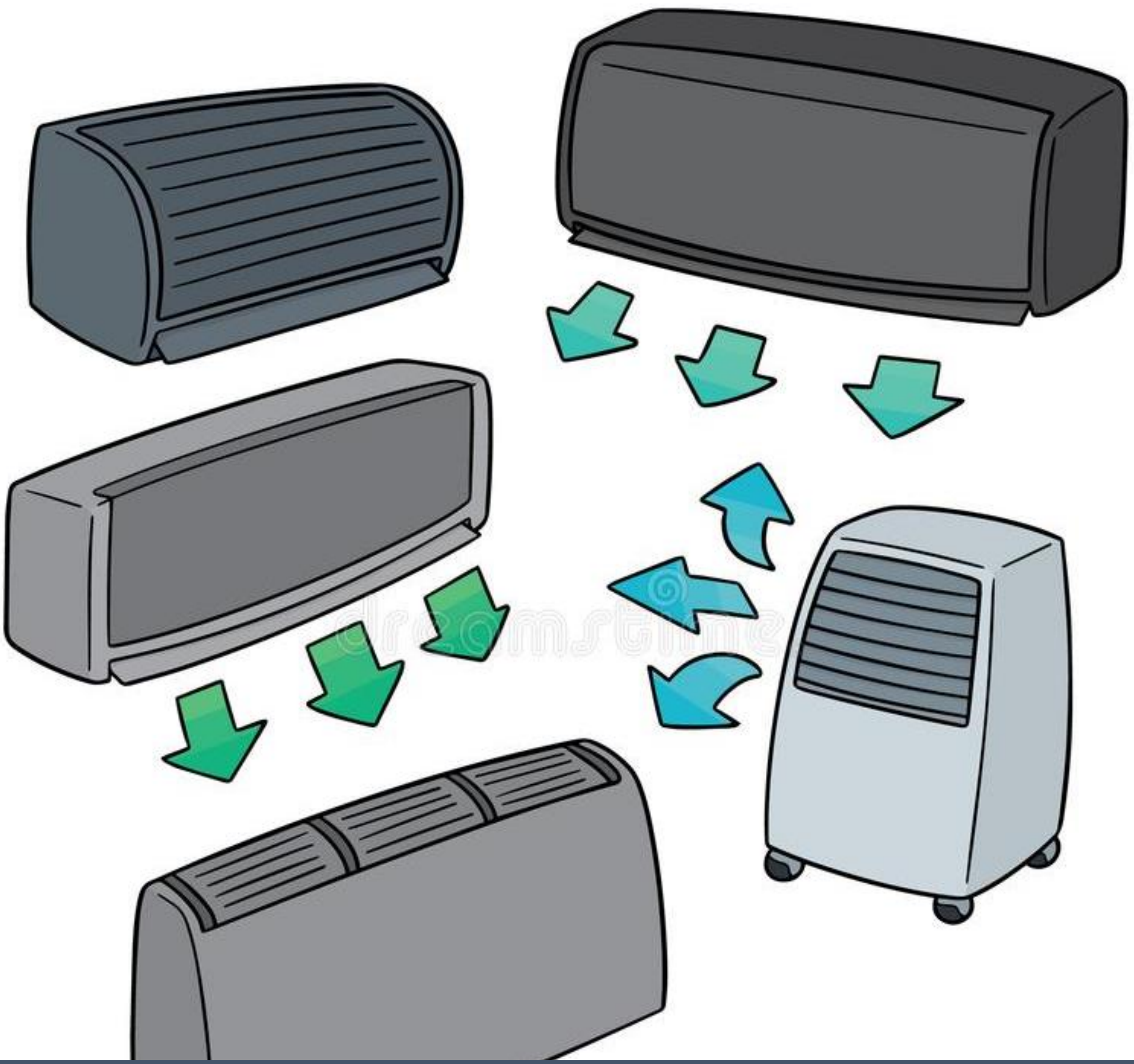


AIR CONDITIONER PROJECT

BY: Momen Hassan, Ahmed Atef, and Ahmed Mohamed Hesham

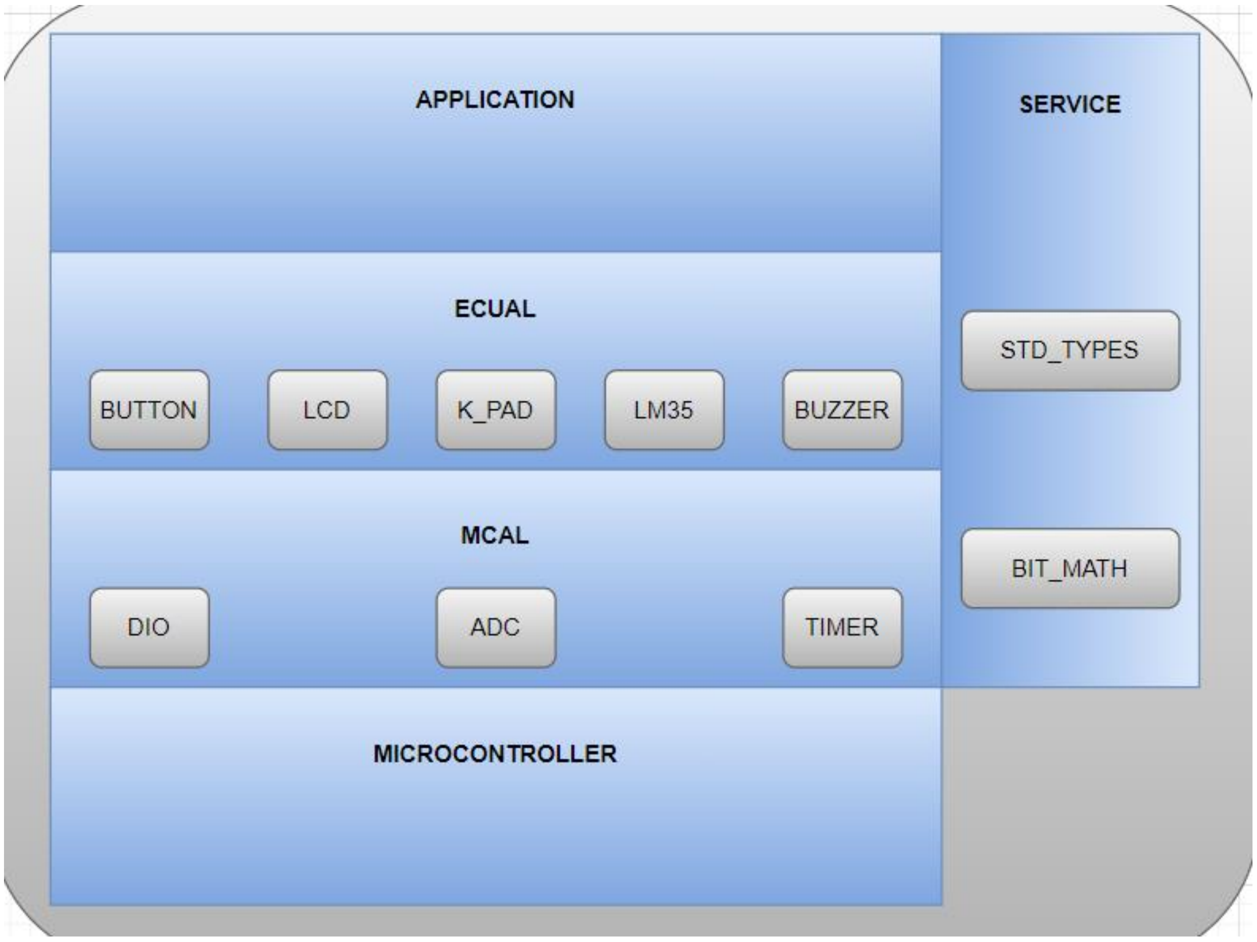
4/13/23

Automotive Boot Camp

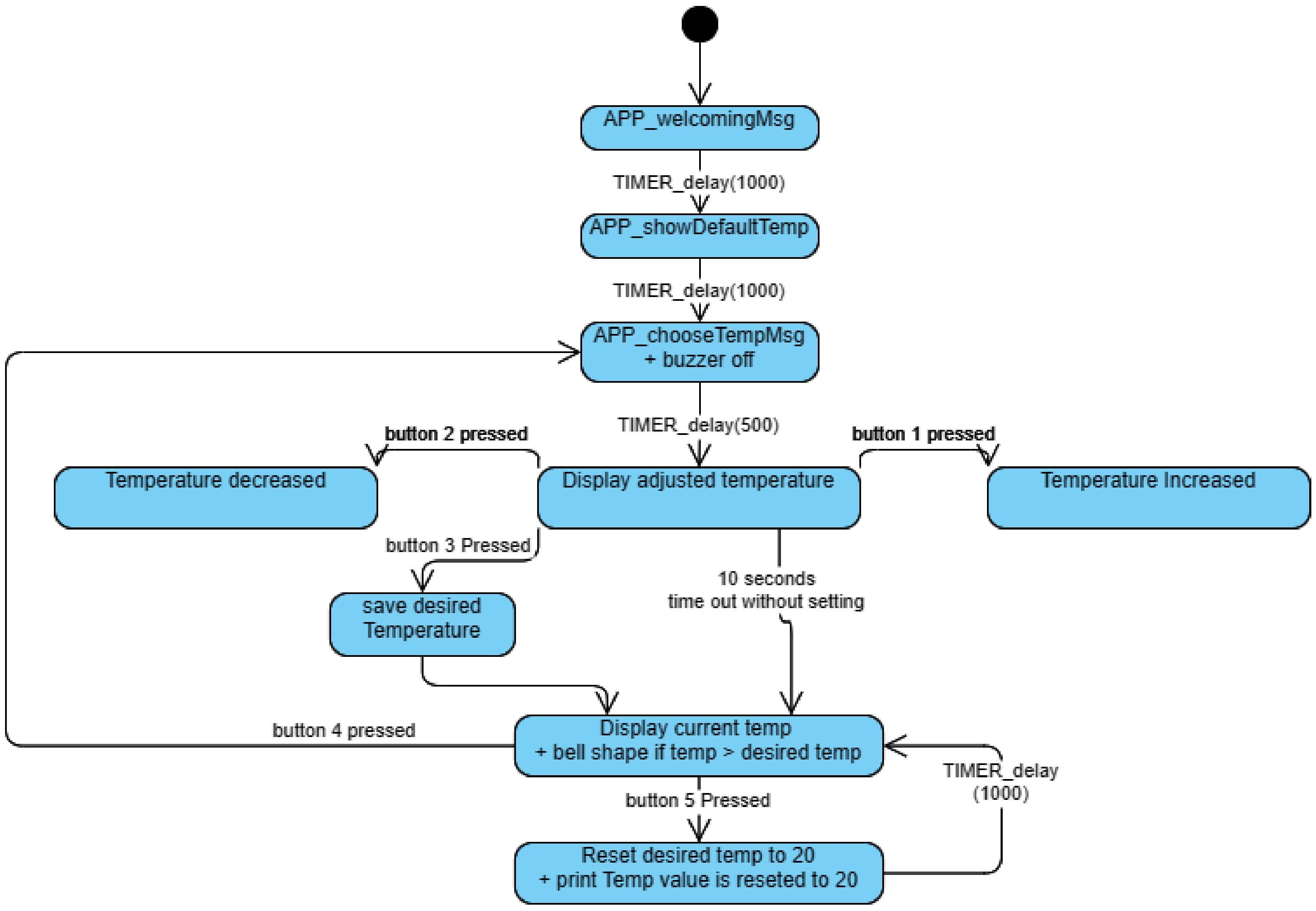
Contents

Air Conditioner Design

Layered Architecture



Project State Machine Diagram



INTRODUCTION

This project is developing software for temperature sensor integrated with ATMEGA32, 3*3 keypad, and LCD to make some functionalities similar to what happens with ACs.

So basically, the user will adjust the temperature he/she desires and the MC will compare it with the actual temperature and it buzzes if the actual temperature is higher than the desired temperature.

It consists of four layers:

1- APP

This layer is responsible for integrating application modules and peripherals to perform project functionality via using their APIs

2- ECUAL

In this layer modules' drivers are developed which are the keypad's driver, temperature sensor's driver, and LCD driver.

This layer is like a middle junction between the application layer and the microcontroller abstraction layer

3- MCAL

In this layer, peripherals' drivers are developed: DIO's driver, timers' driver, and ADC.

4- SERVICE

This layer consists of files.h which will serve the main three layers while developing, like it has important data types' type defs, and bit manipulation macros like functions

Module, Peripherals, & Supporting Drivers Description

DIO (Digital Input/Output): This module deals with the digital input and output operations, such as reading and writing to digital pins of a microcontroller or a microprocessor. It may include functions for setting pin direction, reading and writing digital values, and handling interrupts related to digital pins.

Timer: This module deals with timer operations, such as configuring and handling timers in the microcontroller or microprocessor. It may include functions for setting timer intervals, handling timer interrupts, and measuring time. And This module deals with generating PWM signals using normal mode, which are used for controlling the intensity of an output signal, such as controlling the speed of motors or the brightness of LEDs. It may include functions for configuring and controlling PWM signals.

ADC: Through this module we can initialize ADC peripheral which is responsible for converting analog input signal to digital signal, and we check on the completion of the conversation by polling on interrupt flag.

LM35: This module is the temperature sensor used in our project. Its electrical output is proportional to the temperature in degrees Celsius. It is a linear device which means that the voltage at the output of the LM35 increases proportionally to the temperature. It is rated to operate over a -55°C to 150°C temperature range. The scale factor is 0.01 V/°C.

LCD: LCD stands for Liquid Crystal Display. LCD is a type of flat panel display which uses liquid crystals in its primary form of operation. It uses the light-modulating properties of liquid crystals combined with polarizers to display images. It has two pieces of polarized glass (also called substrate) that contain a liquid crystal material between them. A backlight or a reflector creates light that passes through the first substrate and the liquid crystal. The liquid crystals can twist or untwist depending on the electric voltage applied across them. This changes the angle of light passing through the second substrate and the polarizing film. The angel of light determines the color and brightness of the pixels that form the images on the LCD.

Keypad: We are using 3*3 keypad which means we have 3 rows and 3 columns, rows are connected to output high, and columns are connected to be inputs and enable internal pullups. For reading we pass low output simultaneously to the row pins and check if there is any change in the columns.

Buzzer: is an output module that buzzes when you write on it HIGH output.

BIT_MATH: This module provides functions for performing bitwise operations, such as AND, OR, XOR, and shifting, which are commonly used for manipulating individual bits in registers or memory locations.

Standard Types: This module includes standard data types, such as integer types, floating-point types, and Boolean types, which are used for representing data in a standardized way across the system.

DRIVERS' DOCUMENTATION

1. DIO

DIO_init(uint8_t portNumber, uint8_t pinNumber, uint8_t direction);

Function Name	**DIO_init**
Description	Initializes DIO pins' direction, output current, and internal attach
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber, uint8_t direction
Parameters (out)	None
Return Value	WRONG_PORT_NUMBER, WRONG_PIN_NUMBER, WRONG_DIRECTION, E_OK

DIO_write(uint8_t portNumber, uint8_t pinNumber, uint8_t value);

Function Name	**DIO_write**
Description	Write on DIO pins' a specific output High or Low
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber, uint8_t value
Parameters (out)	None
Return Value	WRONG_PORT_NUMBER, WRONG_PIN_NUMBER, WRONG_VALUE, E_OK

DIO_toggle(uint8_t portNumber, uint8_t pinNumber);

Function Name	**DIO_toggle**
Description	Toggle the output of a specific pin
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber
Parameters (out)	None
Return Value	WRONG_PORT_NUMBER, WRONG_PIN_NUMBER, E_OK

DIO_read(uint8_t portNumber, uint8_t pinNumber, uint8_t *value);

Function Name	**DIO_read**
Description	Read input from a pin and send it back in a pointer to uint8_t
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber
Parameters (out)	uint8_t *value
Return Value	WRONG_PORT_NUMBER, WRONG_PIN_NUMBER, E_OK

2. TIMERS

en_timerError_t TIMER_init(u8 u8_a_timerUsed);

Function Name	**TIMER_init**
Description	Initializes a specific timer to work as a CTC or overflow timer
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t timerUsed
Parameters (out)	None
Return Value	EN_timerError_t

en_timerError_t TIMER_setTime(u8 u8_a_timerUsed, u32 u32_a_desiredTime);

Function Name	**TIMER_setTime**
Description	Used to set time at which the timer interrupt will fires and execute a desired function
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t timerUsed, uint32_t desiredTime
Parameters (out)	None
Return Value	EN_timerError_t

en_timerError_t TIMER_start(u8 u8_a_timerUsed);

Function Name	**TIMER_start**
Description	Start specific timer to count
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t timerUsed
Parameters (out)	None
Return Value	EN_timerError_t

en_timerError_t TIMER_stop(u8 u8_a_timerUsed);

Function Name	**TIMER_stop**
Description	Stop specific timer from counting
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t timerUsed
Parameters (out)	None
Return Value	EN_timerError_t

en_timerError_t TIMER_pwmGenerator(u8 u8_a_timerUsed, u32 u32_a_desiredDutyCycle);

Function Name	**TIMER_pwmGenerator**
Description	Generates PWM signal using normal mode for a specific timer
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_timerUsed, u8_a_desiredDutyCycle
Parameters (out)	None
Return Value	en_timerError_t

void TIMER_setCallBack(u8 u8_a_timerUsed, void (*funPtr)(void));

Function Name	**TIMER_setCallBack**
Description	Initializes Sends pointer to function to be called when the timer's interrupt fires
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	uint8_t portNumber, uint8_t pinNumber, uint8_t direction
Parameters (out)	None
Return Value	None

en_timerError_t TIMER_stopInterrupt(u8 u8_a_timerUsed);

Function Name	**TIMER_stopInterrupt**
Description	Disable a specific timer’s peripheral interrupt
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_timerUsed
Parameters (out)	None
Return Value	en_timerError_t

en_timerError_t TIMER_delay(u8 u8_a_timerUsed, u32 u32_a_timeInMS);

Function Name	**TIMER_enableInterrupt**
Description	Generates a delay using a specific timer
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_timerUsed, u32_a_timeInMS
Parameters (out)	None
Return Value	en_timerError_t

en_timerError_t TIMER_enableInterrupt(u8 u8_a_timerUsed);

Function Name	**TIMER_enableInterrupt**
Description	Enables a specific timer’s peripheral interrupt
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_timerUsed
Parameters (out)	None
Return Value	en_timerError_t

3. ADC

void ADC_init (void);

Function Name	**ADC_init**
Description	Initialize ADC according to preprocessed configured definitions
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

u16 ADC_read (void);

Function Name	**ADC_read**
Description	Pulls on ADC interrupt flag till it rises to start returning a conversion to the user
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	u16

4. LCD

void LCD_Init(void);

Function Name	**LCD_Init**
Description	Initialize LCD according to preprocessed configured definitions
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

void LCD_PinsInit ();

Function Name	**LCD_PinInit**
Description	Initialize LCD pins directions according to preprocessed configured definitions
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

void LCD_WriteChar(u8 u8_a_ch);

Function Name	**LCD_WriteChar**
Description	Prints Character on LCD
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	U8_a_ch
Parameters (out)	None
Return Value	None

void LCD_WriteString(u8 *u8_a_str);

Function Name	**LCD_WriteString**
Description	Prints string on LCD
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	*u8_a_str
Parameters (out)	None
Return Value	None

void LCD_WriteNumber(i32 i32_a_num);

Function Name	**LCD_WriteNumber**
Description	Prints a specific number on LCD
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	i32_a_num
Parameters (out)	None
Return Value	None

void LCD_SetCursor(u8 u8_a_line,u8 u8_a_cell);

Function Name	**LCD_SetCursor**
Description	Changes Cursor’s Location
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_line, u8_a_cell
Parameters (out)	None
Return Value	None

void LCD_Clear(void);

Function Name	**LCD_Clear**
Description	Clears LCD’s screen and set cursor at line 0 cell 0
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

void LCD_ClearLoc(u8 u8_a_line ,u8 u8_a_cell,u8 u8_a_num);

Function Name	**LCD_ClearLoc**
Description	Clear specific cells from a specific location
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_line, u8_a_cell_, u8_a_num
Parameters (out)	None
Return Value	None

void LCD_CustomChar(u8 u8_a_loc,u8 *u8_a_pattern);

Function Name	**LCD_CustomChar**
Description	Creates a customized character
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_loc, *u8_a_pattern
Parameters (out)	None
Return Value	None

5. KEYPAD

void KEYPAD_init (void);

Function Name	**KEYPAD_init**
Description	Initialize KEYPAD according to preprocessed configured definitions
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

u8 KEYPAD_read (void);

Function Name	**KEYPAD_read**
Description	returns 0 if there is no key pressed or equivalent value for the key if there is a key pressed
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	U8

6. LM35

void LM35_init (void);

Function Name	**LM35_init**
Description	Initialize LM35 according to preprocessed configured definitions
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

u16 LM35_read (void);

Function Name	**LM35_read**
Description	returns Degree Celsius after Conversion
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	u16

7. BUZZEER

en_buzzerError_t BUZZER_init (u8 u8_a_buzzerNumber);

Function Name	**BUZZER_init**
Description	Initialize Buzzer according to preprocessed configured definitions
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_buzzerNumber
Parameters (out)	None
Return Value	BUZZER_OK, WRONG_BUZZER.

en_buzzerError_t BUZZER_on (u8 u8_a_buzzerNumber);

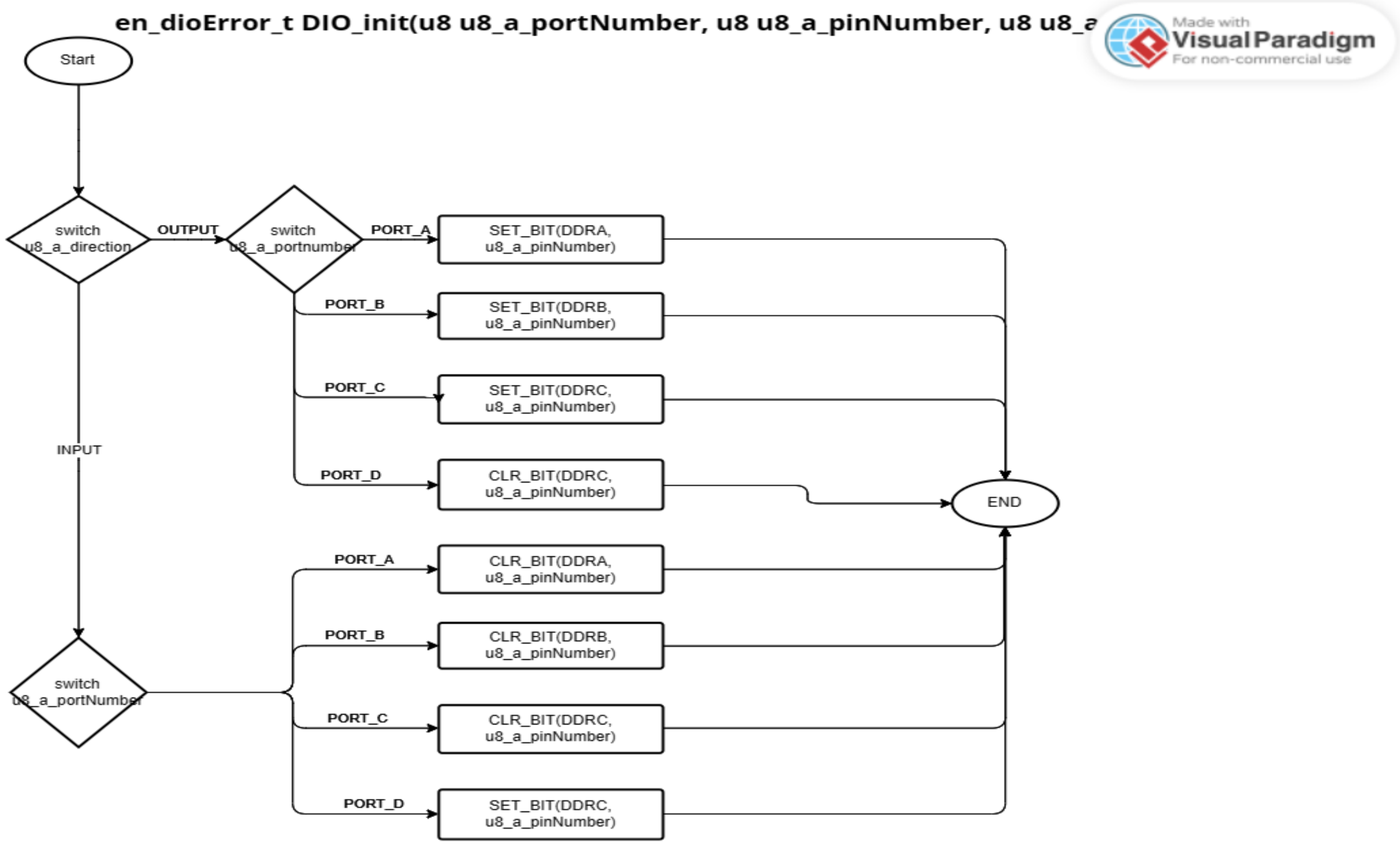
Function Name	**BUZZER_on**
Description	Switches Buzzer On
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_buzzerNumber
Parameters (out)	None
Return Value	BUZZER_OK, WRONG_BUZZER.

en_buzzerError_t BUZZER_off (u8 u8_a_buzzerNumber);

Function Name	**BUZZER_off**
Description	Turns Buzzer off
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	u8_a_buzzerNumber
Parameters (out)	None
Return Value	BUZZER_OK, WRONG_BUZZER.

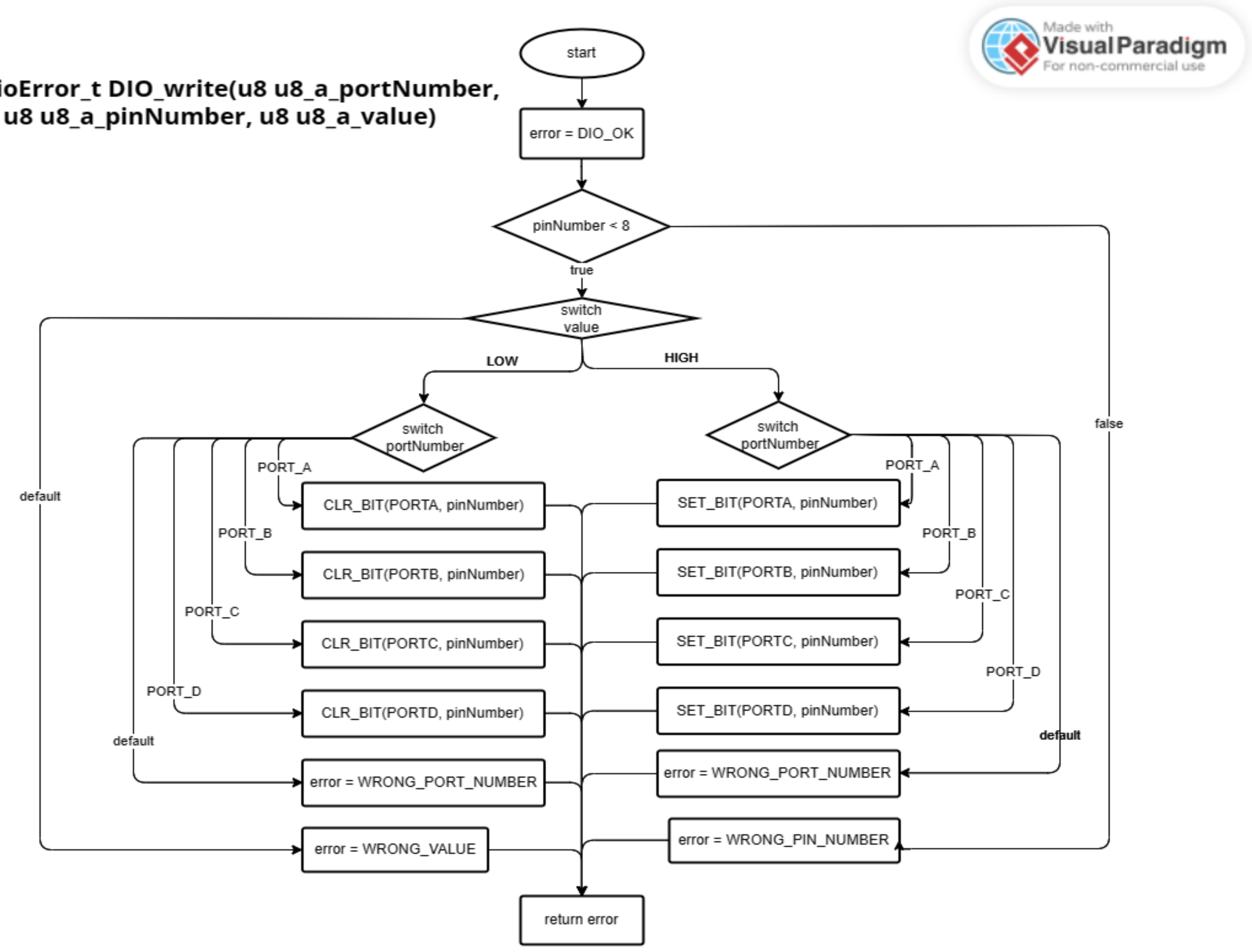
FUNCTIONS' FLOWCHARTS

1. DIO

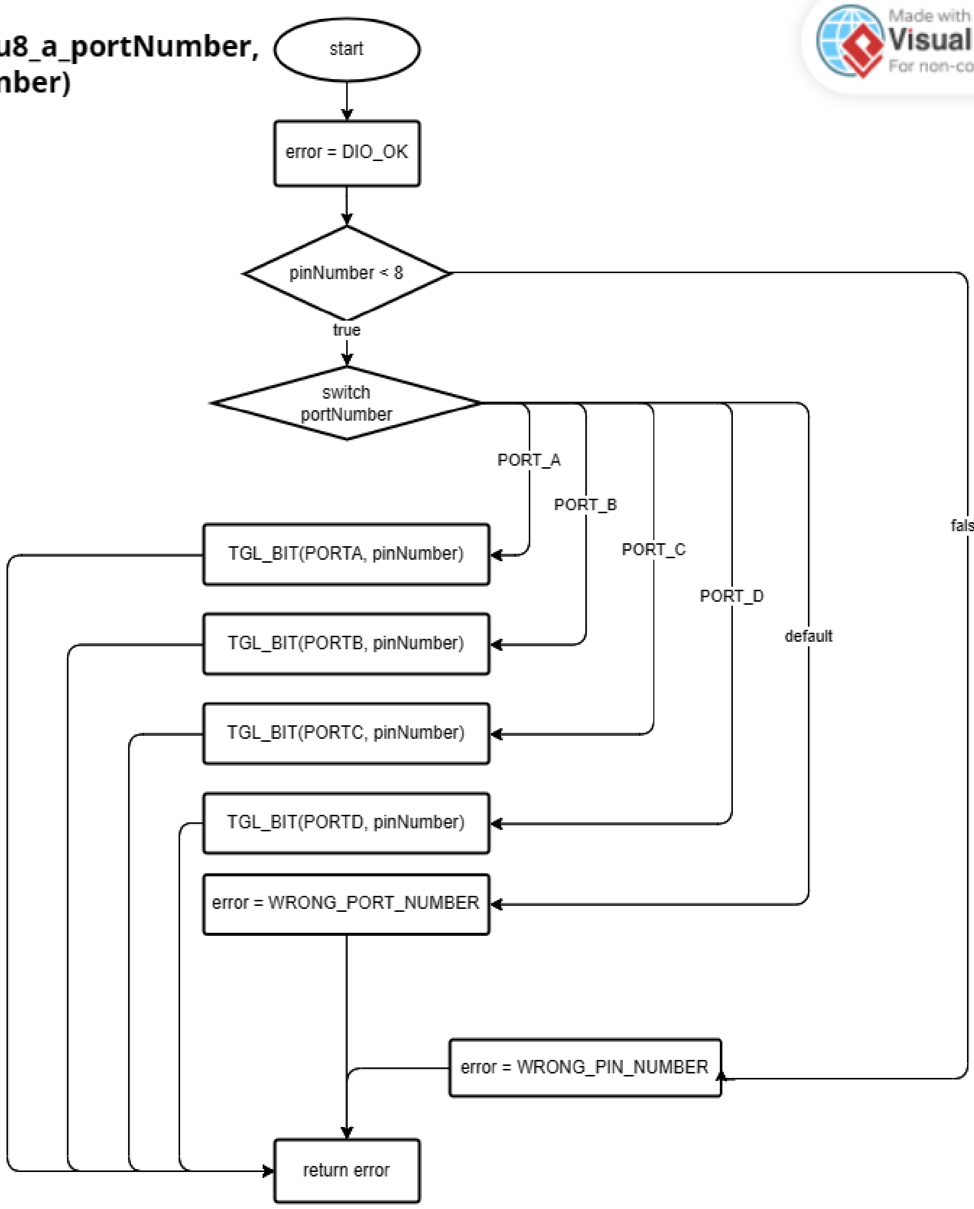


en_dioError_t DIO_init(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 u8_a
Made with Visual Paradigm For non-commercial use
Start
switch u8_a_direction
OUTPUT
switch u8_a_portnumber
PORT_A
SET_BIT(DDRA, u8_a_pinNumber)
PORT_B
SET_BIT(DDRB, u8_a_pinNumber)
PORT_C
SET_BIT(DDRC, u8_a_pinNumber)
PORT_D
CLR_BIT(DDRC, u8_a_pinNumber)
END
INPUT
switch u8_a_portNumber
PORT_A
CLR_BIT(DDRA, u8_a_pinNumber)
PORT_B
CLR_BIT(DDRB, u8_a_pinNumber)
PORT_C
CLR_BIT(DDRC, u8_a_pinNumber)
PORT_D
SET_BIT(DDRC, u8_a_pinNumber)

en_dioError_t DIO_write(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 u8_a_value)

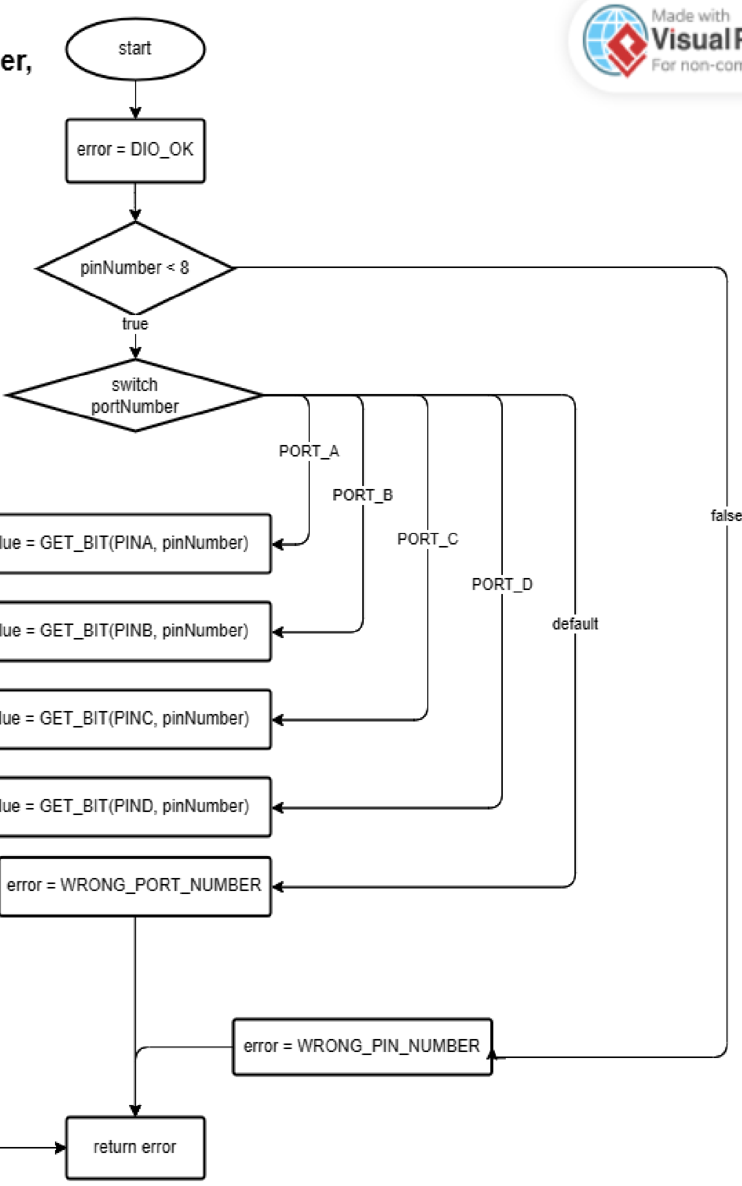

start
error = DIO_OK
pinNumber < 8
true
false
switch value
LOW
HIGH
default
switch portNumber
PORT_A
PORT_B
PORT_C
PORT_D
default
CLR_BIT(PORTA, pinNumber)
CLR_BIT(PORTB, pinNumber)
CLR_BIT(PORTC, pinNumber)
CLR_BIT(PORTD, pinNumber)
error = WRONG_PORT_NUMBER
error = WRONG_VALUE
switch portNumber
PORT_A
PORT_B
PORT_C
PORT_D
default
SET_BIT(PORTA, pinNumber)
SET_BIT(PORTB, pinNumber)
SET_BIT(PORTC, pinNumber)
SET_BIT(PORTD, pinNumber)
error = WRONG_PORT_NUMBER
error = WRONG_PIN_NUMBER
return error
Made with Visual Paradigm For non-commercial use

en_dioError_t DIO_toggle(u8 u8_a_portNumber, u8 u8_a_pinNumber)



start
error = DIO_OK
pinNumber < 8
true
false
switch portNumber
PORT_A
PORT_B
PORT_C
PORT_D
default
TGL_BIT(PORTA, pinNumber)
TGL_BIT(PORTB, pinNumber)
TGL_BIT(PORTC, pinNumber)
TGL_BIT(PORTD, pinNumber)
error = WRONG_PORT_NUMBER
error = WRONG_PIN_NUMBER
return error

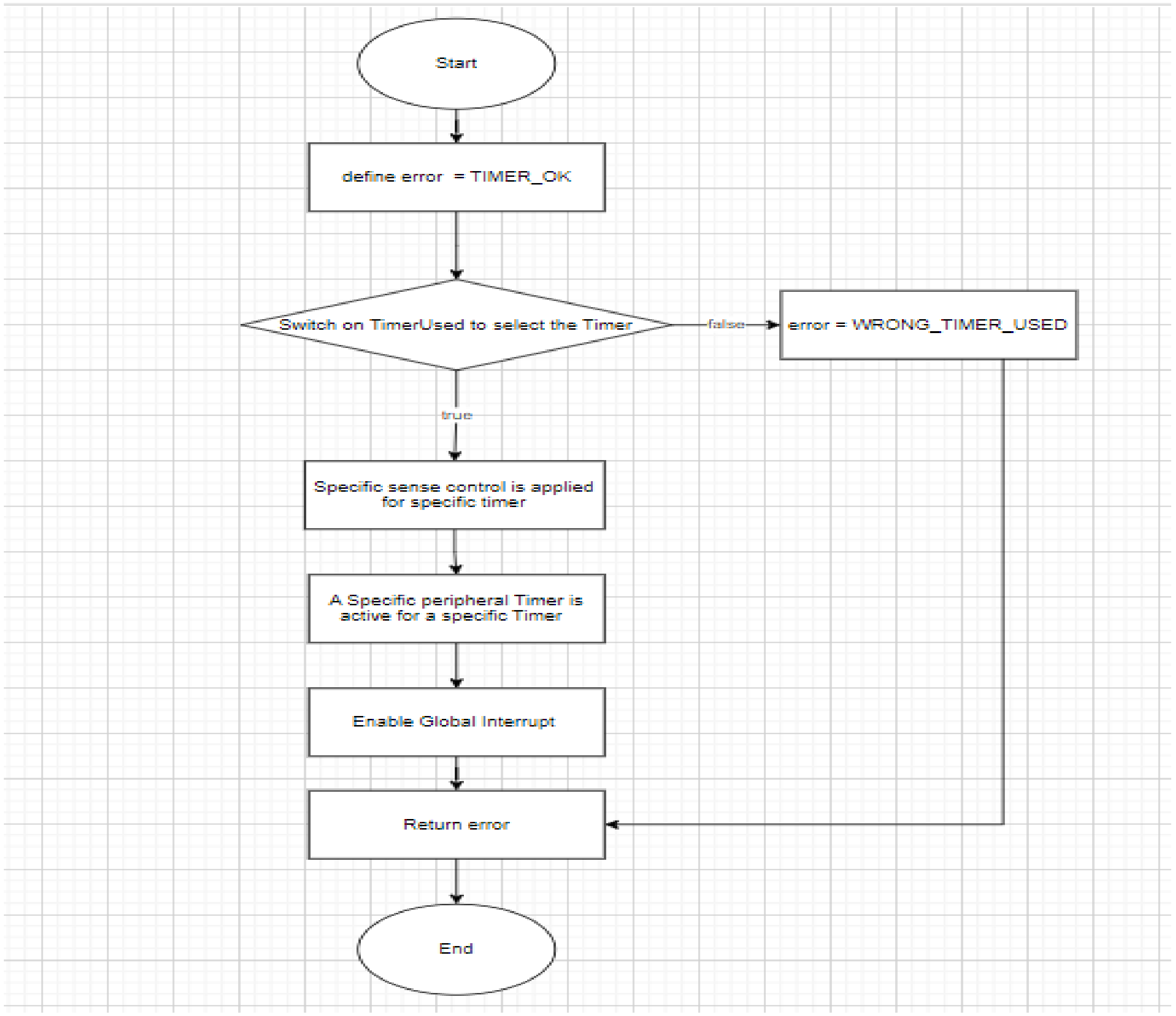
en_dioError_t DIO_read(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 *u8_a_value)



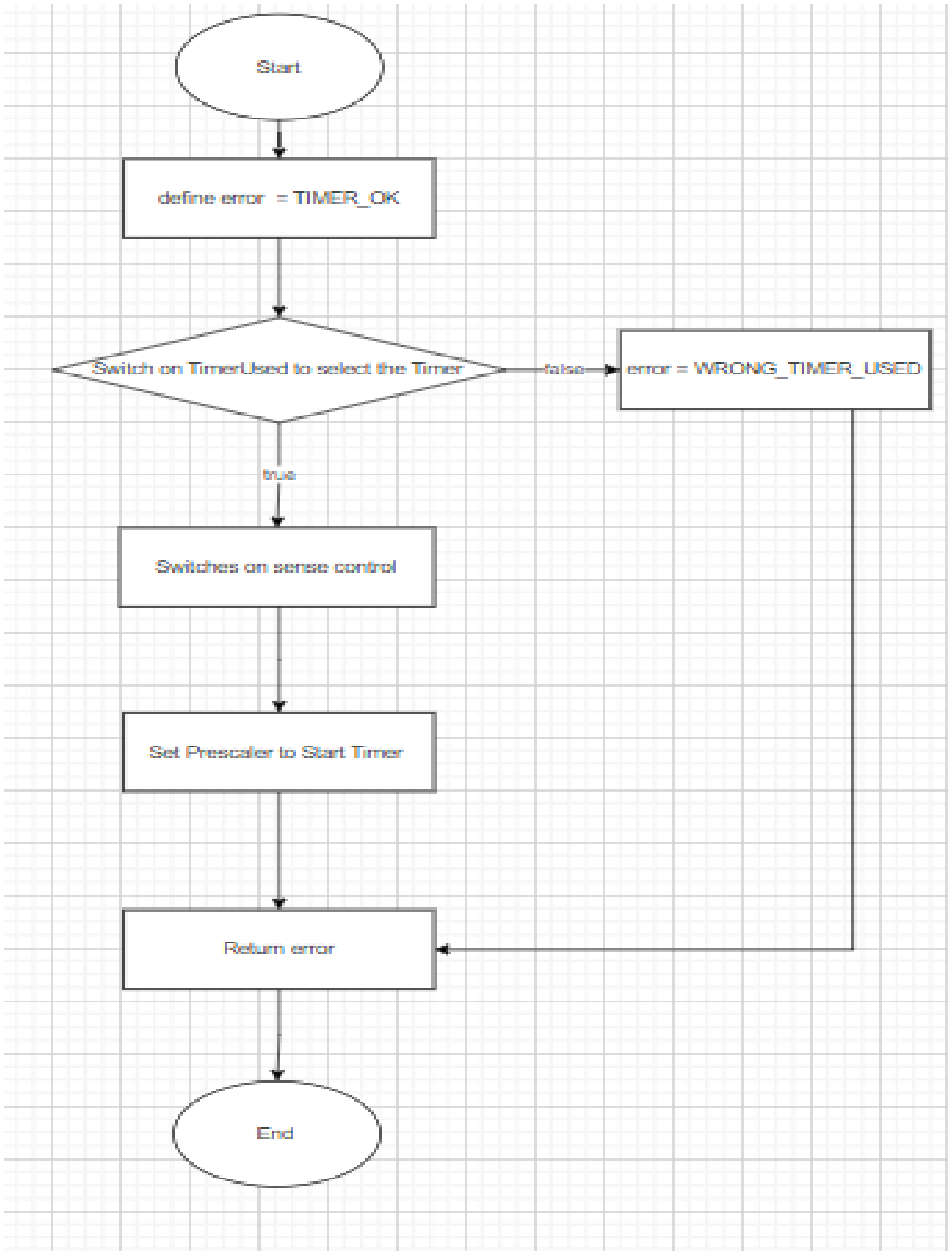
start
error = DIO_OK
pinNumber < 8
true
false
switch portNumber
PORT_A
PORT_B
PORT_C
PORT_D
default
value = GET_BIT(PINA, pinNumber)
value = GET_BIT(PINB, pinNumber)
value = GET_BIT(PINC, pinNumber)
value = GET_BIT(PIND, pinNumber)
error = WRONG_PORT_NUMBER
error = WRONG_PIN_NUMBER
return error

2. TIMERS

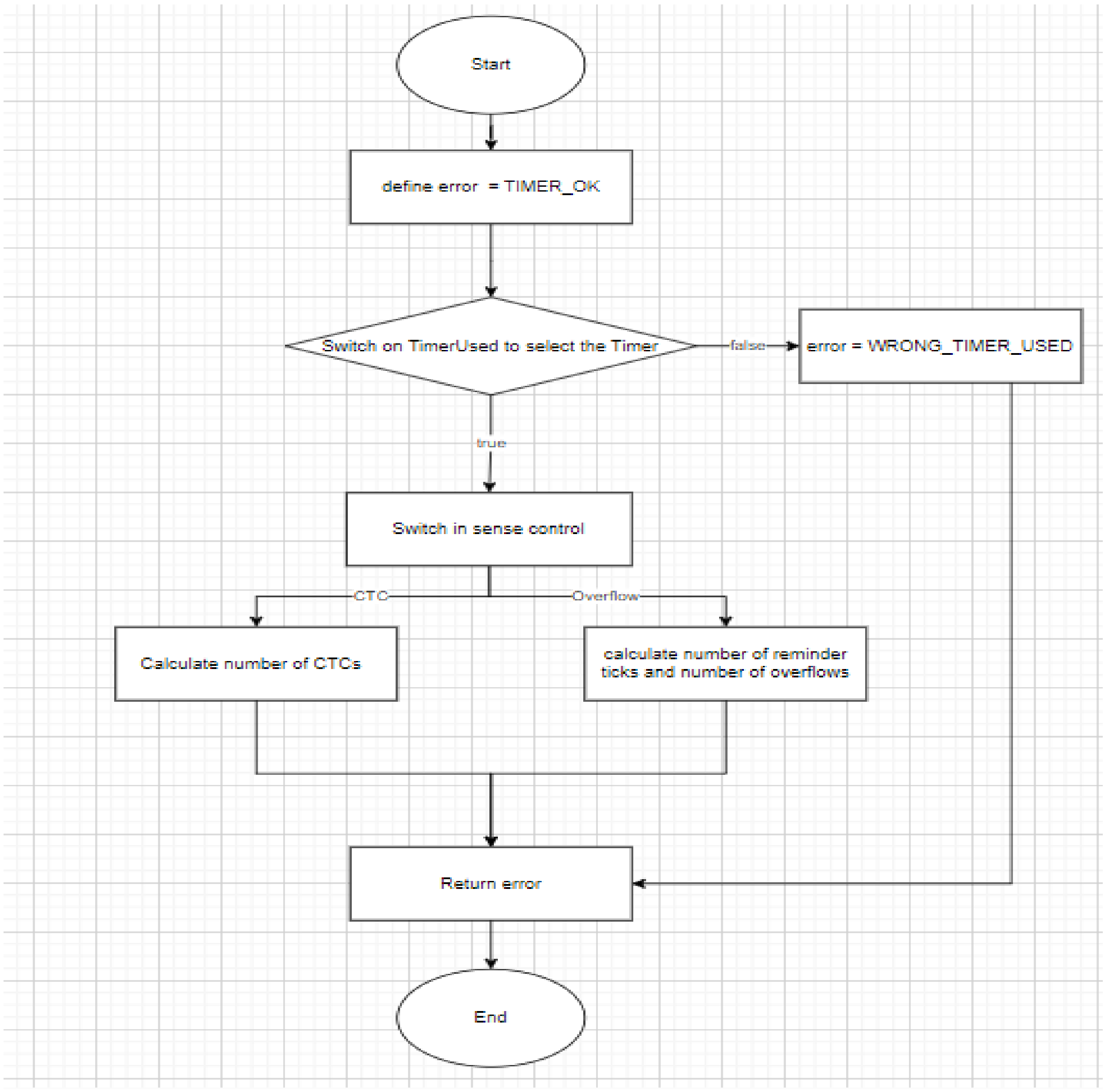
en_timerError_t TIMER_init(u8 u8_a_timerUsed);


Start
define error = TIMER_OK
Switch on TimerUsed to select the Timer
false
error = WRONG_TIMER_USED
true
Specific sense control is applied for specific timer
A Specific peripheral Timer is active for a specific Timer
Enable Global Interrupt
Return error
End

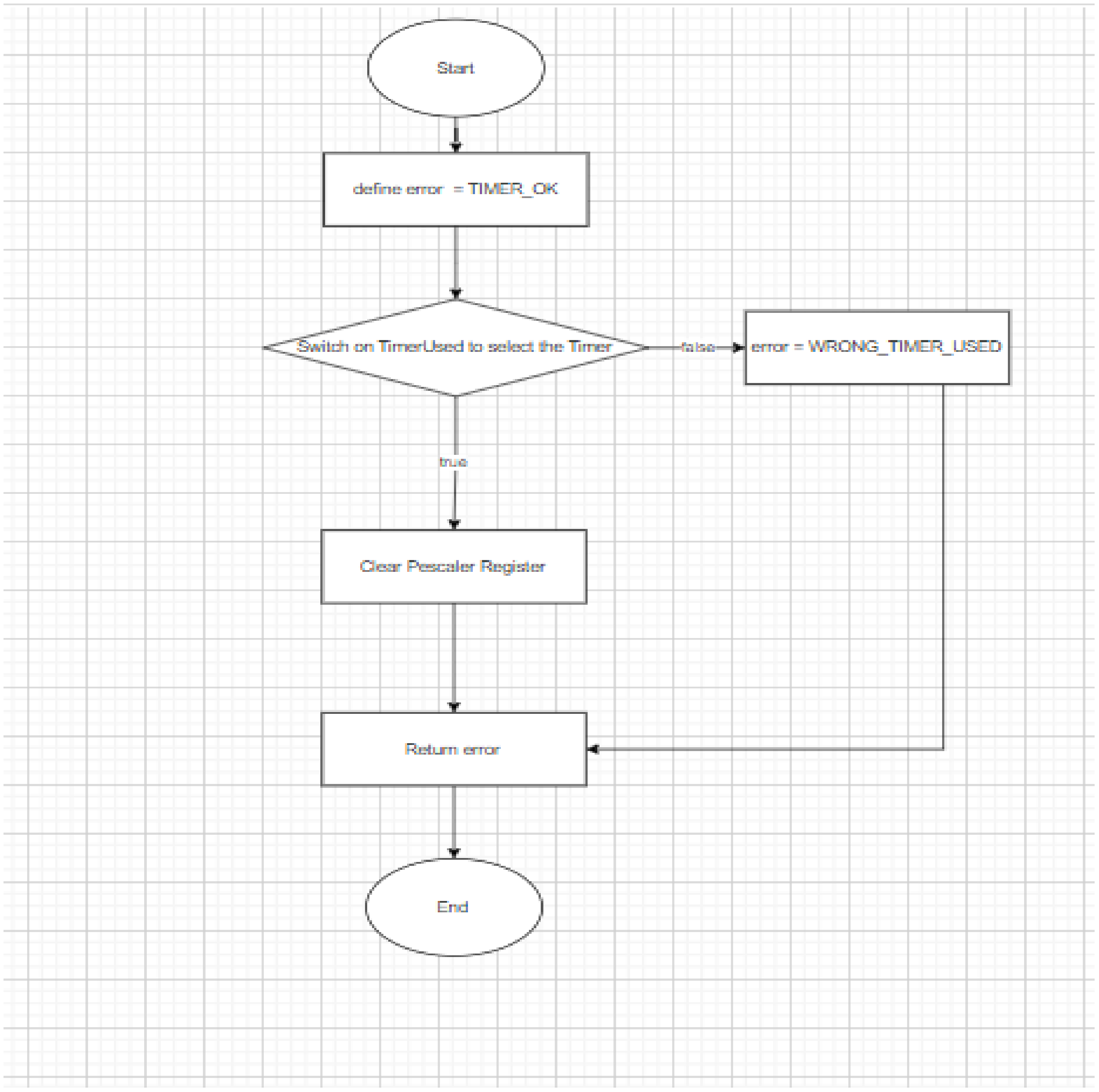
en_timerError_t TIMER_start(u8 u8_a_timerUsed);


Start
define error = TIMER_OK
Switch on TimerUsed to select the Timer
false
error = WRONG_TIMER_USED
true
Switches on sense control
Set Prescaler to Start Timer
Return error
End

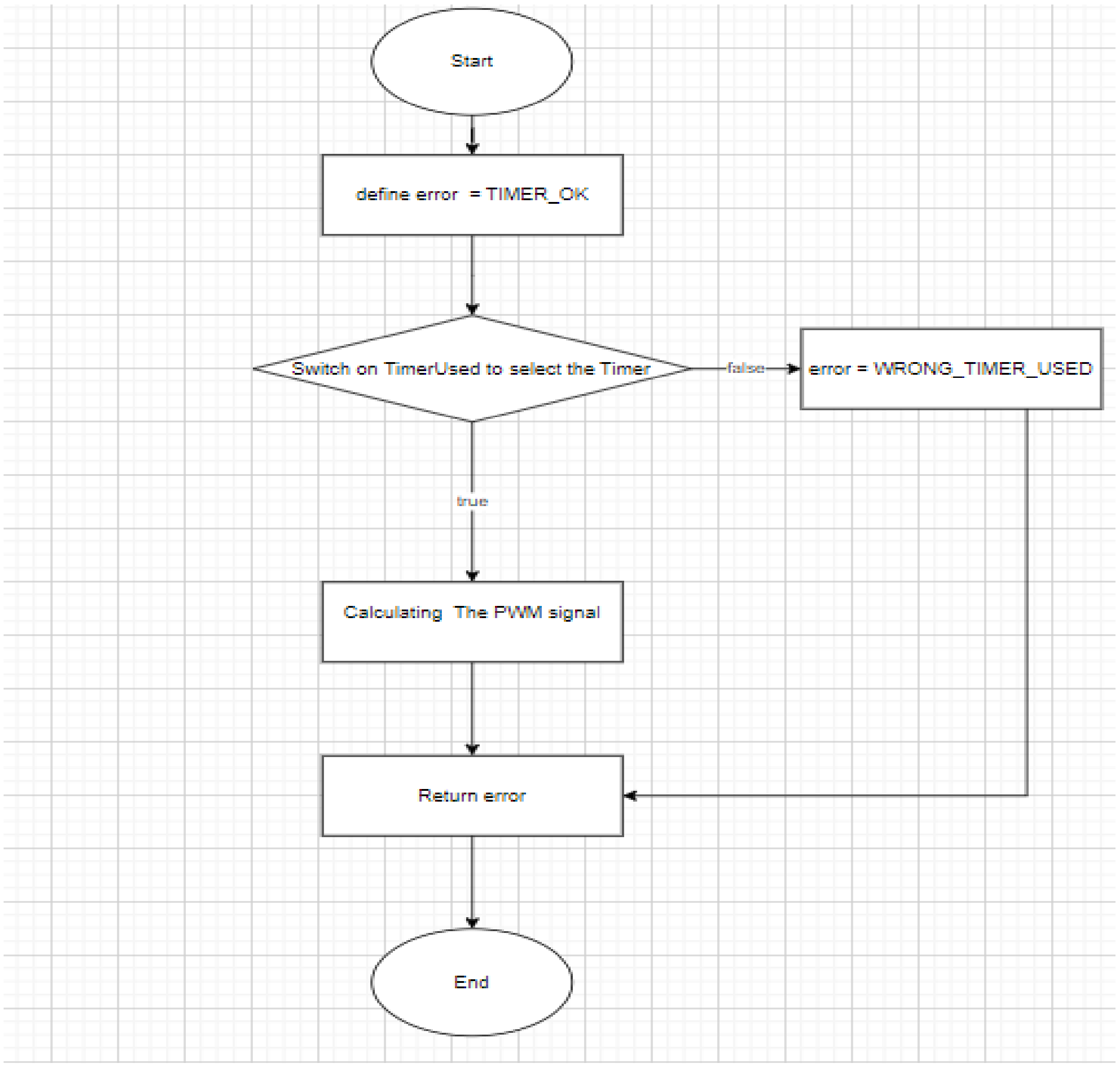
en_timerError_t TIMER_setTime(u8 u8_a_timerUsed, u32 u32_a_desiredTime);



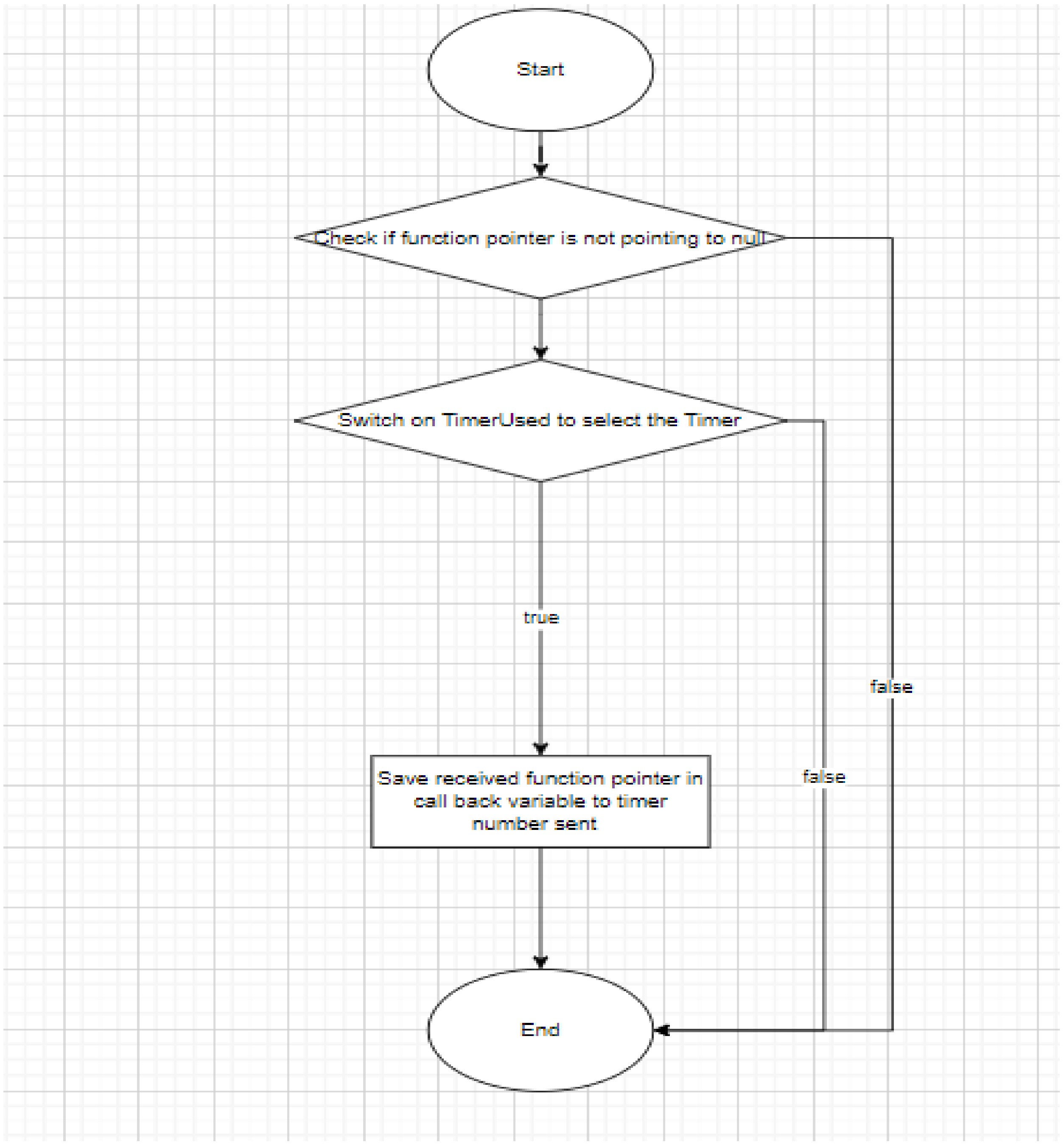
en_timerError_t TIMER_stop(u8 u8_a_timerUsed);



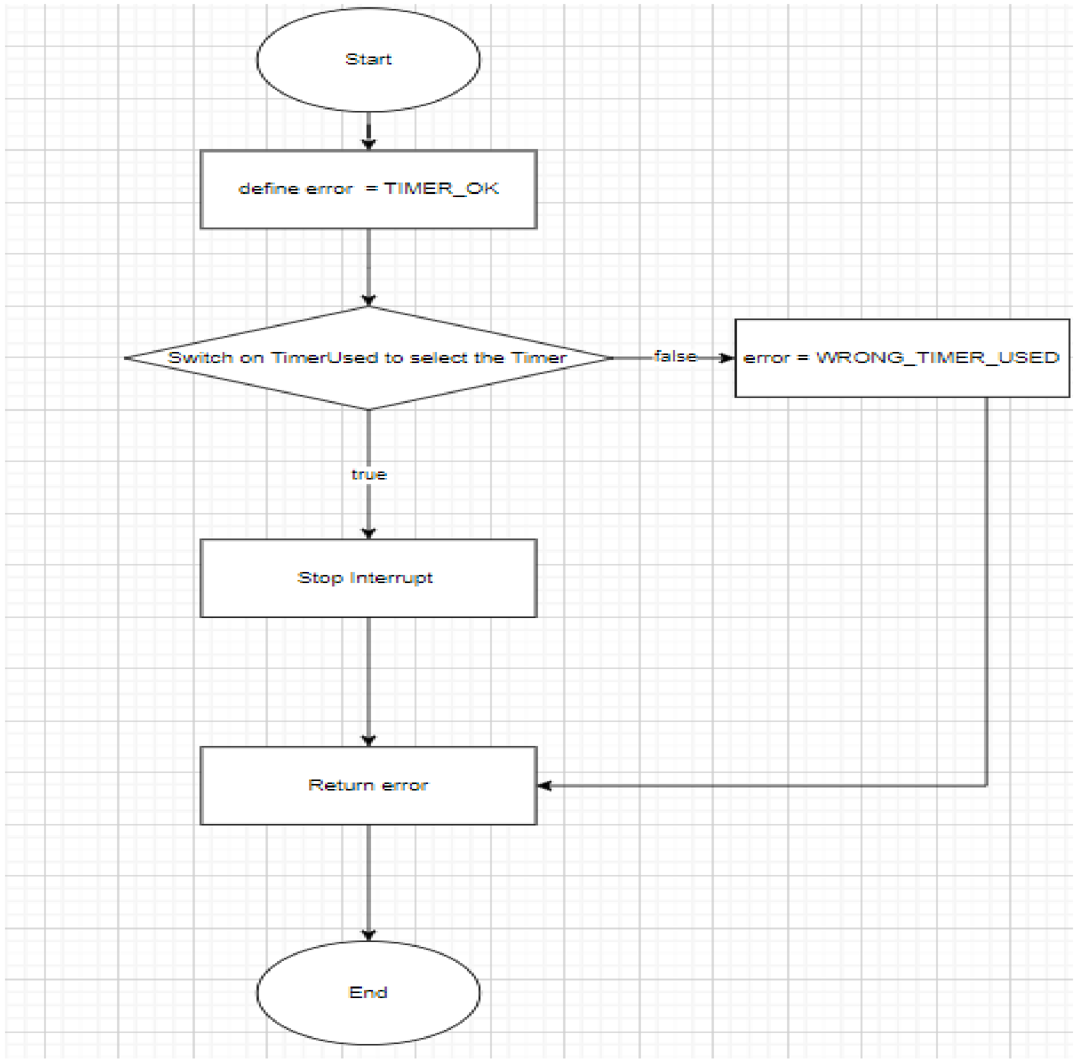
en_timerError_t TIMER_pwmGenerator(u8 u8_a_timerUsed, u32 u32_a_desiredDutyCycle);



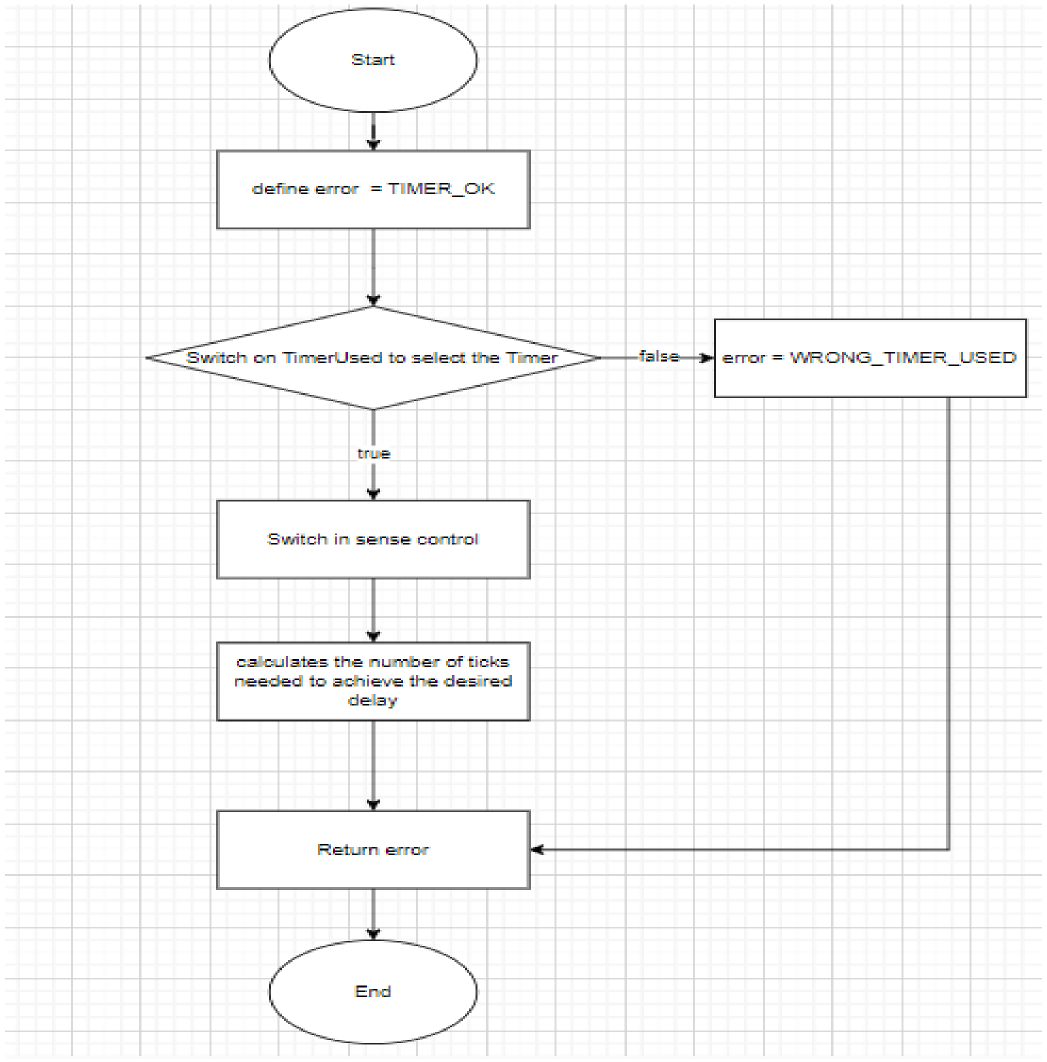
Void TIMER_setCallBack(u8 u8_a_timerUsed, void (*funPtr)(void));


Start
Check if function pointer is not pointing to null
Switch on TimerUsed to select the Timer
true
false
false
Save received function pointer in call back variable to timer number sent
End

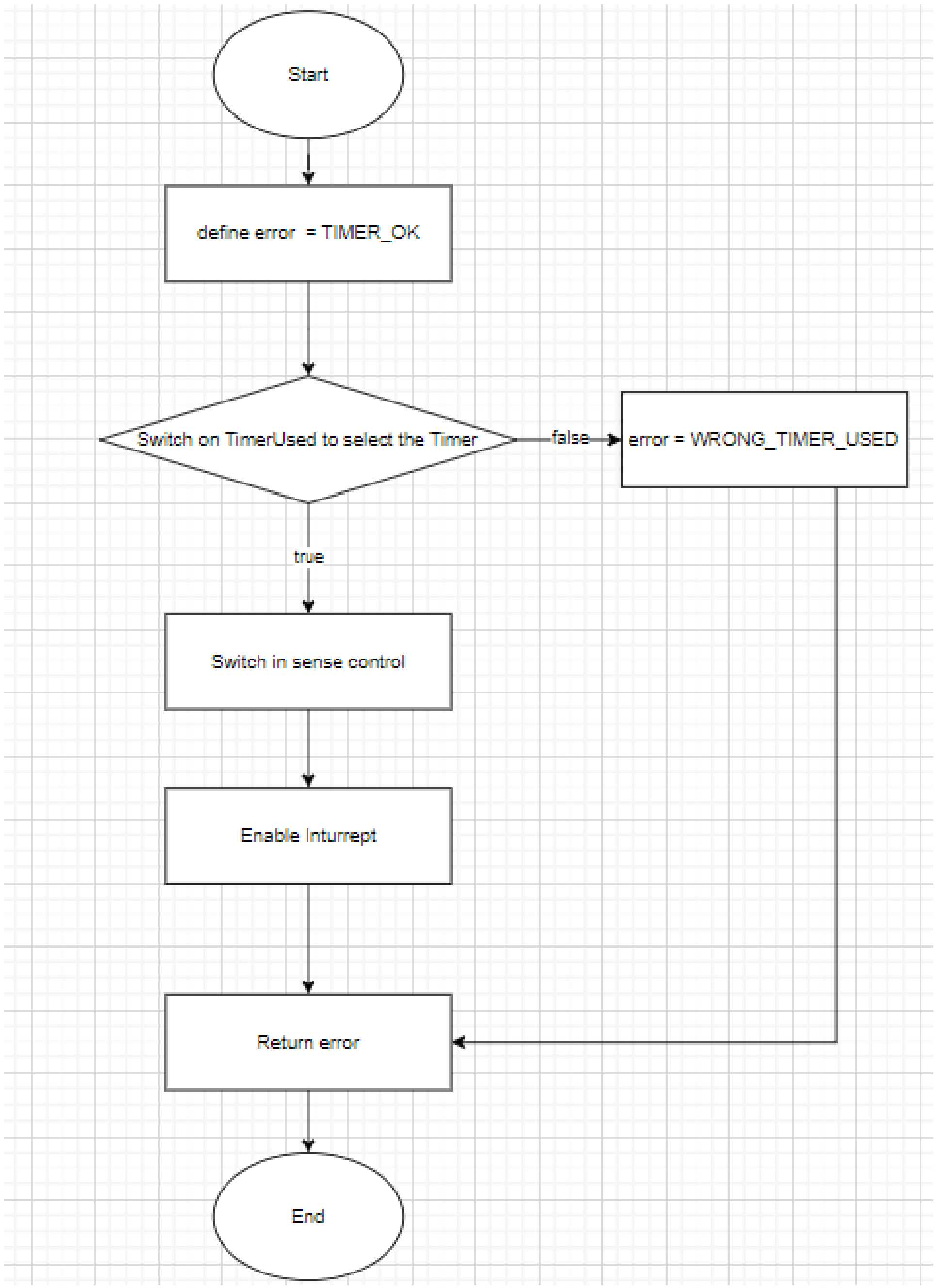
en_timerError_t TIMER_stopInterrupt(u8 u8_a_timerUsed);



en_timerError_t TIMER_delay(u8 u8_a_timerUsed, u32 u32_a_timeInMS);

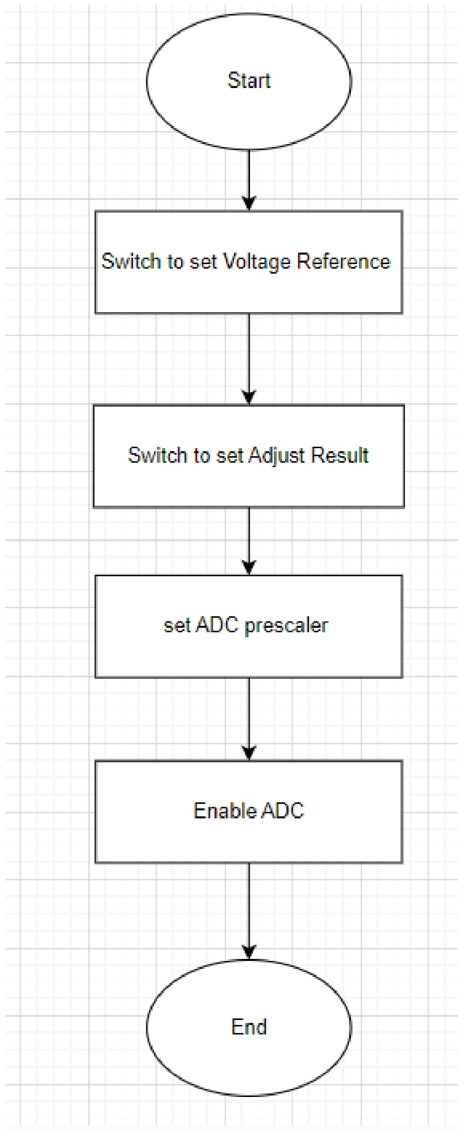

Start
define error = TIMER_OK
Switch on TimerUsed to select the Timer
false
error = WRONG_TIMER_USED
true
Switch in sense control
calculates the number of ticks needed to achieve the desired delay
Return error
End

en_timerError_t TIMER_enableInterrupt(u8 u8_a_timerUsed);

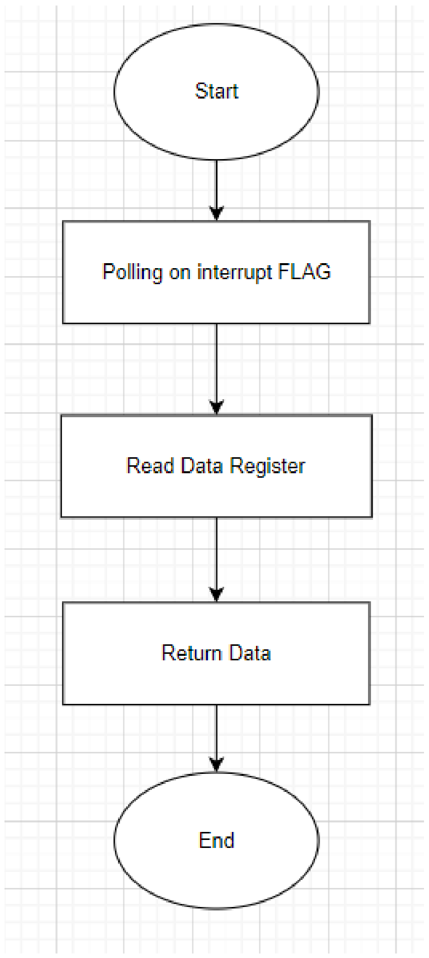


3. ADC

ADC_init()

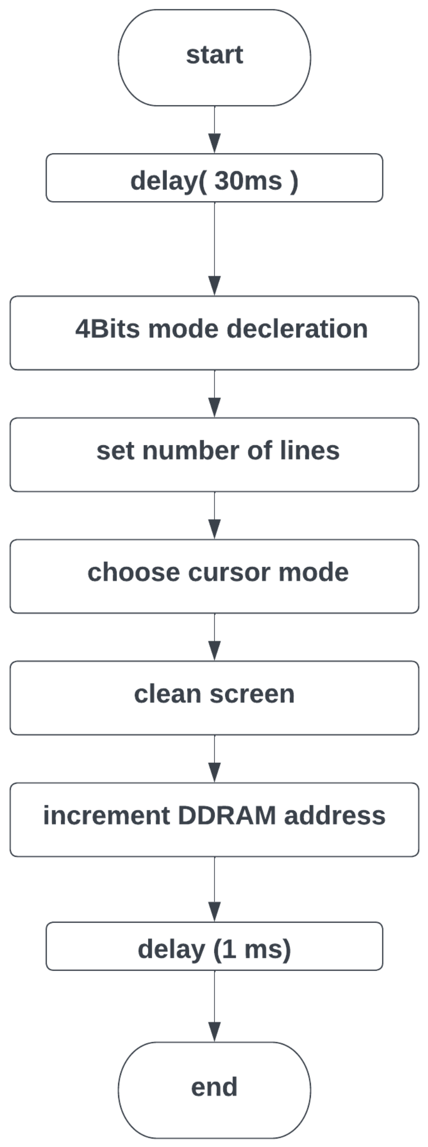


u16 ADC_read()

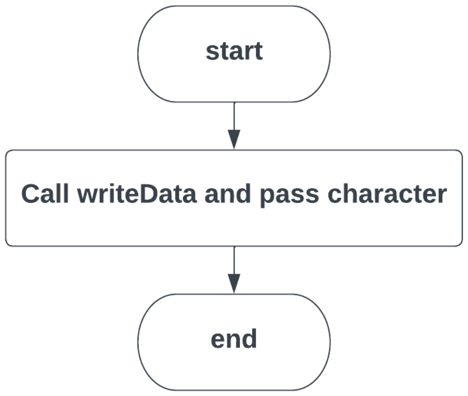

Start
Polling on interrupt FLAG
Read Data Register
Return Data
End

4. LCD

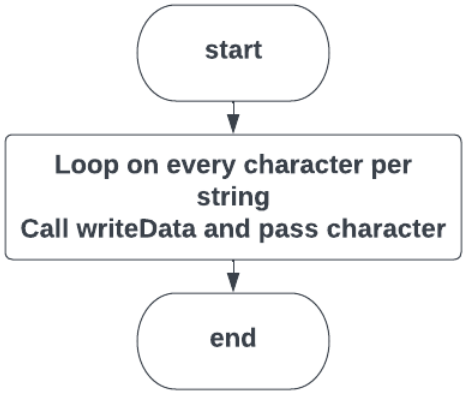
LCD_Init()



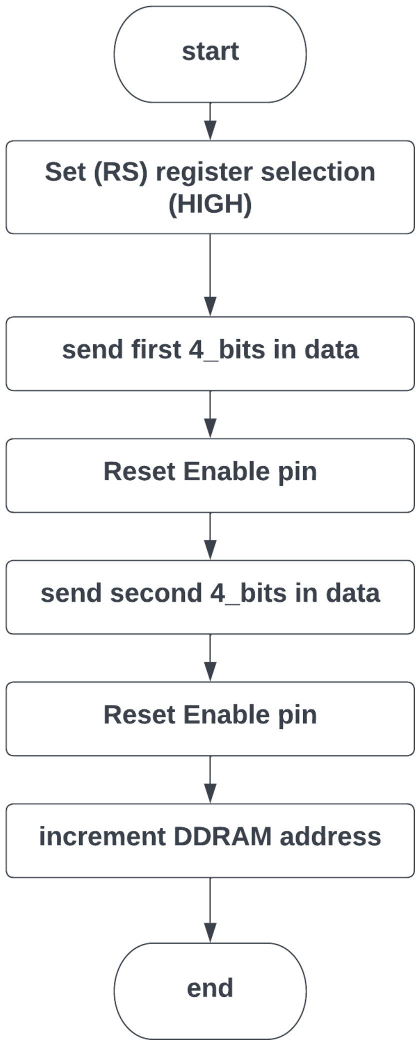
LCD_WriteChar(u8 ch)



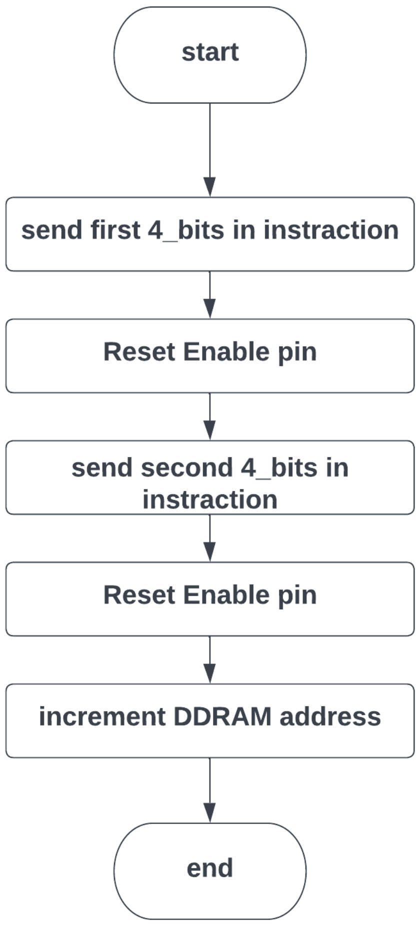
LCD_WriteString(u8*str)


start
Loop on every character per string
Call writeData and pass character
end

WriteData(u8 data)

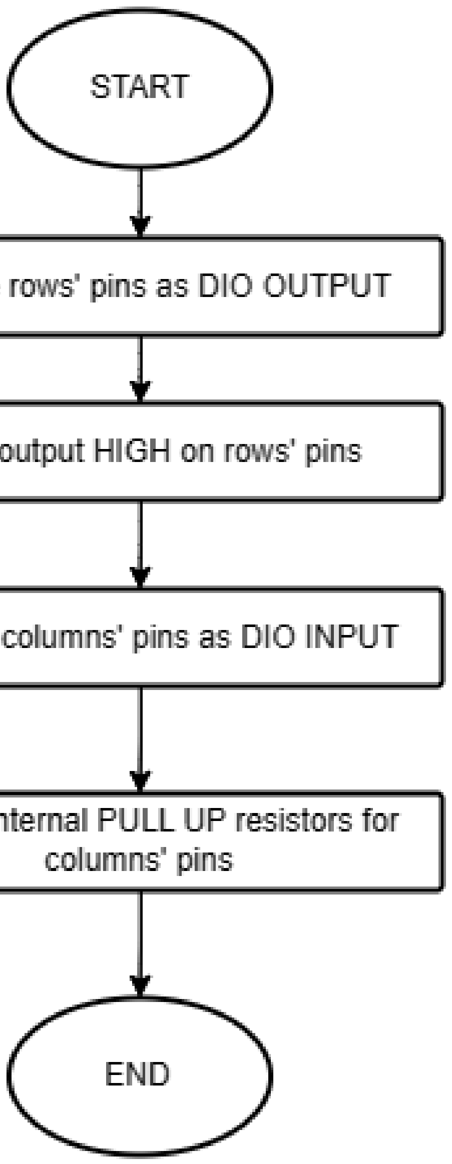

start
Set (RS) register selection (HIGH)
send first 4_bits in data
Reset Enable pin
send second 4_bits in data
Reset Enable pin
increment DDRAM address
end

WriteIns(u8 ins)

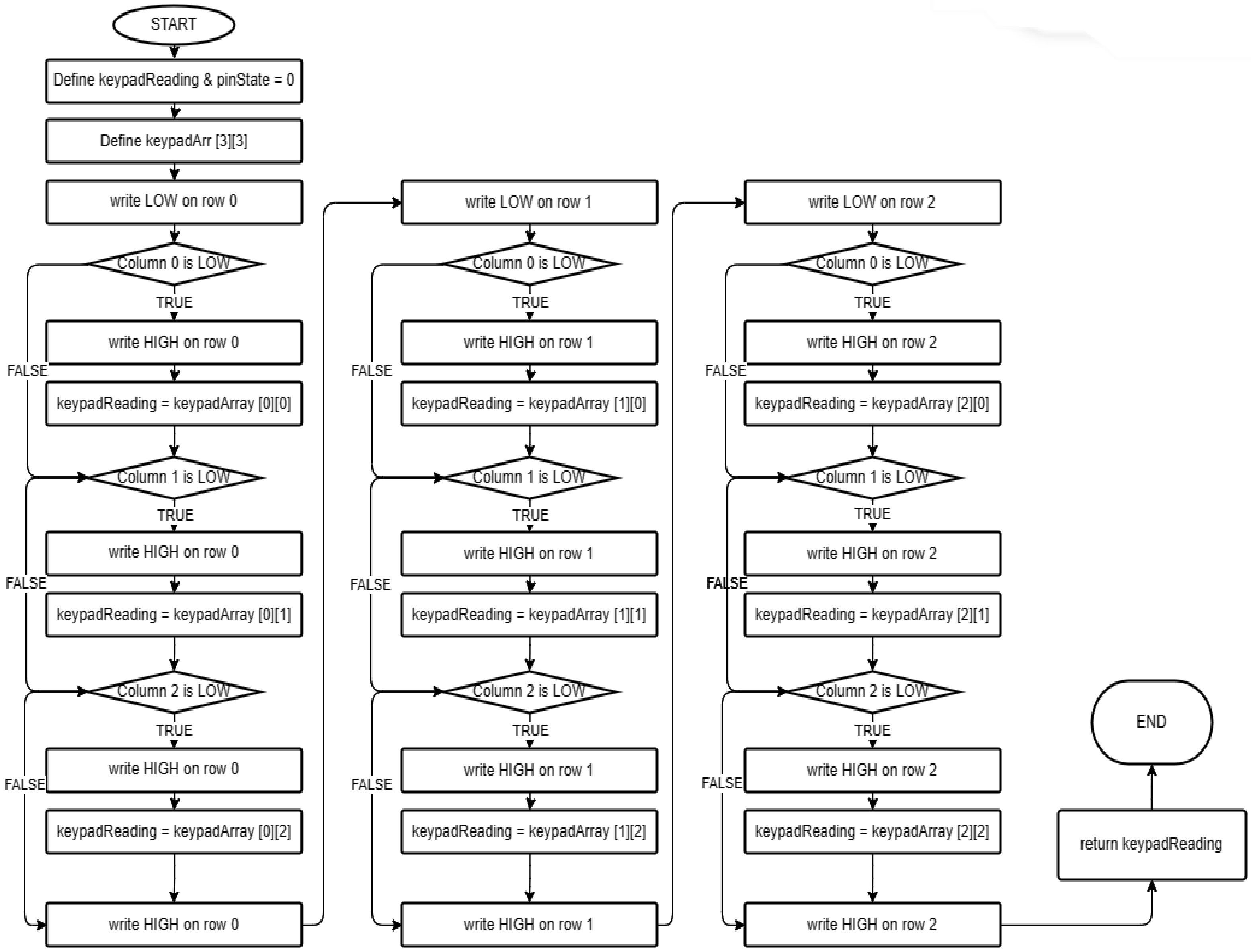

start
send first 4_bits in instraction
Reset Enable pin
send second 4_bits in instraction
Reset Enable pin
increment DDRAM address
end

5. KEYPAD

void KEYPAD_init(void);

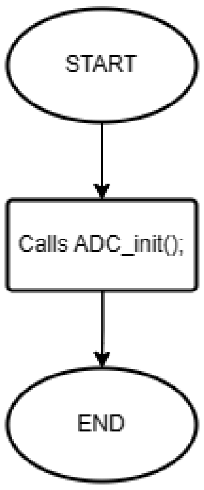

START
Initialize rows' pins as DIO OUTPUT
Write output HIGH on rows' pins
Initialize columns' pins as DIO INPUT
Enable internal PULL UP resistors for columns' pins
END

u8 KEYPAD_read(void)

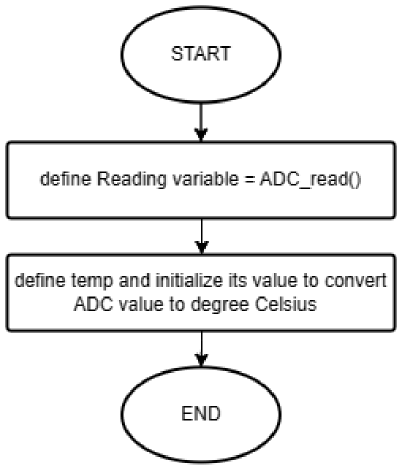

START
Define keypadReading & pinState = 0
Define keypadArr [3][3]
write LOW on row 0
Column 0 is LOW
TRUE
FALSE
write HIGH on row 0
keypadReading = keypadArray [0][0]
Column 1 is LOW
write HIGH on row 0
keypadReading = keypadArray [0][1]
Column 2 is LOW
write HIGH on row 0
keypadReading = keypadArray [0][2]
write HIGH on row 0
write LOW on row 1
write HIGH on row 1
keypadReading = keypadArray [1][0]
keypadReading = keypadArray [1][1]
keypadReading = keypadArray [1][2]
write LOW on row 2
write HIGH on row 2
keypadReading = keypadArray [2][0]
keypadReading = keypadArray [2][1]
keypadReading = keypadArray [2][2]
return keypadReading
END

6. LM35

void LM35_init(void)

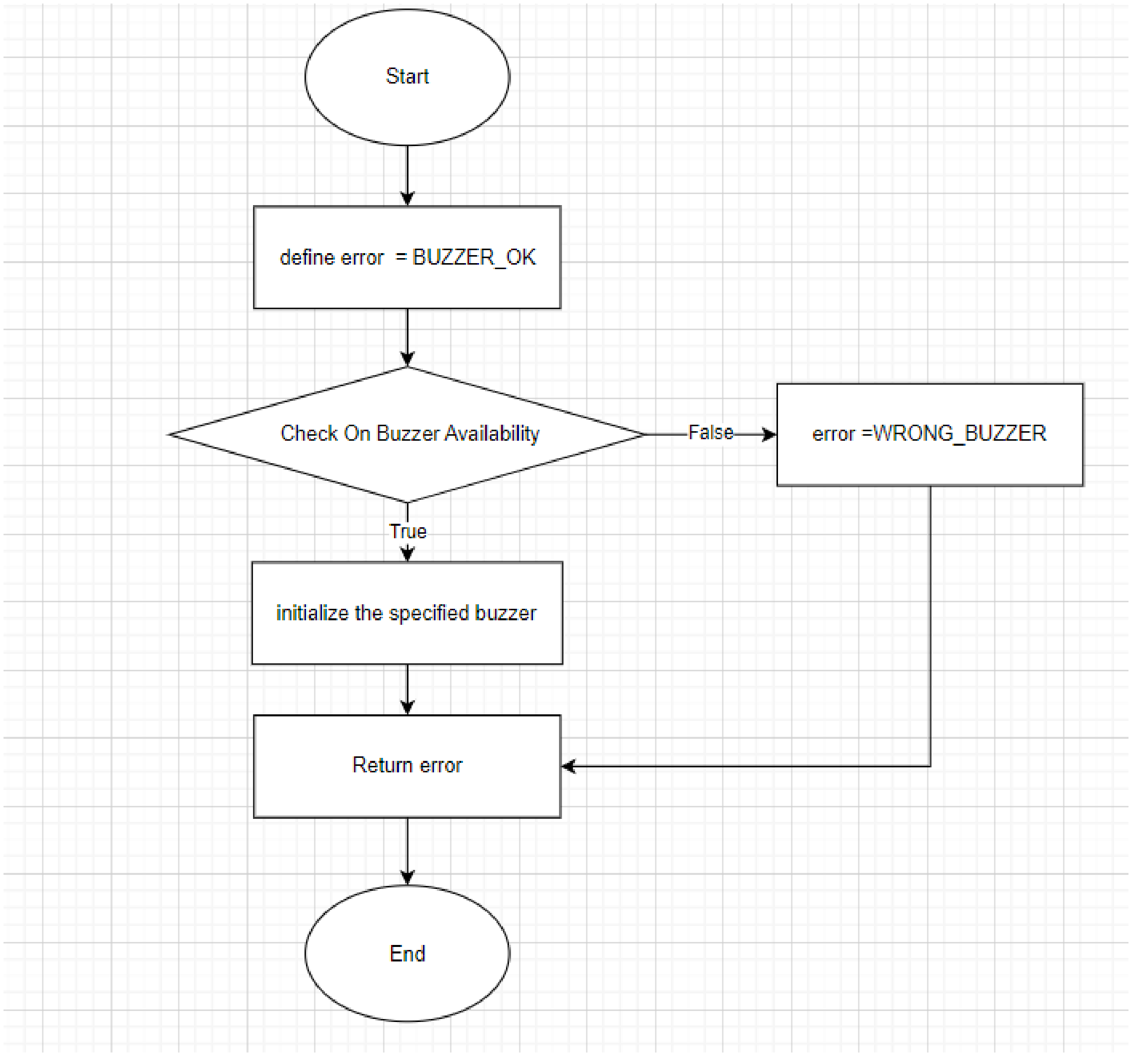


u16 LM35_read(void)

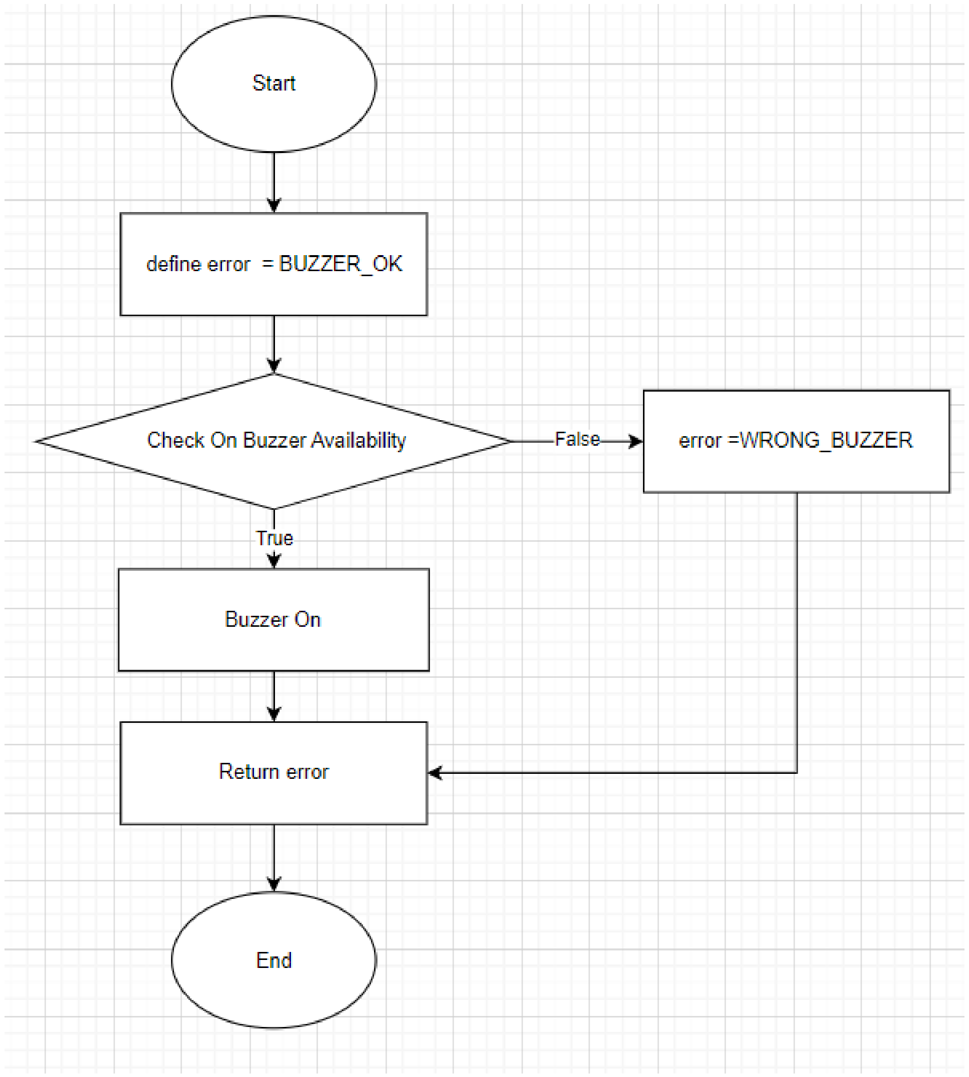

START
define Reading variable = ADC_read()
define temp and initialize its value to convert ADC value to degree Celsius
END

7. BUZZER

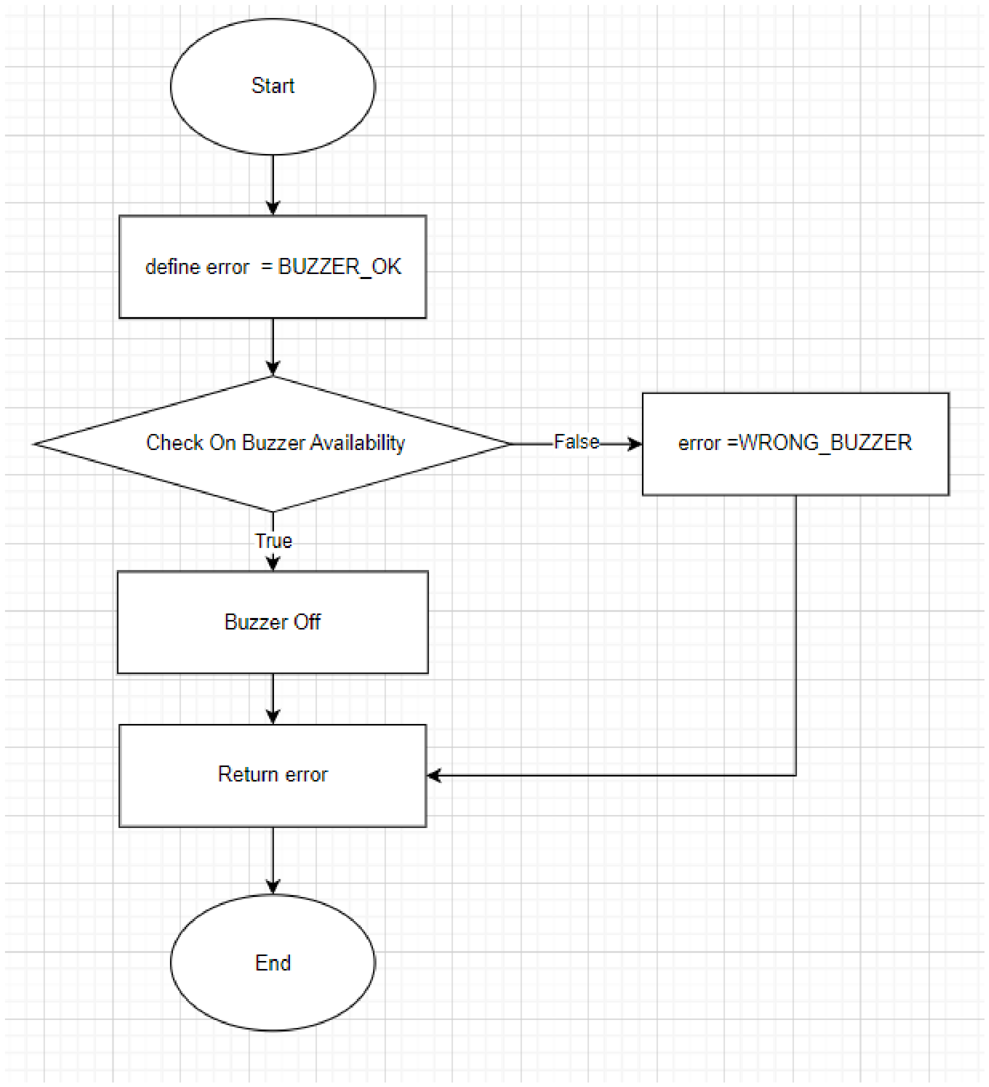
BUZZER_init()



BUZZER_on


Start
define error = BUZZER_OK
Check On Buzzer Availability
False
error =WRONG_BUZZER
True
Buzzer On
Return error
End

BUZZER_OFF


Start
define error = BUZZER_OK
Check On Buzzer Availability
False
error =WRONG_BUZZER
True
Buzzer Off
Return error
End