Voltage}{ADC\ Max\ value * Sensor\ Max\ Rated\ Voltage}$$

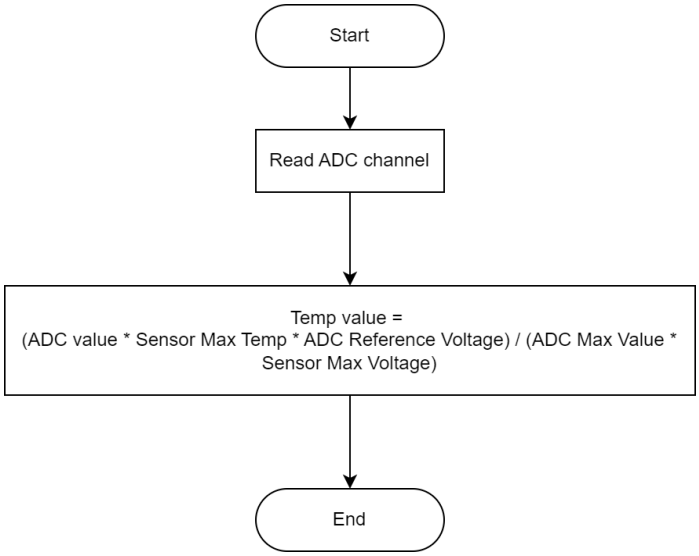
LM32 driver documentation

```
/**
 * @brief Calculate the temperature from the ADC digital value.
 */
uint8 LM35_getTemperature(void);
```

Low level Design

LM32 module

uint8 LM35_getTemperature(void);


Start
Read ADC channel
Temp value =
(ADC value * Sensor Max Temp * ADC Reference Voltage) / (ADC Max Value *
Sensor Max Voltage)
End

High level design

Module description

Buzzer module

The buzzer module is responsible for the initialization of the pins connected to the buzzer module and controls the state of the buzzer.

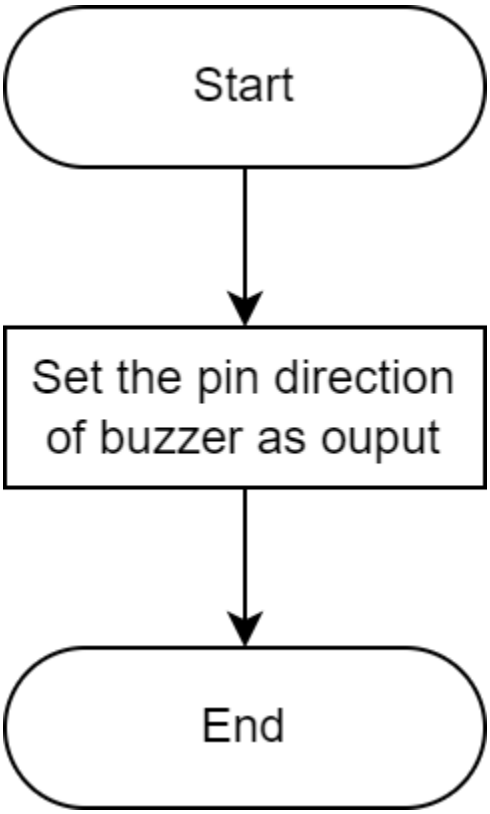
Buzzer driver documentation

```
/**
 * @brief Initializes the pin connected to the buzzer.
 * @param[in] port_id Specifies the GPIO port to be configured.
 * This parameter can be one of PORTx_ID.
 * @param[in] pin_id Specifies the GPIO pin to be configured.
 * This parameter can be one of PINx_ID.
 */
void BUZZER_init(uint8 port_id, uint8 pin_id);

/**
 * @brief Turns on the buzzer.
 * @param[in] port_id Specifies the GPIO port to be configured.
 * This parameter can be one of PORTx_ID.
 * @param[in] pin_id Specifies the GPIO pin to be configured.
 * This parameter can be one of PINx_ID.
 */
void BUZZER_on(uint8 port_id, uint8 pin_id);

/**
 * @brief Turns off the buzzer.
 * @param[in] port_id Specifies the GPIO port to be configured.
 * This parameter can be one of PORTx_ID.
 * @param[in] pin_id Specifies the GPIO pin to be configured.
 * This parameter can be one of PINx_ID.
 */
void BUZZER_off(uint8 port_id, uint8 pin_id);
```

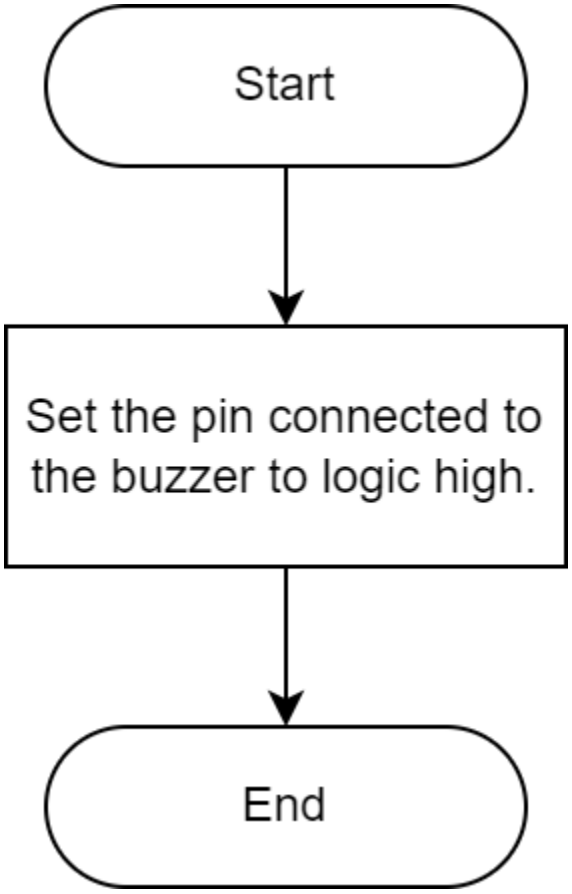
Low level Design

Buzzer module

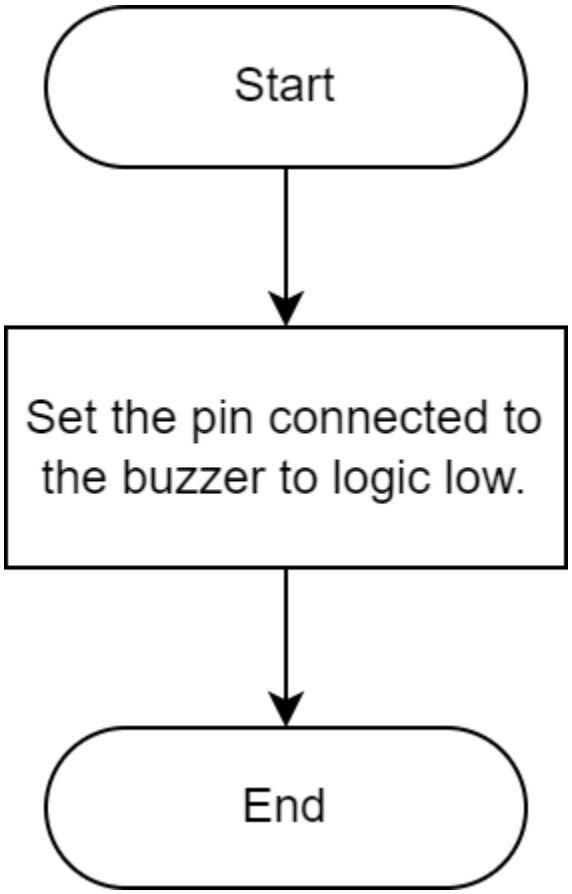
`void BUZZER_init`

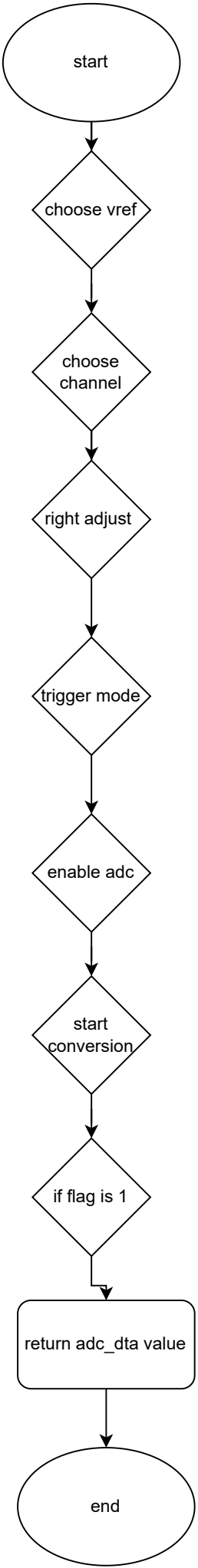


void BUZZER_on



void BUZZER_off




start
choose vref
choose channel
right adjust
trigger mode
enable adc
start conversion
if flag is 1
return adc_dta value
end

ADC driver APIs

```
/*******************************************************************************
 *                                 Definitions                                 *
 *******************************************************************************/
#define ADC_MAXIMUM_VALUE    1023
#define ADC_REF_VOLT_VALUE   5

/*******************************************************************************
 *                                 type definitions                            *
 *******************************************************************************/
typedef enum{
       AREF,AVCC,INTERNAL=3
}EN_ADC_Vrefernce;

typedef enum{
       ADC0,ADC1,ADC2,ADC3,ADC4,ADC5,ADC6,ADC7
}EN_ADC_channel;

typedef enum{
       F_2=1,F_4,F_8,F_16,F_32,F_64,F_128
}EN_ADC_prescaler;

typedef enum{
       Vref_error,channel_error,clk_error,success
}EN_ADC_state;
/*******************************************************************************
 *                        Functions Prototypes                                 *
 *******************************************************************************/
Description
EN_ADC_state ADC_init(EN_ADC_Vrefernce Vref,EN_ADC_prescaler ADC_CLK);
Initialize acd vref and enable adc
EN_ADC_state ADC_analogRead(uint8 channel_ID,uint16* ADC_read);

Choose channel and read digital value from digital register
```