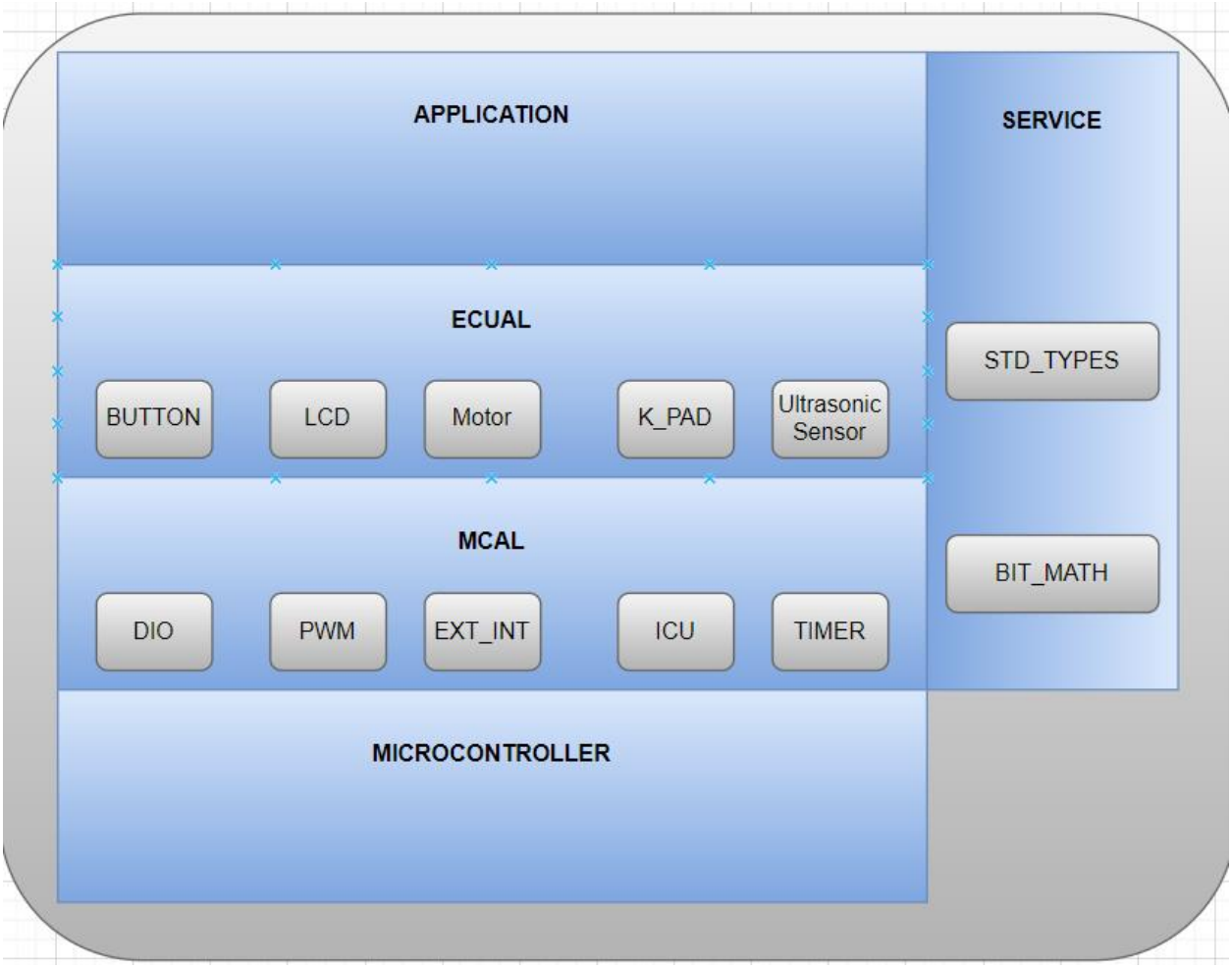
Obstacle Avoiding Car Project

BY: Momen Hassan

Contents

AIR CONDITIONER DESIGN

LAYERED ARCHITECTURE



INTRODUCTION

The project is a system designed for an autonomous car that uses an ATmega32 microcontroller as its core component. The car is equipped with four motors (M1, M2, M3, M4) and an ultrasonic sensor that is used to detect obstacles in its path.

The system is designed to be easy to use, with a simple interface consisting of a keypad button for starting and stopping the car, and a button for changing the default direction of rotation. The car is programmed to start at 0 speed and with the default rotation direction set to the right.

Once the car is started, the LCD display shows a centered message on line 1 indicating the default rotation direction, and on line 2 there is a prompt for the user to choose between right and left rotation. The user has 5 seconds to select their preferred rotation direction by pressing the PBUTTONO. The LCD display will update accordingly to show the selected rotation direction.

After the 5-second period, the default rotation direction is set, and the car will start moving 2 seconds later. The car is programmed to move forward at 30% speed for the first 5 seconds, and then increase its speed to 50% as long as there are no obstacles detected that are closer than 70 centimeters. The LCD display will show the speed and direction of movement, as well as the distance of any detected obstacles.

If an obstacle is detected within a range of 30 to 70 centimeters, the car will slow down to 30% speed, and the LCD display will be updated accordingly. If the obstacle is closer than 30 centimeters, the car will stop and rotate 90 degrees to the right or left (depending on the selected rotation direction) and the LCD display will be updated accordingly. If the obstacle is less than 20 centimeters away, the car will stop, move backwards at 30% speed until the distance is greater than 20 and less than 30 centimeters, and then perform the appropriate action based on the obstacle's location.

As a bonus feature, the car will check if it has rotated 360 degrees without finding any obstacle further than 20 centimeters away. If it has, it will stop and the LCD display will be updated. The car will also check for changes in obstacle distance every 3 seconds and move in the direction of the furthest object.

Overall, the system provides an autonomous car that can navigate through its surroundings and avoid obstacles using a combination of sensors and programmed behaviors. The simple interface and modular software architecture make the system easy to use and customize for different applications.

It consists of four layers:

1-Application Layer: The application layer is responsible for the higher-level logic of the project. It includes the algorithms and decision-making processes that control the robot's movements based on the input received from the Ultrasonic sensor, button, and keypad. The application layer uses the services provided by the lower-level layers to control the hardware components of the robot.

2-ECUAL: The ECUAL layer provides a high-level interface to the external hardware components used in the project, such as the Ultrasonic sensor, button, keypad, and LCD display. It abstracts the low-level details of these components and provides a simplified interface that can be used by the application layer. The ECUAL layer also includes bitmath and std_types libraries that provide low-level bit manipulation and standard data types, respectively.

3-MCAL: The MCAL layer provides an interface to the microcontroller's hardware components, such as the DIO (Digital Input/Output), Timer, and ADC (Analog-to-Digital Converter) modules. It provides a low-level interface to these components that can be used by the ECUAL layer to control the external hardware components. The MCAL layer also includes bitmath and std_types libraries for low-level bit manipulation and standard data types, respectively.

4-Service Layer: The service layer provides a set of common functions that can be used by the application layer to control the robot's movements. These functions include controlling the speed and direction of the motors, reading input from the Ultrasonic sensor, and updating the LCD display with relevant information. The service layer abstracts the low-level details of the MCAL and ECUAL layers and provides a simplified interface that can be used by the application layer. It also uses bitmath and std_types libraries for low-level bit manipulation and standard data types, respectively.

Overall, the project involves designing a car that can move forward, backward, and rotate in both directions while avoiding obstacles. The ECUAL layer abstracts the low-level details of the external hardware components, such as the Ultrasonic sensor, button, keypad, and LCD display. The MCAL layer provides a low-level interface to the microcontroller's hardware components, which can be used by the ECUAL layer. The service layer provides a set of common functions that can be used by the application layer to control the robot's movements. The application layer uses the services provided by the lower-level layers to control the hardware components of the robot and make high-level decisions about its movements.

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.

LCD: This display is used to show the current status of the car and distance readings from the ultrasonic sensor. It is controlled by the application layer and communicates with the microcontroller using supporting drivers.

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.

Motor: This module controls the four motors (M1, M2, M3, M4) of the car using the ATmega32 microcontroller. It receives commands from the application layer to move the car in different directions at different speeds.

Ultrasonic: This module handles the ultrasonic sensor and communicates with the application layer to provide distance readings of objects in front of the car. It also triggers the car to stop or change direction based on the distance readings.

PWM: is a method of controlling analog circuits using digital signals. It involves producing a square wave with a fixed frequency and varying the width of the pulse to control the average voltage level. PWM is commonly used in controlling the speed of DC motors, brightness of LEDs, and voltage level of audio signals.

ICU: is a peripheral in microcontrollers that allows for the measurement of various properties of incoming signals. It works by capturing the timestamp of rising or falling edges of the input signal and storing them in registers. This timestamp data can then be used to calculate properties such as frequency, duty cycle, and pulse width of the signal.

External Interrupt: is a feature in microcontrollers that allows an external signal to interrupt the normal program execution and perform a specific task. It is a hardware feature that is triggered by a change in the state of an external signal, such as a button press or a sensor trigger.

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. 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

4. 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

5.EXTERNAL INTERRUPTS

EN_extintError_t EXTINT_Init (uint8_t intNumber);

Function Name	**EXTINT_Init**
Description	Initializes an external interrupt.
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	The function takes one input parameter called "intNumber". This parameter is of type uint8_t and is presumably used to specify which external interrupt to initialize.
Parameters (out)	None
Return Value	EN_extintError_t

void EXTINT_setCallBackInt (uint8_t intNumber, void (*funPtr) (void));

Function Name	**EXTINT_setCallBackInt**
Description	Sets a callback function to be executed when an external interrupt occurs.
Sync\Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	The function takes two input parameters: "intNumber" of type uint8_t, which presumably specifies which external interrupt to set the callback function for, and "funPtr" of type void pointer, which is a pointer to the callback function to be executed
Parameters (out)	None
Return Value	EN_extintError_t

6.Motor

void motor_init(void);

Function Name	**motor_init**
Description	Initializes the motor driver
Sync\Async	Synchronous
Reentrancy	Non-reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

void motor_set_speed(uint8_t speed);

Function Name	**motor_set_speed**
Description	Sets the speed of the motor
Sync\Async	Synchronous
Reentrancy	Non-reentrant
Parameters (in)	speed: the desired speed (0-100)
Parameters (out)	None
Return Value	None

void motor_set_direction (uint8_t direction);

Function Name	**motor_set_direction**
Description	Sets the direction of the motor
Sync\Async	Synchronous
Reentrancy	Non-reentrant
Parameters (in)	direction: the desired direction
Parameters (out)	None
Return Value	None

7. Ultrasonic

void ultrasonic_init(void)

Function Name	**ultrasonic_init**
Description	Initializes the ultrasonic sensor driver
Sync\Async	Synchronous
Reentrancy	Non-reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

void ultrasonic_trigger_measurement (void);

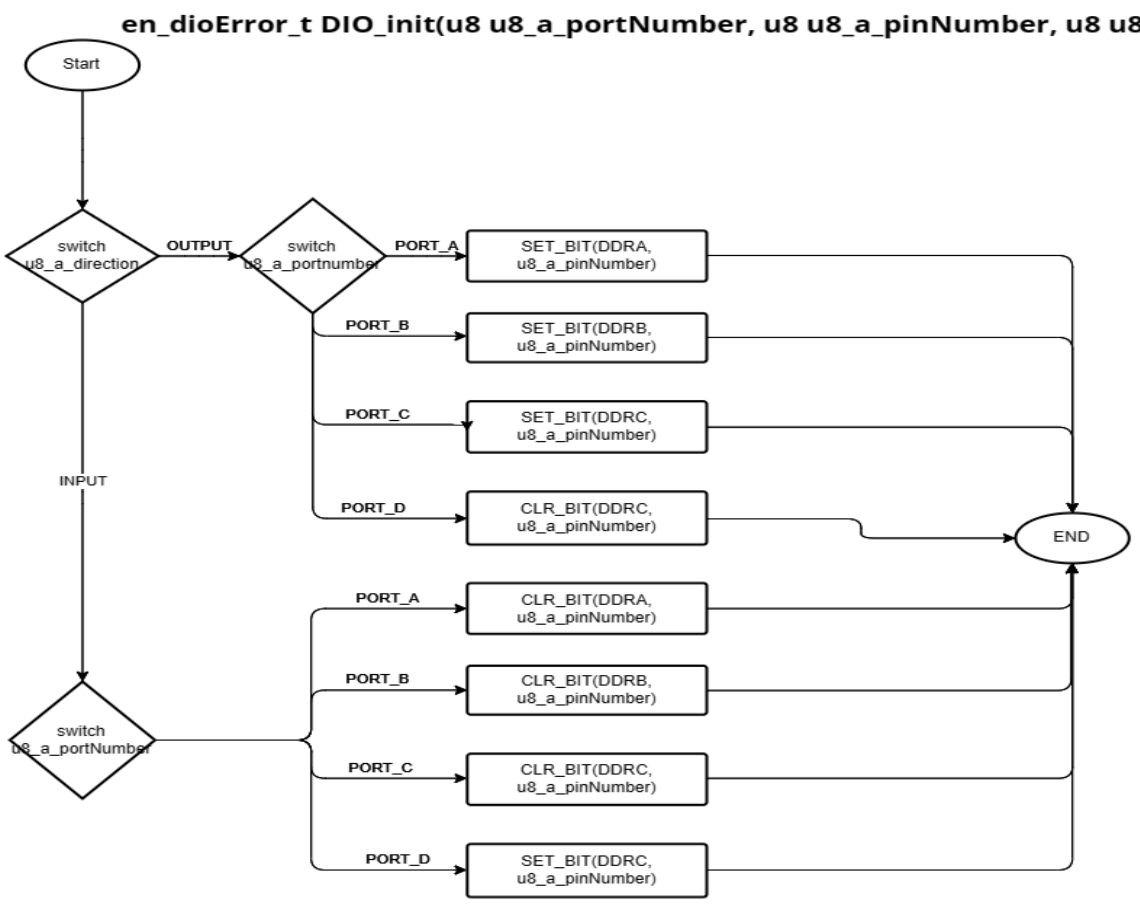
Function Name	**ultrasonic_trigger_measurement**
Description	Triggers a distance measurement
Sync\Async	Synchronous
Reentrancy	Non-reentrant
Parameters (in)	None
Parameters (out)	None
Return Value	None

Ultrasonic_Status ultrasonic_get_distance (uint8_t distance);

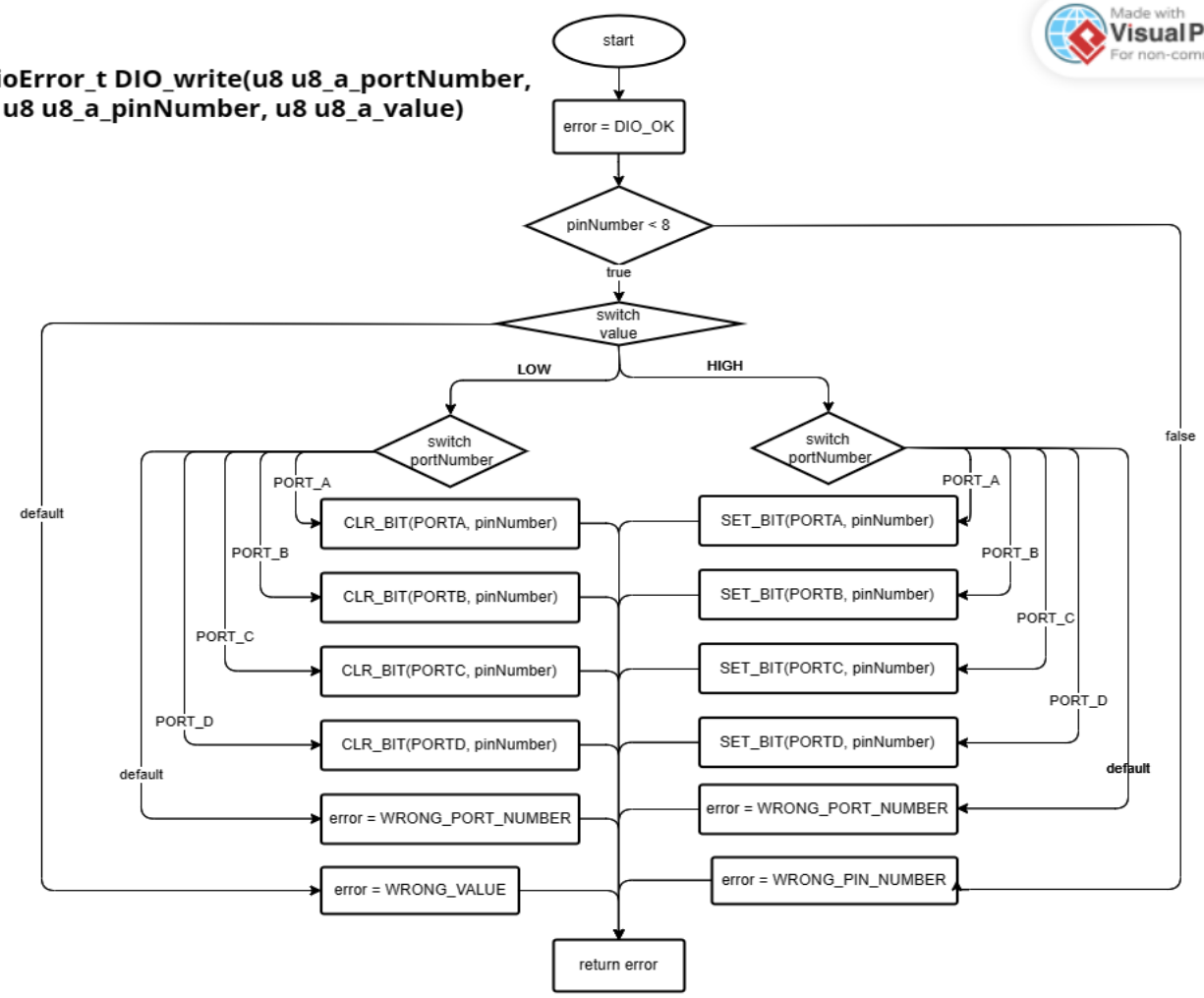
Function Name	**ultrasonic_get_distance**
Description	Gets the distance measured by the ultrasonic sensor
Sync\Async	Synchronous
Reentrancy	Non-reentrant
Parameters (in)	distance: a pointer to a variable to store the distance
Parameters (out)	None
Return Value	ULTRASONIC_OK if successful, ULTRASONIC_ERROR otherwise

FUNCTIONS' FLOWCHARTS

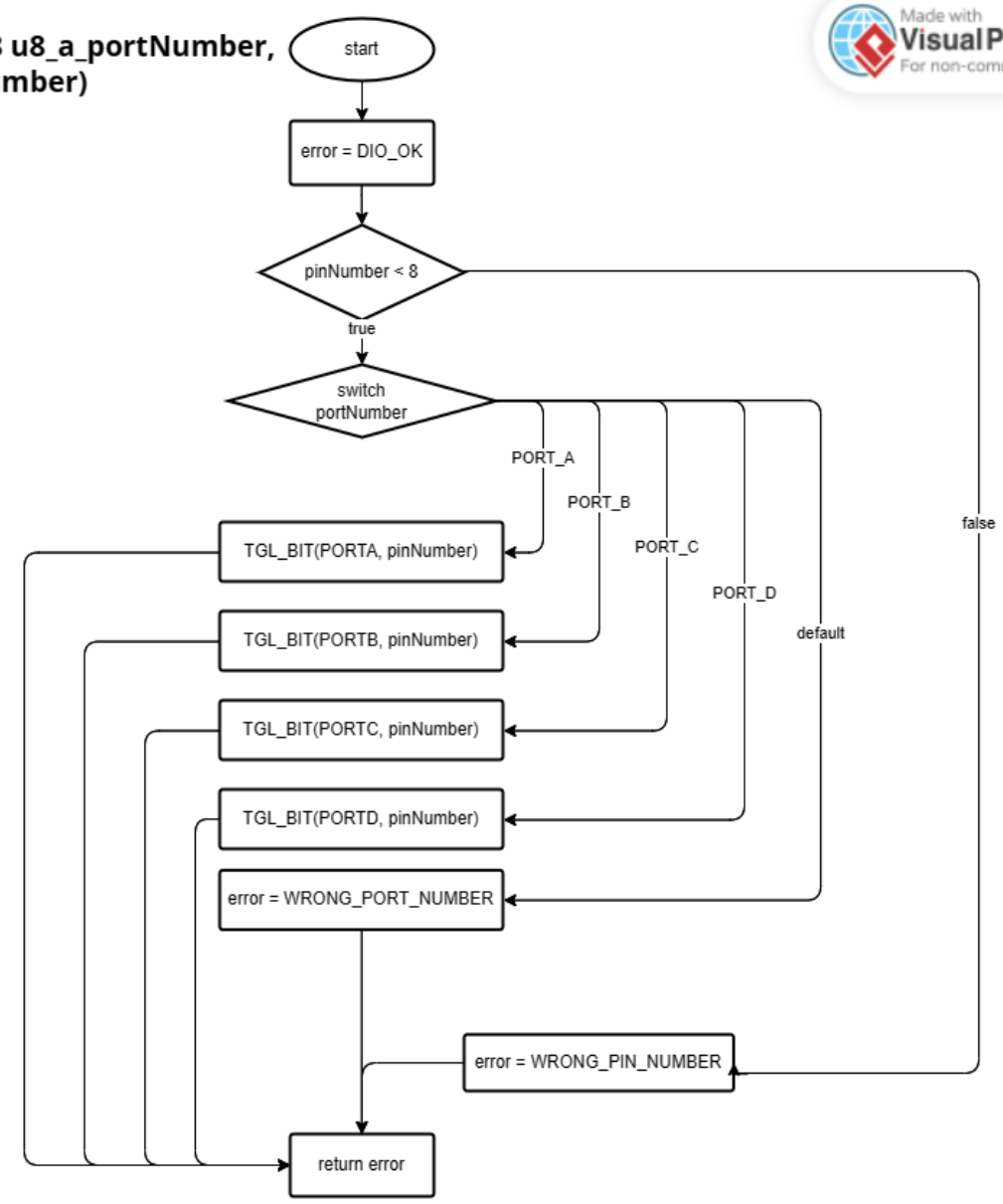
1. DIO



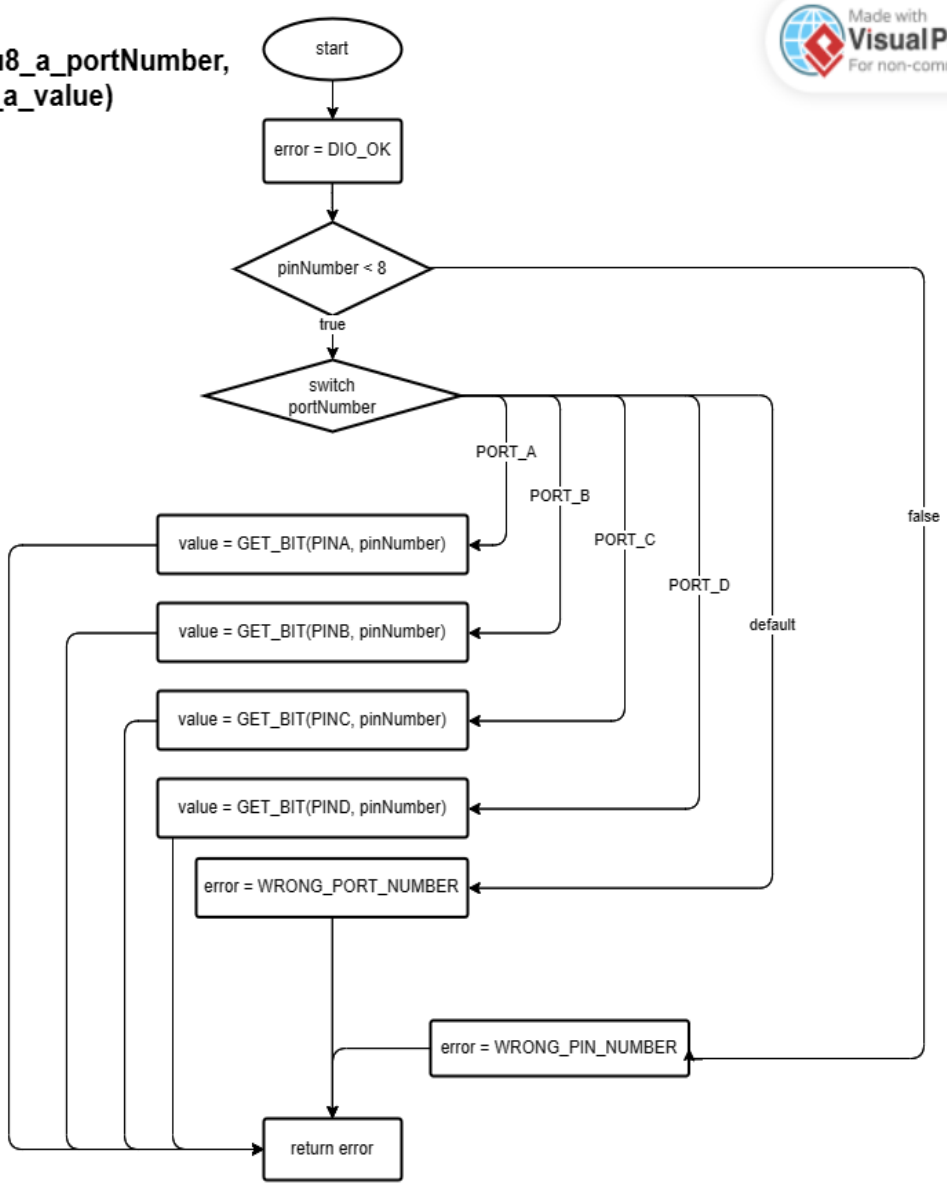
en_dioError_t DIO_write(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 u8_a_value)



en_dioError_t DIO_toggle(u8 u8_a_portNumber, u8 u8_a_pinNumber)

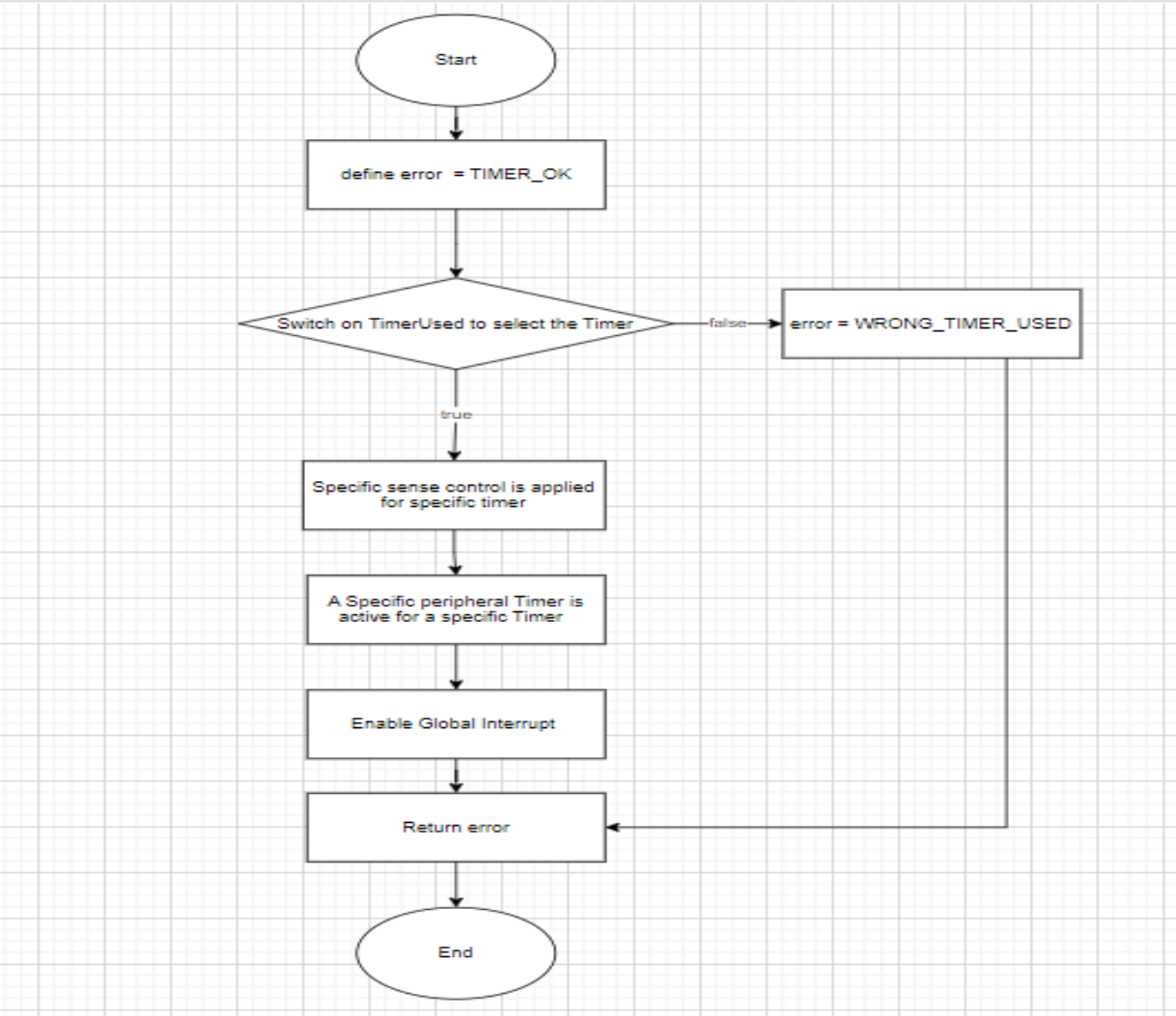


en_dioError_t DIO_read(u8 u8_a_portNumber, u8 u8_a_pinNumber, u8 *u8_a_value)

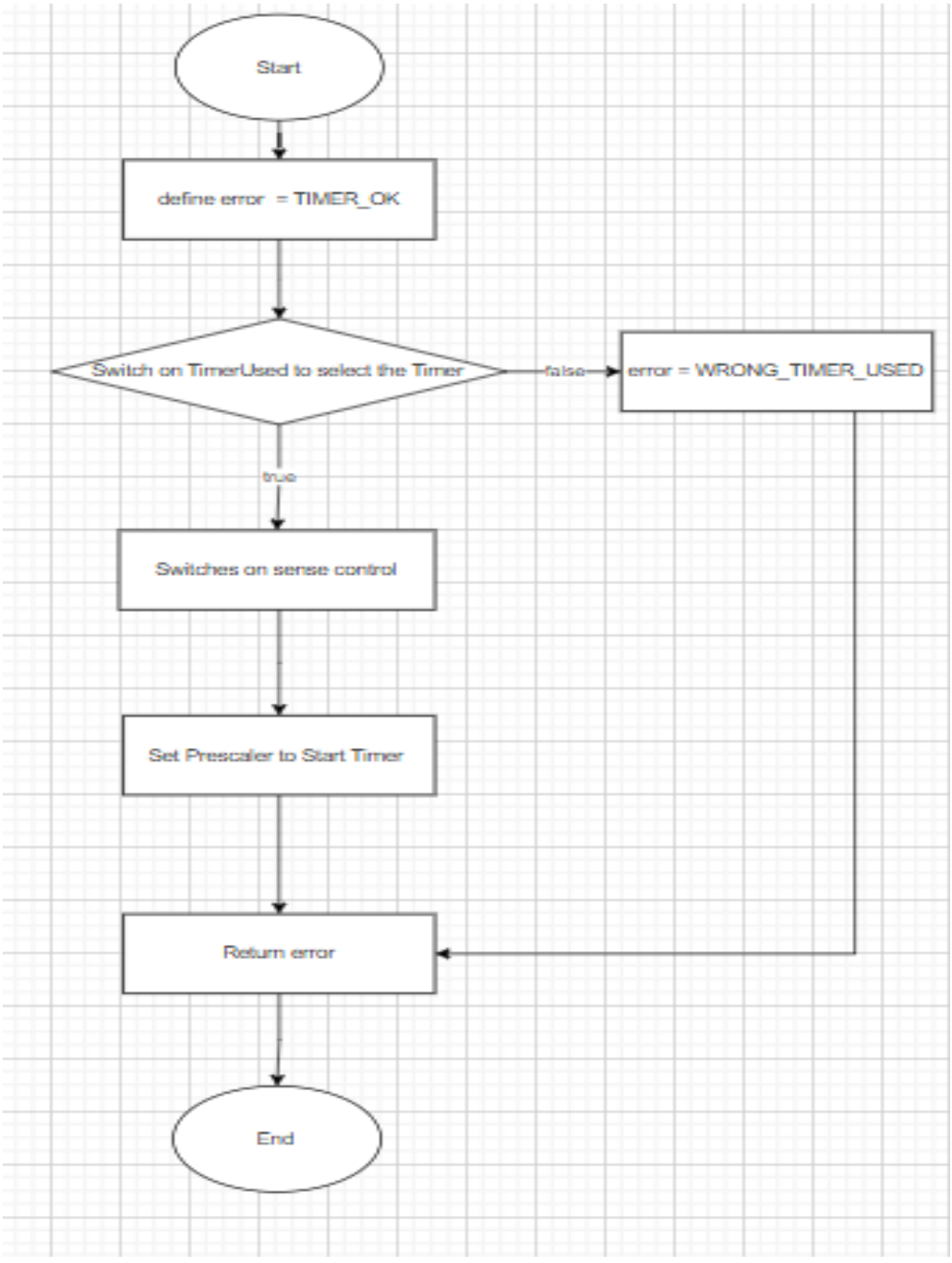


2. TIMERS

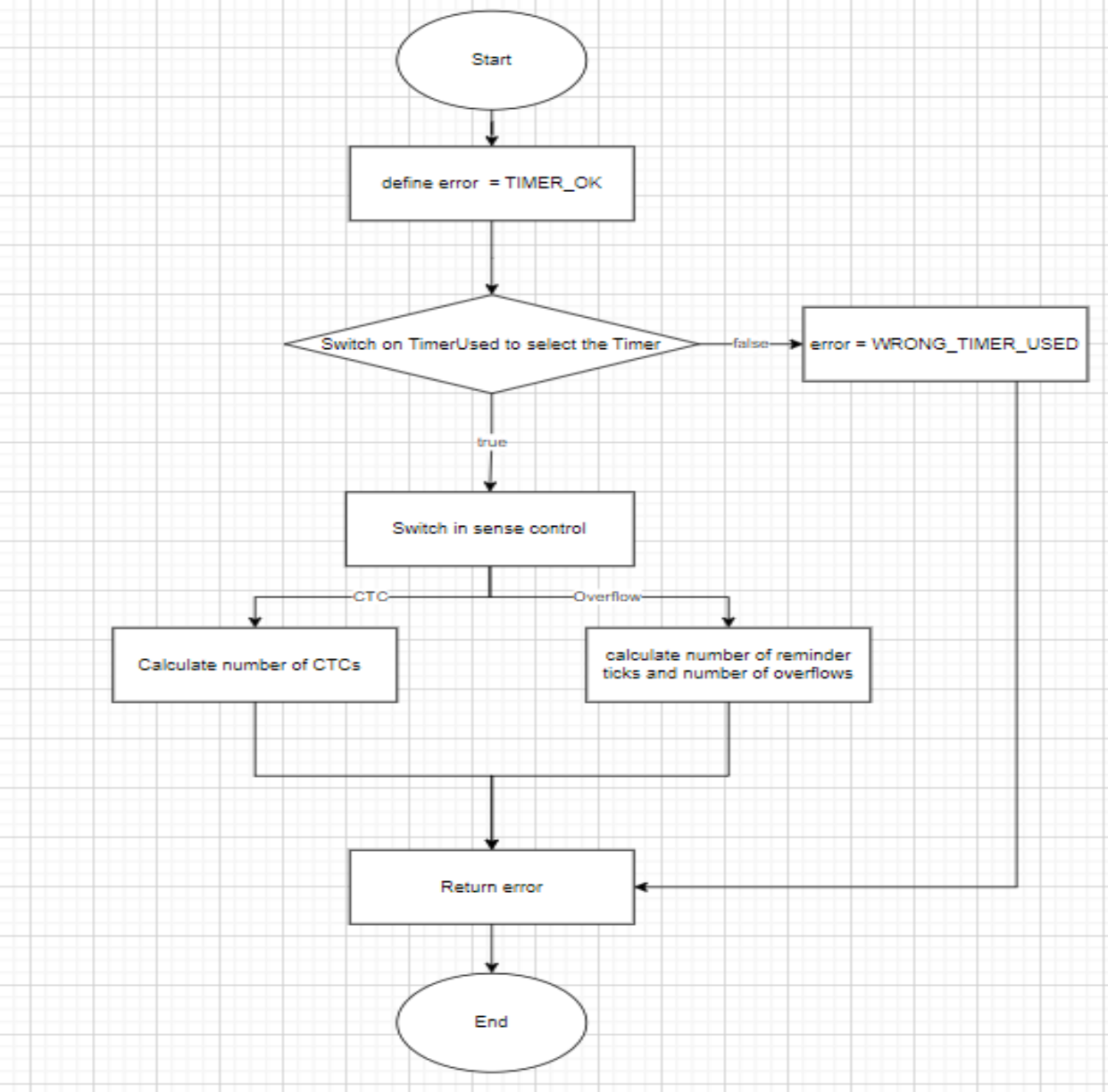
en_timerError_t TIMER_init(u8 u8_a_timerUsed);



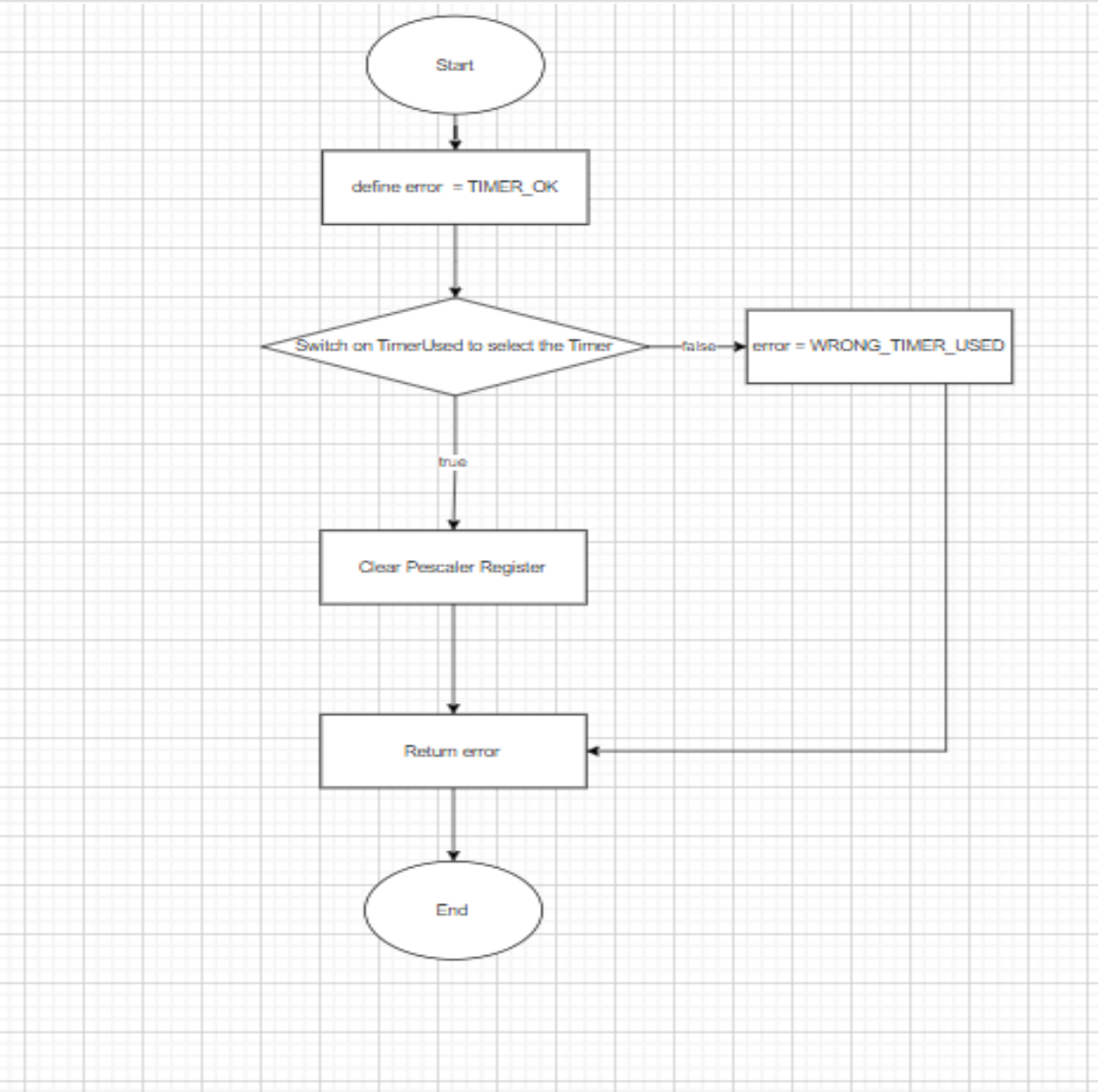
en_timerError_t TIMER_start(u8 u8_a_timerUsed);



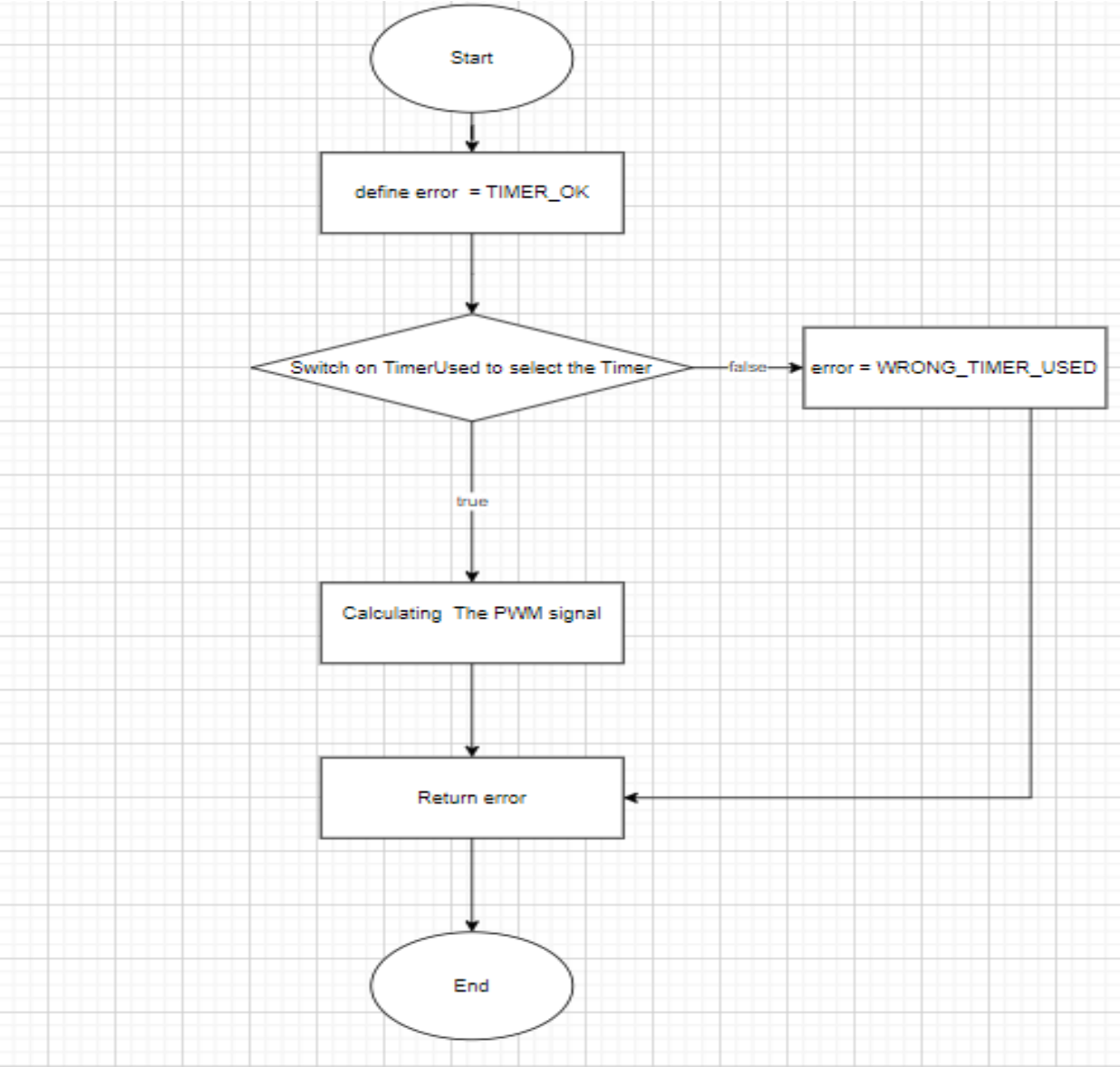
en_timerError_t TIMER_setTime(u8 u8_a_timerUsed, u32 u32_a_desiredTime);



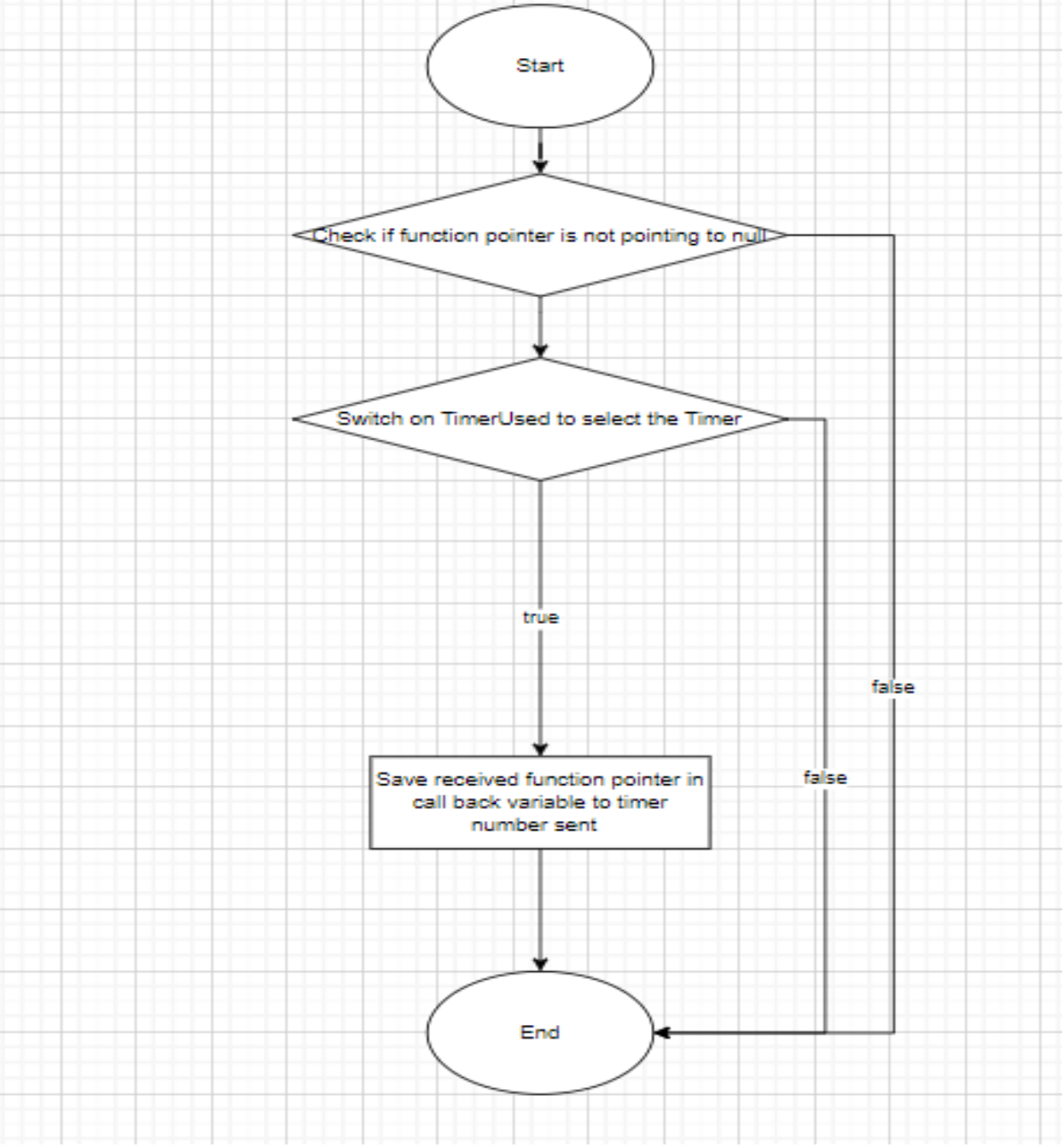
en_timerError_t TIMER_stop(u8 u8_a_timerUsed);



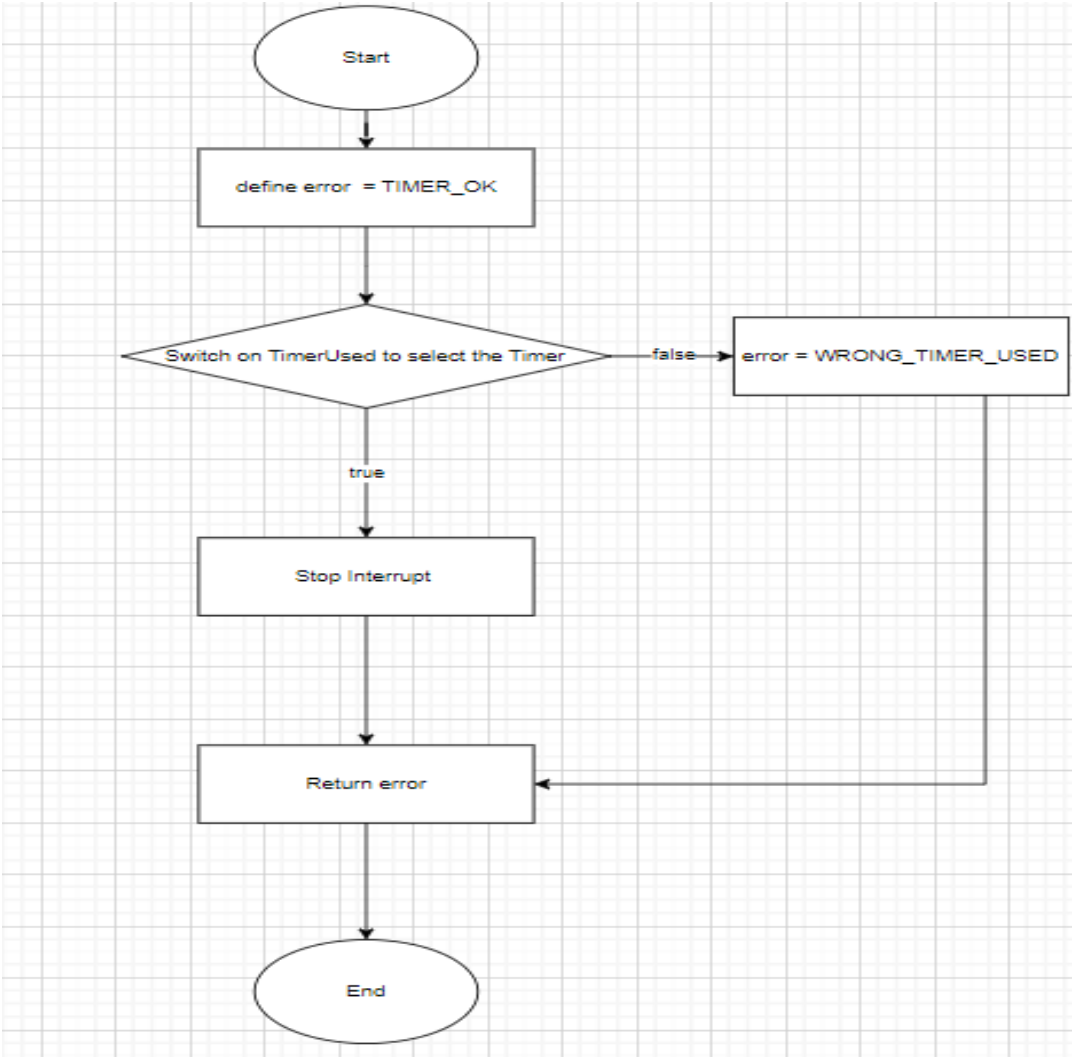
en_timerError_t TIMER_pwmGenerator(u8 u8_a_timerUsed, u32 u32_a_desiredDutyCycle);



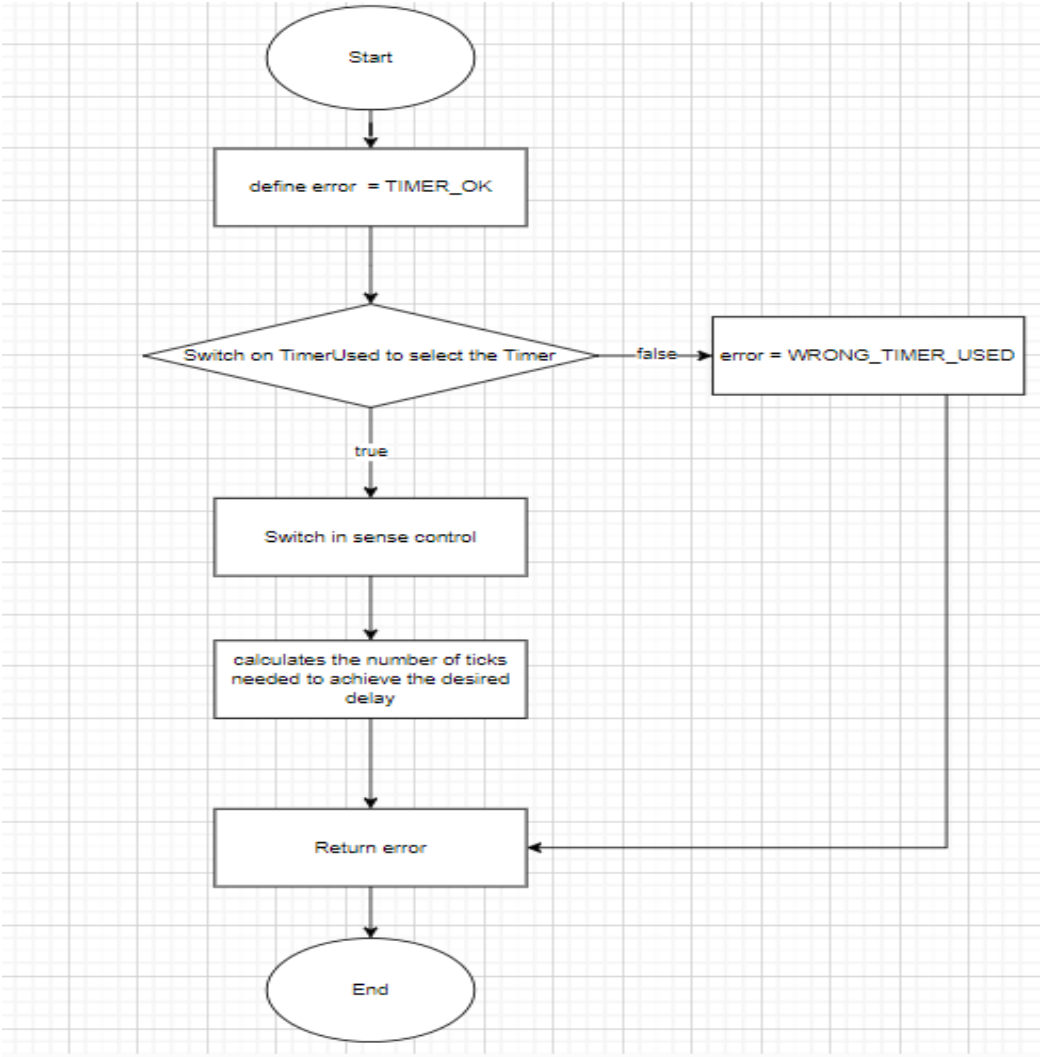
Void TIMER_setCallBack(u8 u8_a_timerUsed, void (*funPtr)(void));



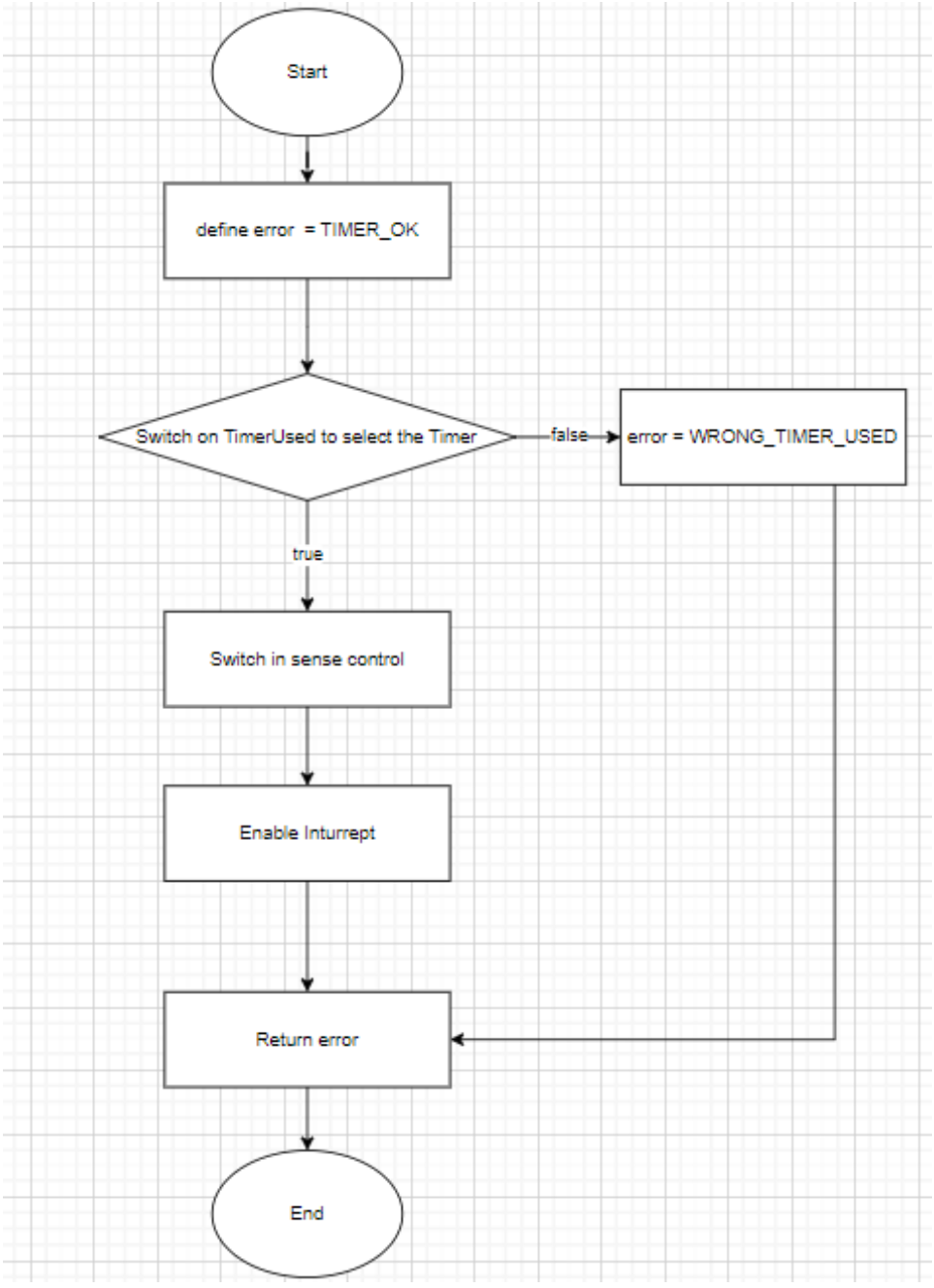
en_timerError_t TIMER_stopInterrupt(u8 u8_a_timerUsed);



en_timerError_t TIMER_delay(u8 u8_a_timerUsed, u32 u32_a_timeInMS);

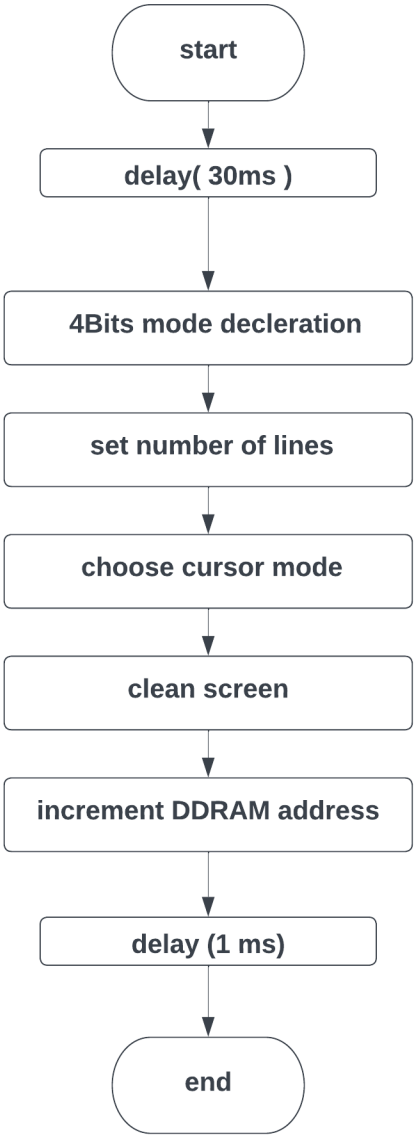


en_timerError_t TIMER_enableInterrupt(u8 u8_a_timerUsed);

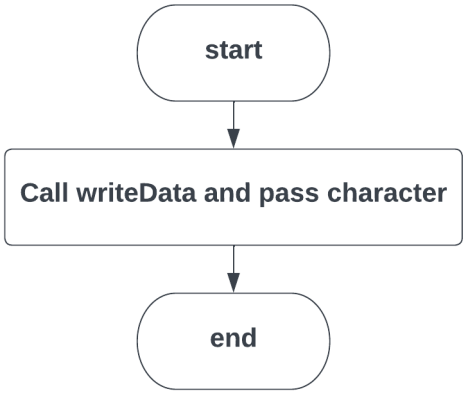


3. LCD

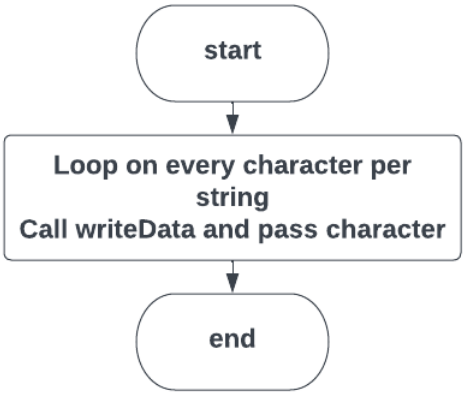
LCD_Init()



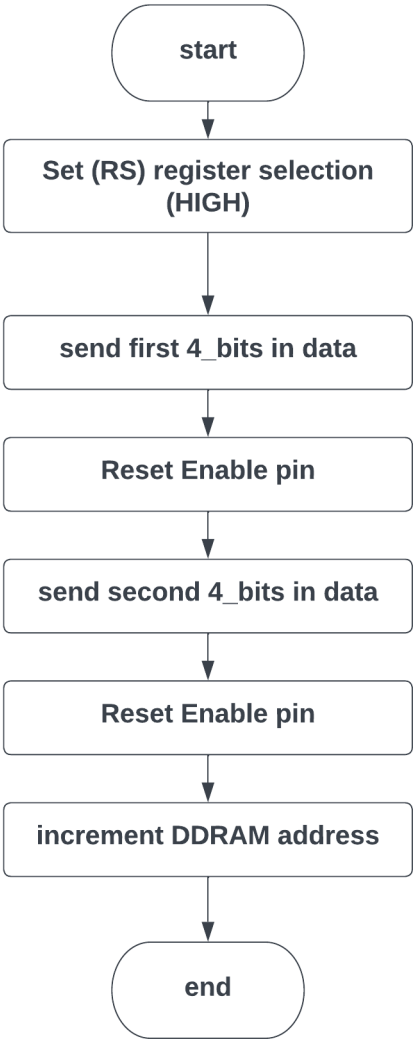
LCD_WriteChar(u8 ch)



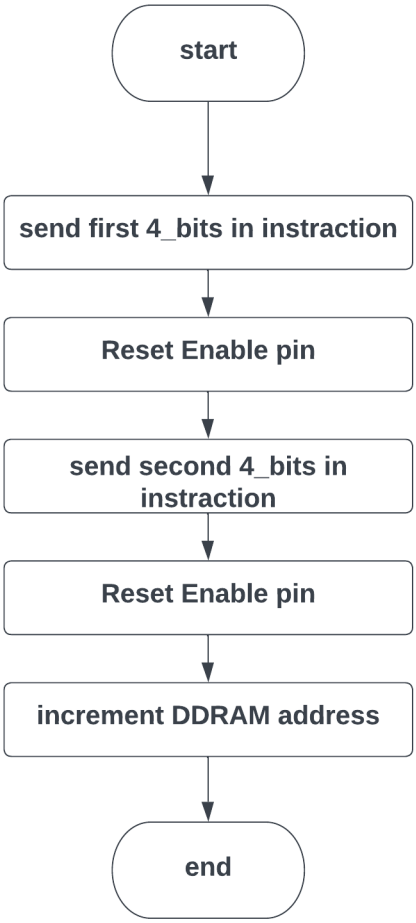
LCD_WriteString(u8*str)



WriteData(u8 data)

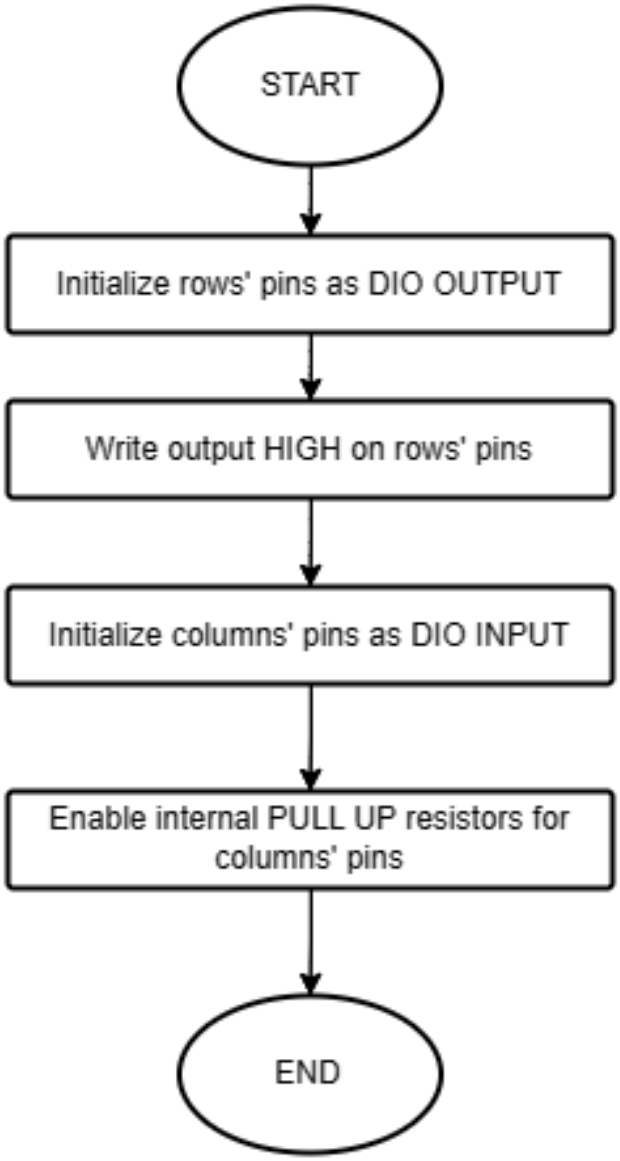


WriteIns(u8 ins)

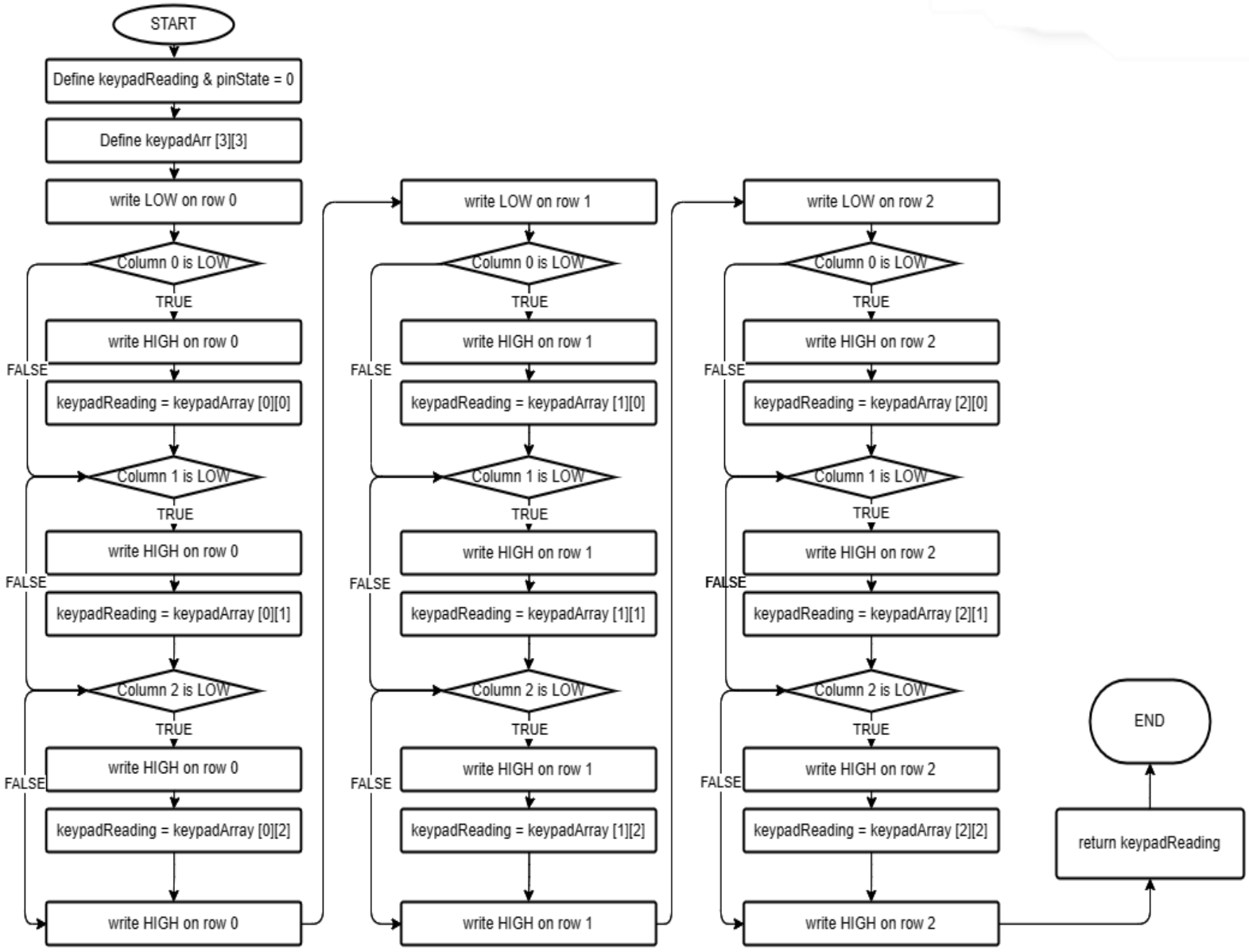


4. KEYPAD

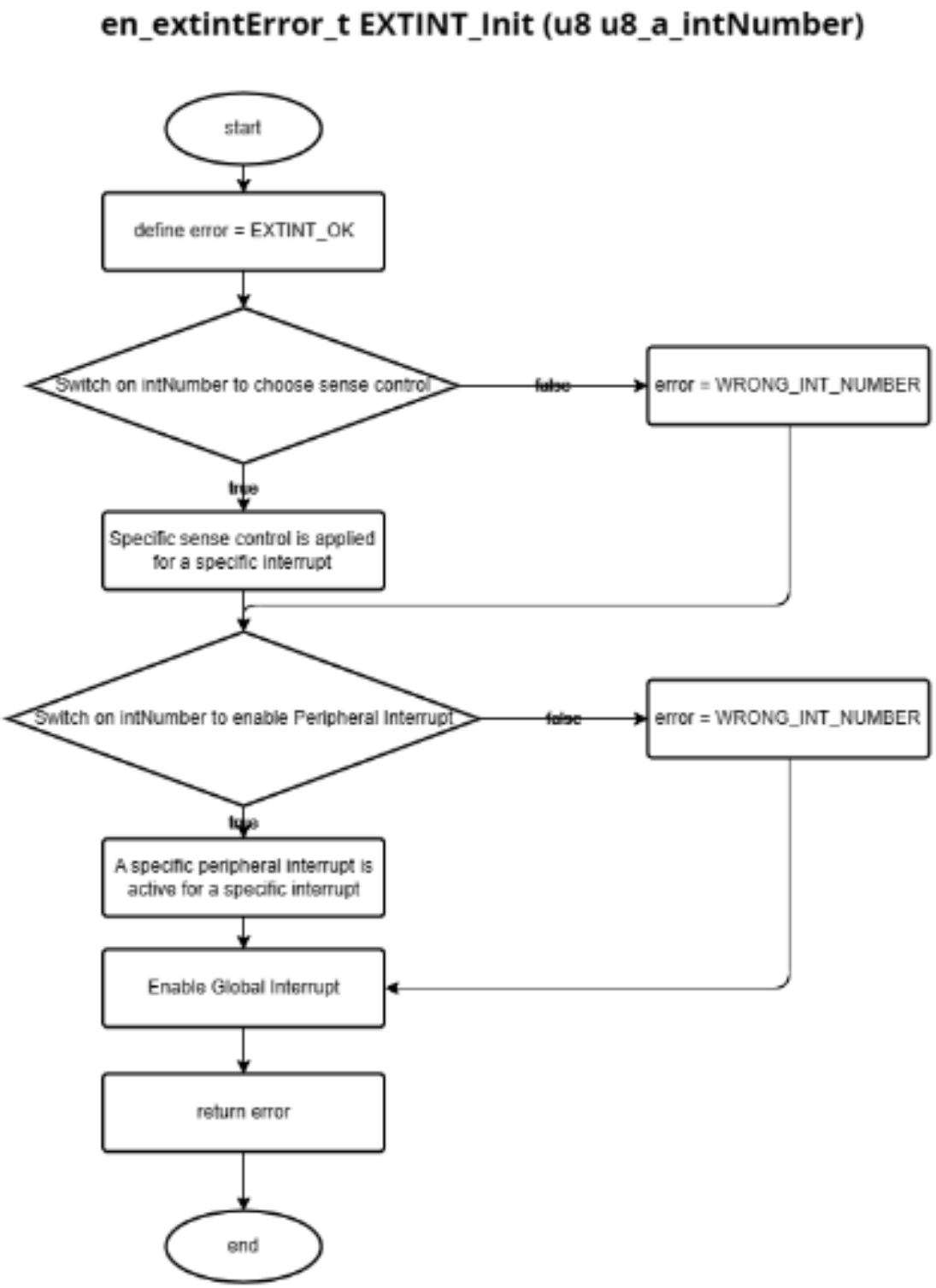
void KEYPAD_init(void);



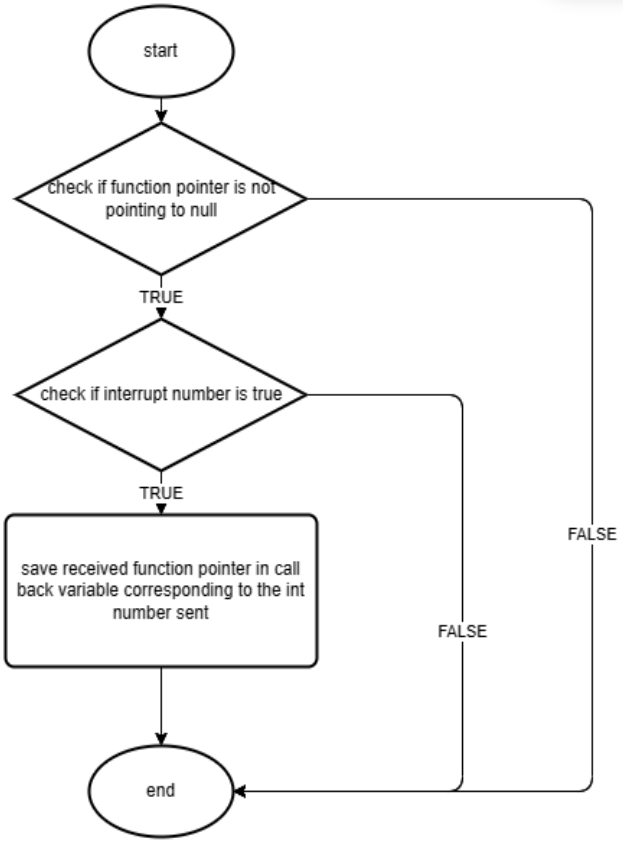
u8 KEYPAD_read(void)



5. EXTERNAL INTERRUPTS



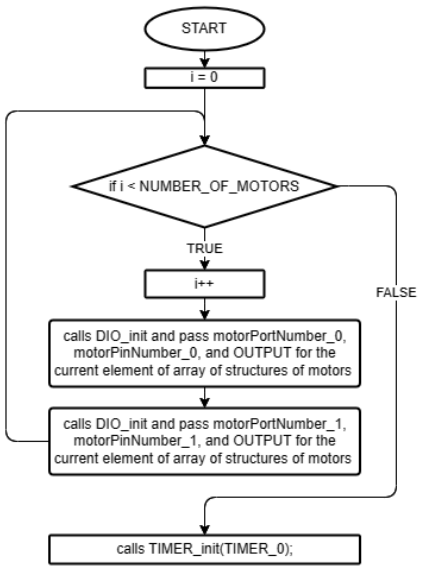
void EXTINT_setCallBackInt (u8 u8_a_intNumber, void (*funP



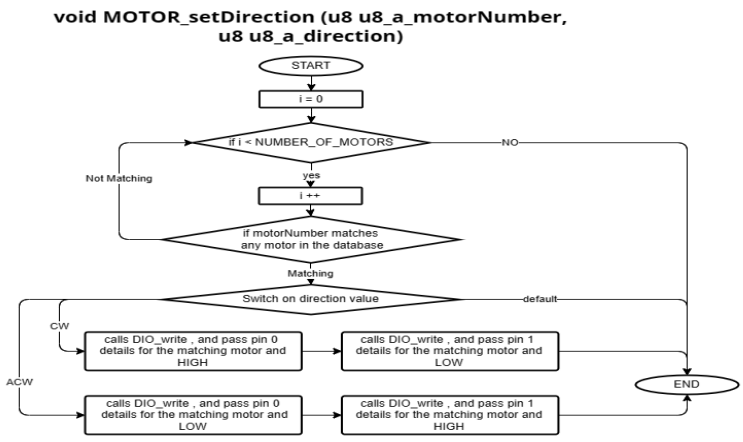
6. Motor

void motor_init(void);

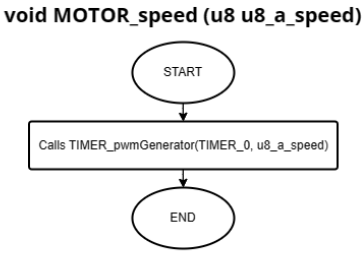
void MOTOR_init (void)



void motor_setdirection (uint8_t direction)

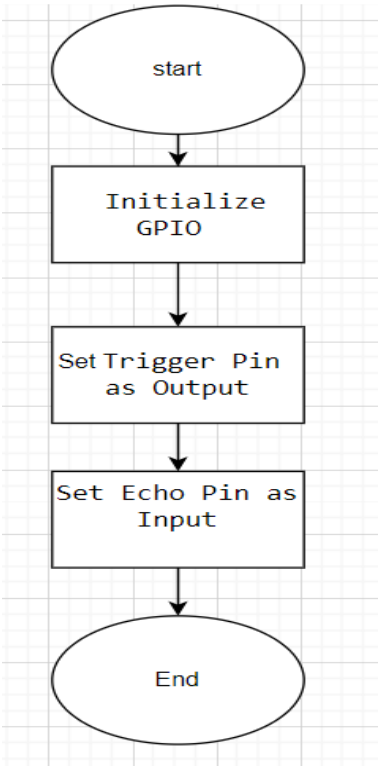


void motor_set_speed(uint8_t speed);

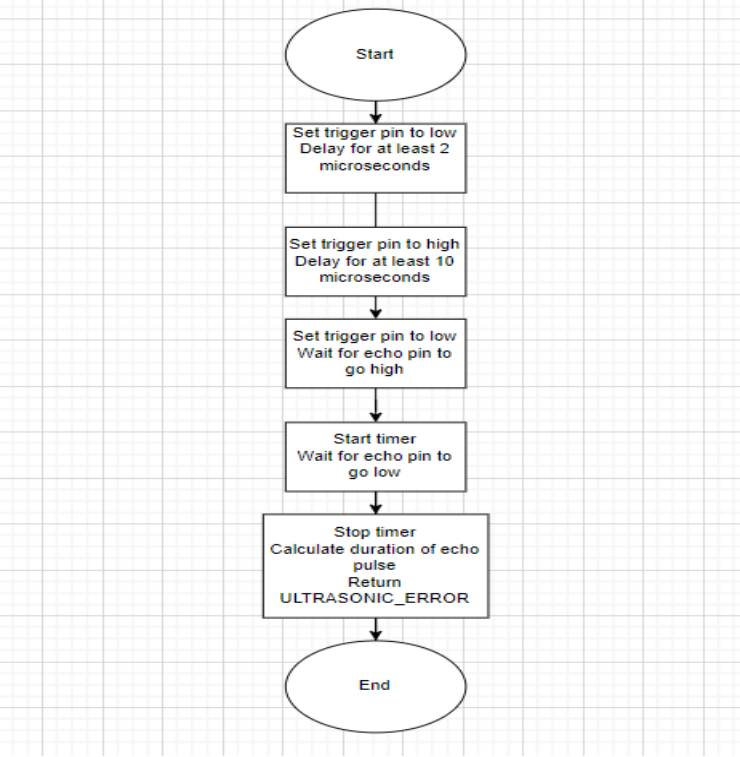


7. Ultrasonic

void ultrasonic_init(void)



void ultrasonic_trigger_measurement (void);



Ultrasonic_Status ultrasonic_get_distance (uint8_t distance);