Design Document

Obstacle Avoidance Robot

Khaled Mustafa Anwar

Sprints

Table of Contents

Introduction

Hardware Components

Hardware Components	Number of items required
ATMega32 micro-controller	1 Unit
DC Motor	4 Units
Push Button	1 Unit
Ultrasonic Sensor	1 Unit
LCD	1 Unit
Keypad	1 Unit

System requirements

1. The car starts initially from 0 speed.
2. The default rotation direction is to the right.
3. Press (Keypad button 1), (Keypad button 2) to start or stop the robot respectively.
4. After Pressing Start:
    1. The LCD will display a centered message in line 1 “Set Def. Rot.”
    2. The LCD will display the selected option in line 2 “Right”
    3. The robot will wait for 5 seconds to choose between Right and Left:
    4. When PBUTTON0 is pressed once, the default rotation will be Left and the LCD line 2 will be updated.
5. When PBUTTON0 is pressed again, the default rotation will be Right and the LCD line 2 will be updated.
    1. For each press the default rotation will changed and the LCD line 2 is updated.
    2. After the 5 seconds the default value of rotation is set.
    3. The robot will move after 2 seconds from setting the default direction of rotation.
    4. For No obstacles or object is far than 70 cm:
6. The robot will move forward with 30% speed for 5 seconds
7. After 5 seconds it will move with 50% speed as long as there was no object or objects are located at more than 70 centimeters distance

The LCD will display the speed and moving direction in line 1: “Speed:00% Dir: F/B/R/S” where F: forward, B: Backwards, R: Rotating, and S: Stopped

8. The LCD will display Object distance in line 2 “Dist.: 000 Cm”

High Level Design

Layered Architecture

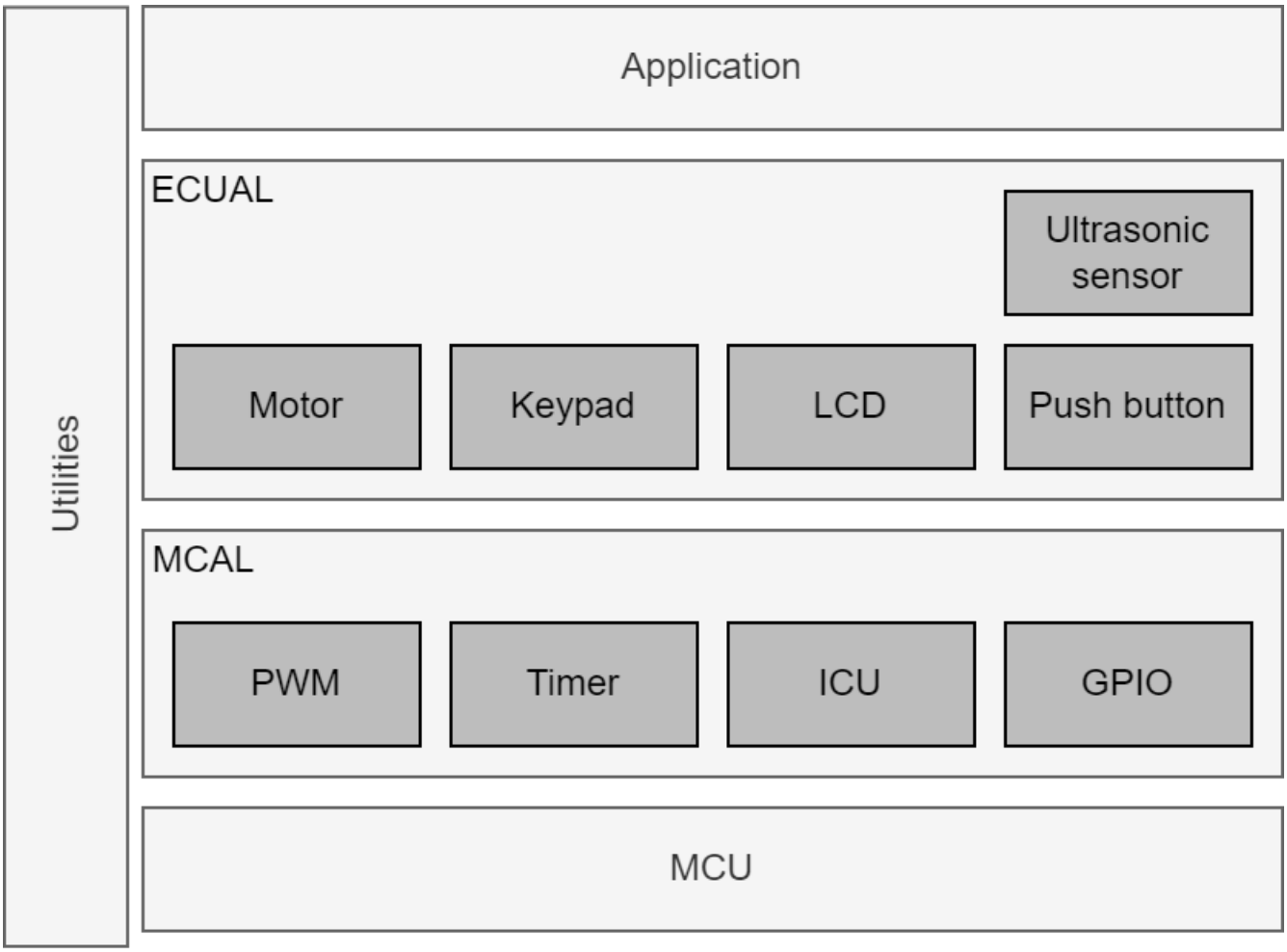


Figure 1: Moving robot layered architecture

System’s Logic Flowchart

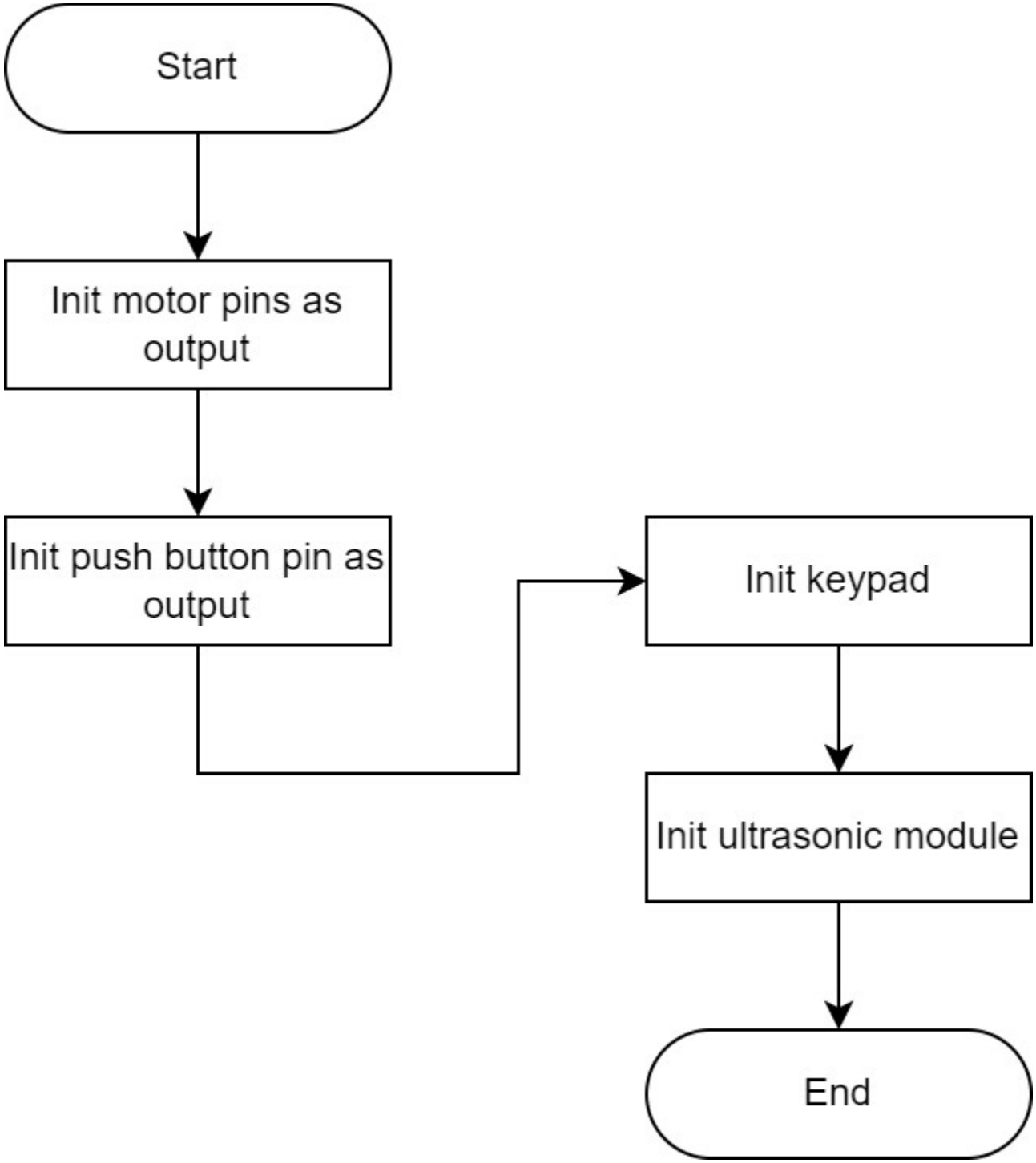


Figure 2: Moving Robot Initialization State

Warm-up Sage

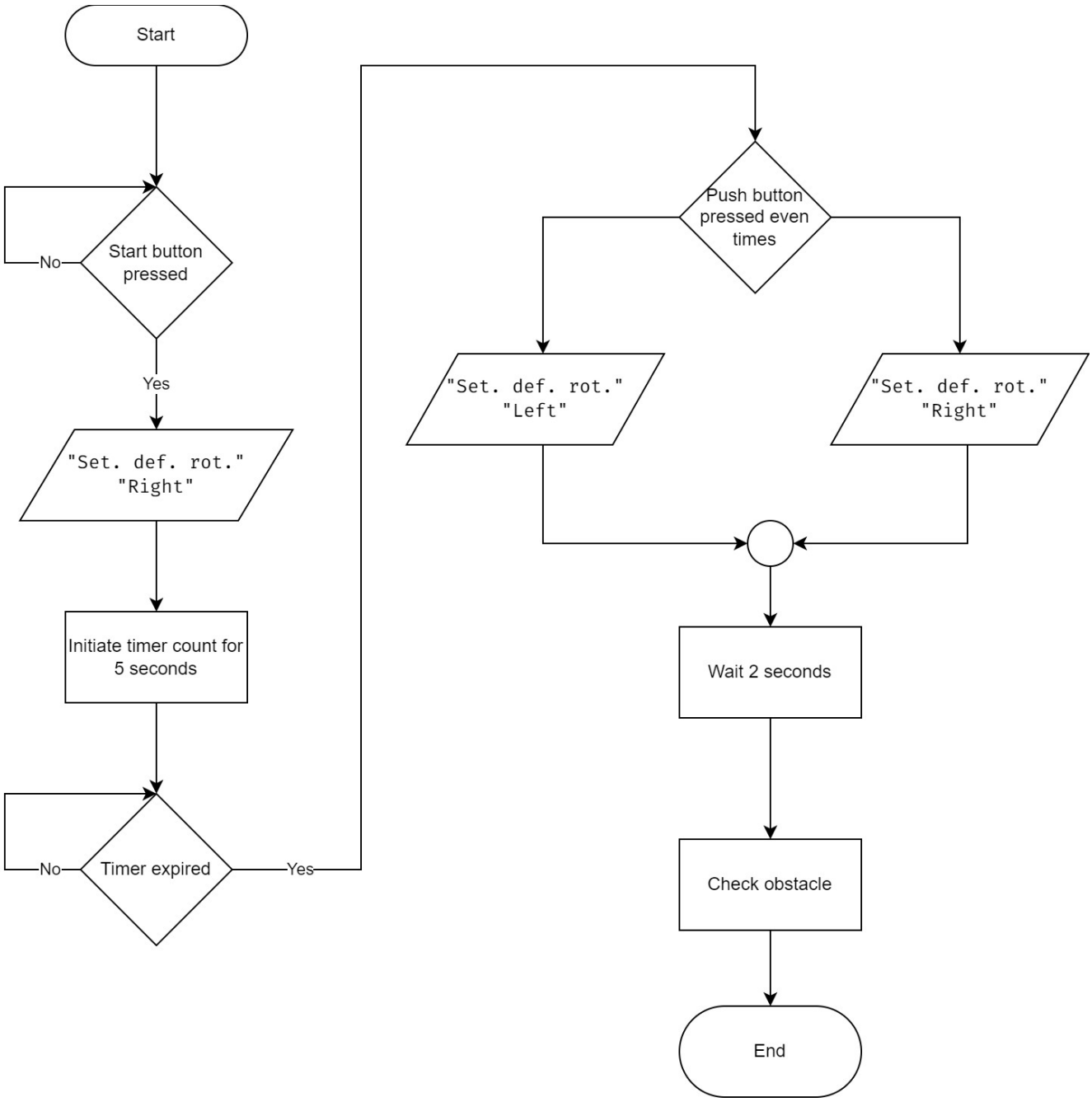
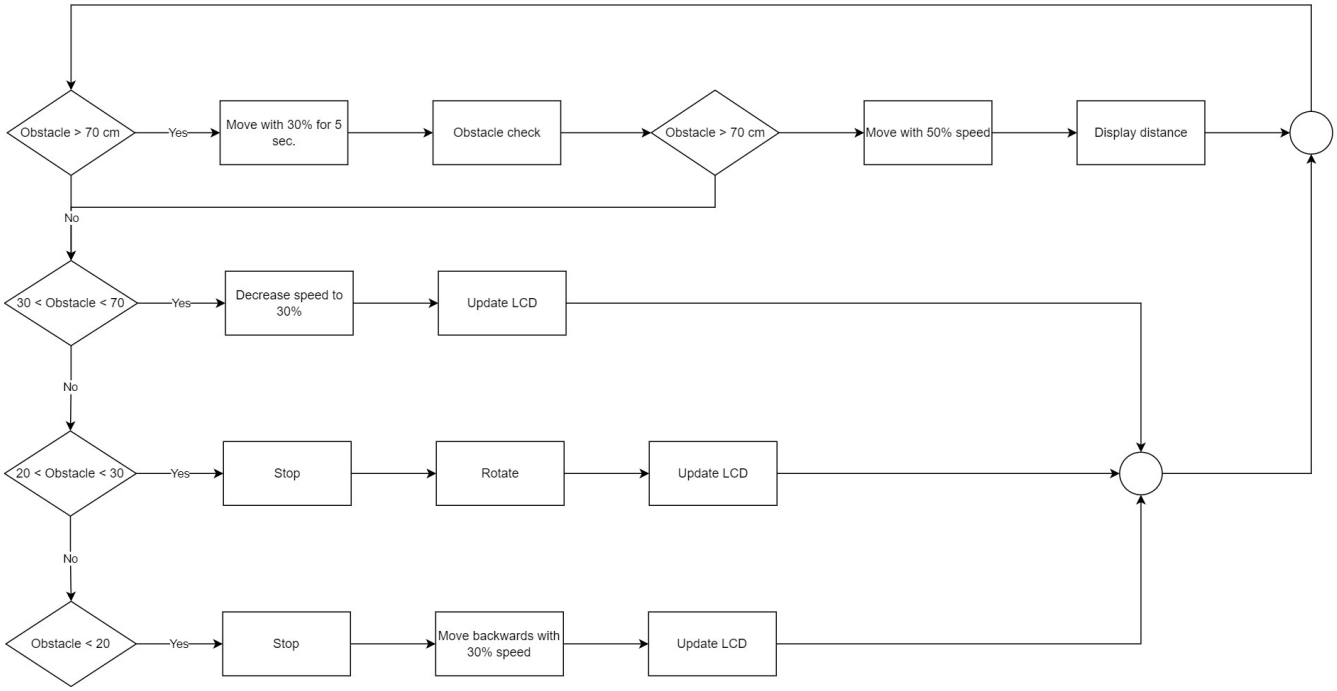


Figure 3: Moving Robot Warm-up Stage

Obstacle detection logic



Modules’ Description

Application Layer

It defines the functionality and behavior of the system. It is the highest layer in the software stack and encompasses the applications and software components that directly interact with the end-users or external systems.

The primary objective of the application layer is to implement the specific features and tasks that fulfill the intended purpose of the embedded system.

Electronic Components Unit Abstraction Layer (ECUAL)

The Electronic Components Unit Abstraction Layer in an embedded systems architecture serves as a bridge between the hardware components and the higher-level software layers. It provides a standardized interface and a set of functions that abstract of individual electronic components, such as sensors, actuators, and peripheral devices.

Microcontroller Abstraction Layer (MCAL)

The Microcontroller Abstraction Layer (MCAL) serves as a bridge between the hardware-specific features of a microcontroller and the higher-level software layers. It provides a standardized set of functions and interfaces that abstract the low-level details of the microcontroller, including its peripherals, timers, interrupts, and input/output (I/O) operations.

Drivers' Documentation

Application Layer

App_init

Function Name	App_init
Syntax	void App_init(void);
Parameters[in]	None
Parameters[out]	None
Parameters[in/out]	None
Return	None

App_start

Function Name	App_start
Syntax	void App_start(void);
Parameters[in]	None
Parameters[out]	None
Parameters[in/out]	None
Return	None

Keypad

KEYPAD_init

Function Name	KEYPAD_init
Syntax	void KEYPAD_init(void);
Parameters[in]	None
Parameters[out]	None
Parameters[in/out]	None
Return	None

KEYPAD_getKeyPressed

Function Name	KEYPAD_getKeyPressed
Syntax	uint8 KEYPAD_getKeyPressed(void);
Parameters[in]	None
Parameters[out]	None
Parameters[in/out]	None
Return	The number of keypad pressed.

LED

LED_init

Function Name	LED_init
Syntax	en_LED_State LED_init(st_LED_config_t *ptr_LED_config);
Parameters[in]	ptr_LED_config: Address to the LED configuration
Parameters[out]	None
Parameters[in/out]	None
Return	• LED_STATUS_SUCCESS • LED_STATUS_FAILED

LED_on

Function Name	LED_on
Syntax	en_LED_State LED_on(st_LED_config_t *ptr_LED_config);
Parameters[in]	ptr_LED_config: Address to the LED configuration
Parameters[out]	None
Parameters[in/out]	None
Return	• LED_STATUS_SUCCESS • LED_STATUS_FAILED

LED_off

Function Name	LED_off
Syntax	en_LED_State LED_off(st_LED_config_t *ptr_LED_config);

Parameters[in]	ptr_LED_config: Address to the LED configuration
Parameters[out]	None
Parameters[in/out]	None
Return	• LED_STATUS_SUCCESS • LED_STATUS_FAILED

LED_toggle

Function Name	LED_init
Syntax	en_LED_State LED_toggle(st_LED_config_t *ptr_LED_config);
Parameters[in]	ptr_LED_config: Address to the LED configuration
Parameters[out]	None
Parameters[in/out]	• LED_STATUS_SUCCESS • LED_STATUS_FAILED
Return	None

Push Button

PB_init

Function Name	PB_init
Syntax	en_PB_State PB_init(st_PB_config_t *ptr_st_PB_config);
Parameters[in]	ptr_st_PB_config: Address of the push button configuration
Parameters[out]	None
Parameters[in/out]	None
Return	PB_STATUS_SUCCESS PB_STATUS_FAILED

PB_status

Function Name	SOS_disable
Syntax	void SOS_disable ptr_sos_db_t *ptr_sos_db);
Parameters[in]	ptr_sos_db: Address of the SOS database
Parameters[out]	None
Parameters[in/out]	None
Return	None

LCD

LCD_init

Function Name	LCD_init
Syntax	void LCD_init(void);
Parameters[in]	None
Parameters[out]	None
Parameters[in/out]	None
Return	None

LCD_sendCommand

Function Name	LCD_sendCommand
Syntax	void LCD_sendCommand(uint8 lcd_command);
Parameters[in]	lcd_command: The command to be sent to the LCD.
Parameters[out]	None
Parameters[in/out]	None
Return	None

LCD_displayString

Function Name	LCD_displayString
Syntax	void LCD_displayString(uint8 *p_string);
Parameters[in]	Character: The character to be displayed.
Parameters[out]	None
Parameters[in/out]	None
Return	None

LCD_moveCursor

Function Name	LCD_moveCursor
Syntax	void LCD_moveCursor(uint8 row, uint8 col);
Parameters[in]	Row: The number of row the cursor will move to. Col: The number of cols the cursor will move to.
Parameters[out]	None
Parameters[in/out]	None
Return	None

Ultrasonic module

ULTRASONIC_init

Function Name	ULTRASONIC_init
Syntax	void ULTRASONIC_init(st_ULTRASONIC_CONFIG *ptr_st_ultrasonic_config);
Parameters[in]	ptr_st_ultrasonic_config: Address of the ultrasonic configuration
Parameters[out]	None
Parameters[in/out]	None
Return	None

ULTRASONIC_triggerFire

Function Name	ULTRASONIC_triggerFire
Syntax	void ULTRASONIC_triggerFire(st_ULTRASONIC_CONFIG *ptr_st_ultrasonic_config);
Parameters[in]	ptr_st_ultrasonic_config: Address of the ultrasonic configuration
Parameters[out]	None
Parameters[in/out]	None
Return	None

ULTRASONIC_distCalc

Function Name	ULTRASONIC_distCalc
Syntax	u32 ULTRASONIC_distCalc(st_ULTRASONIC_CONFIG *st_ultrasonic_config);
Parameters[in]	ptr_st_ultrasonic_config: Address of the ultrasonic configuration
Parameters[out]	None
Parameters[in/out]	None
Return	The distance between the robot and the obstacle.

PWM Module

PWM_init

Function Name	PWM_init
Syntax	void PWM_init();

Parameters[in]	None
Parameters[out]	None
Parameters[in/out]	None
Return	None

PWM_startSignal

Function Name	PWM_startSignal
Syntax	void PWM_startSignal(u16 freq, u8 duty_cycle);
Parameters[in]	Freq duty_cycle
Parameters[out]	None
Parameters[in/out]	None
Return	None

PWM_stopSignal

Function Name	PWM_stopSignal
Syntax	void PWM_stopSignal();
Parameters[in]	None
Parameters[out]	None
Parameters[in/out]	None
Return	None

GPIO

GPIO_setPinDirection

Function Name	GPIO_setPinDirection
Syntax	en_GPIO_State GPIO_setPinDirection(st_GPIO_CONFIG_t *ptr_st_GPIO_CONFIG);
Parameters[in]	ptr_st_GPIO_CONFIG: Address of the GPIO pin configuration.
Parameters[out]	None
Parameters[in/out]	None
Return	• GPIO_STATUS_SUCCESS • GPIO_STATUS_FAILED

GPIO_writePin

Function Name	`GPIO_writePin`
Syntax	`en_GPIO_State GPIO_writePin(st_GPIO_CONFIG_t *ptr_st_GPIO_CONFIG);`
Parameters[in]	`ptr_st_GPIO_CONFIG: Address of the GPIO pin configuration.`
Parameters[out]	`None`
Parameters[in/out]	`None`
Return	• GPIO_STATUS_SUCCESS • `GPIO_STATUS_FAILED`

GPIO_readPin

Function Name	`GPIO_readPin`
Syntax	`en_GPIO_State GPIO_readPin(st_GPIO_CONFIG_t *ptr_st_GPIO_CONFIG);`
Parameters[in]	`ptr_st_GPIO_CONFIG: Address of the GPIO pin configuration.`
Parameters[out]	`None`
Parameters[in/out]	`None`
Return	• GPIO_STATUS_SUCCESS • `GPIO_STATUS_FAILED`

GPIO_togglePin

Function Name	`GPIO_togglePin`
Syntax	`en_GPIO_State GPIO_togglePin(st_GPIO_CONFIG_t *ptr_st_GPIO_CONFIG);`
Parameters[in]	`ptr_st_GPIO_CONFIG: Address of the GPIO pin configuration.`
Parameters[out]	`None`
Parameters[in/out]	`None`
Return	• GPIO_STATUS_SUCCESS • `GPIO_STATUS_FAILED`

Timer

TIMER_init

Function Name	TIMER_init
Syntax	void TIMER_init(ptr_st_TIMER_CONFIG_t *ptr_st_timer_config)
Parameters[in]	ptr_st_timer_config: Address of the configuration structure of the timer module.
Parameters[out]	None
Parameters[in/out]	None
Return	None

TIMER_start

Function Name	TIMER_start
Syntax	void TIMER_start(en_TIMER_ID_t timer_id)
Parameters[in]	timer_id: Timer ID
Parameters[out]	None
Parameters[in/out]	None
Return	None

TIMER_stop

Function Name	TIMER_stop
Syntax	void TIMER_stop(en_TIMER_ID_t timer_id)
Parameters[in]	timer_id: Timer ID
Parameters[out]	None
Parameters[in/out]	None
Return	None

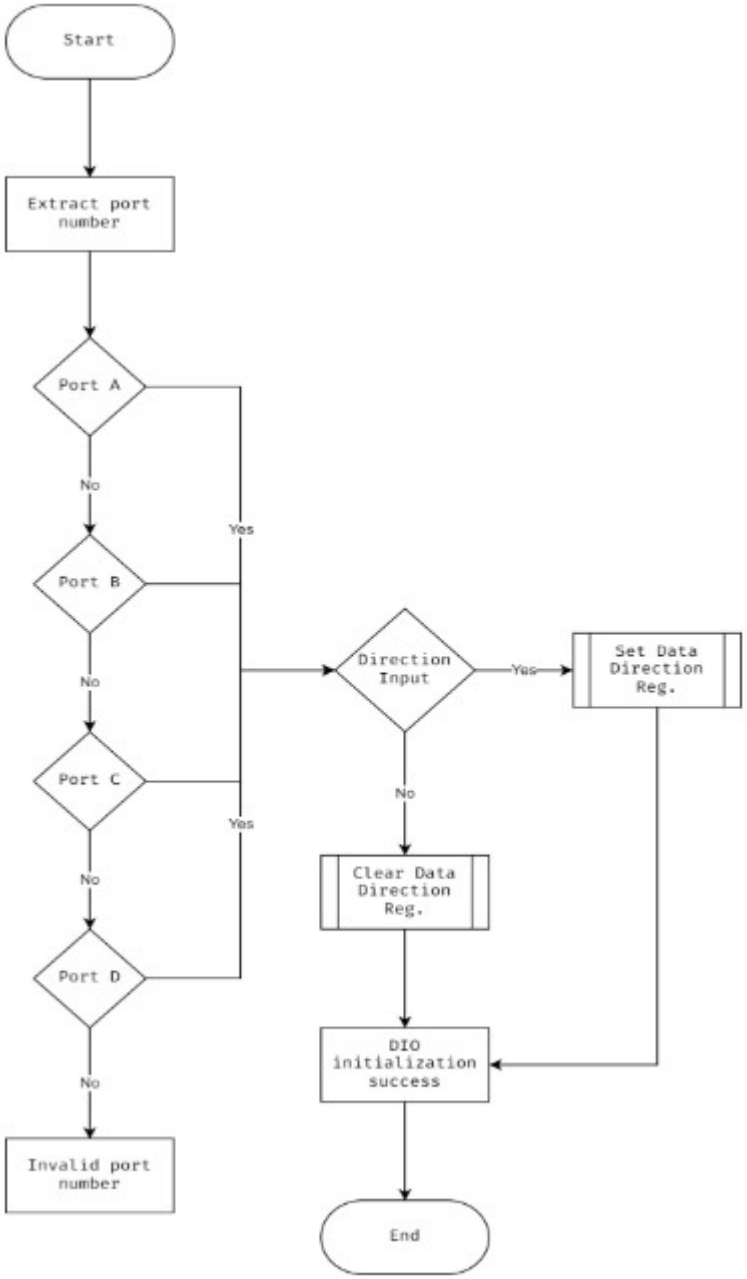
TIMER_delay

Function Name	TIMER_delay
Syntax	void TIMER_delay(uint32 delay_ms)
Parameters[in]	delay_ms: Specified time for delay in milli-seconds
Parameters[out]	None
Parameters[in/out]	None
Return	None

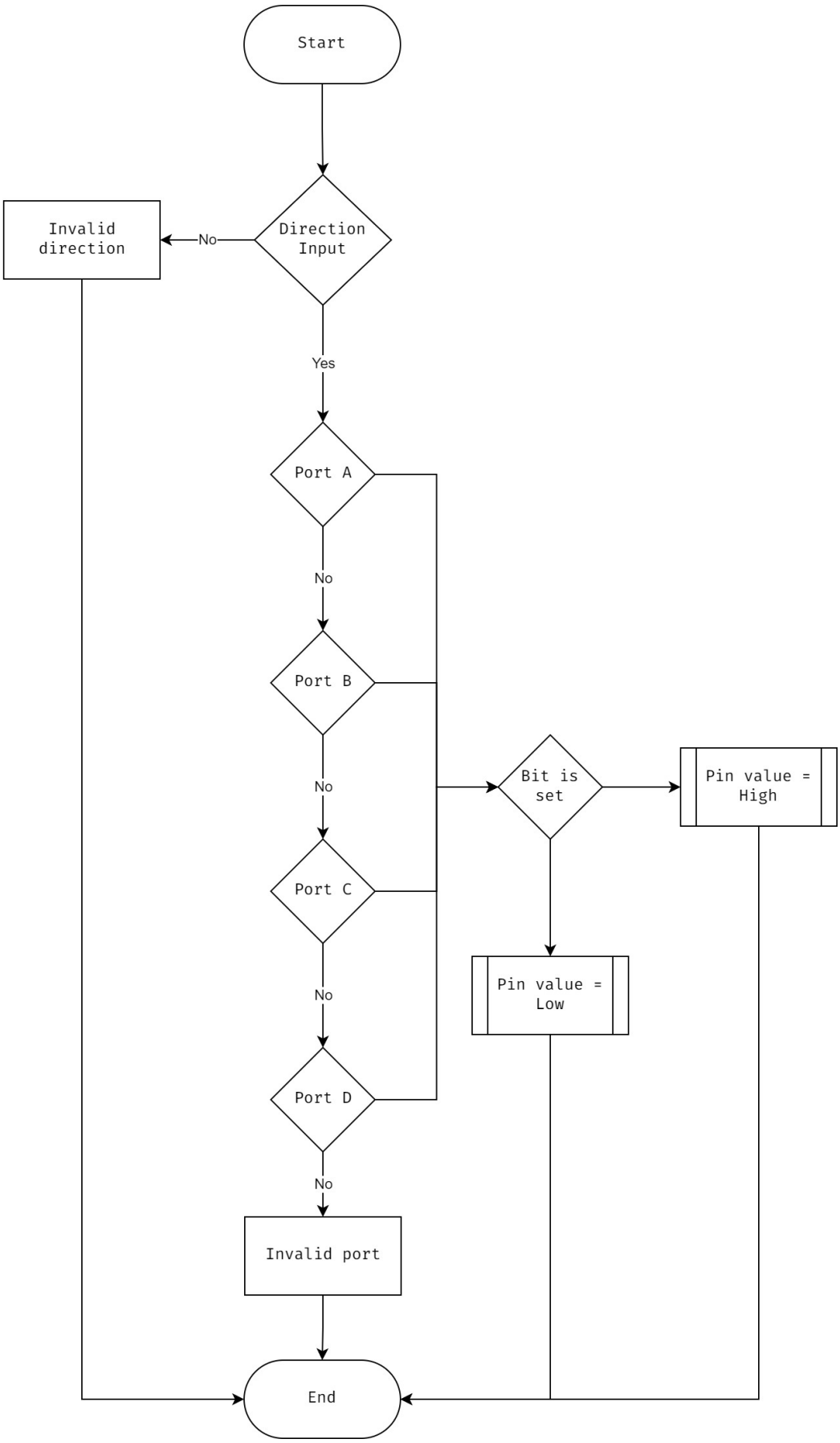
Low-level Design

GPIO Module

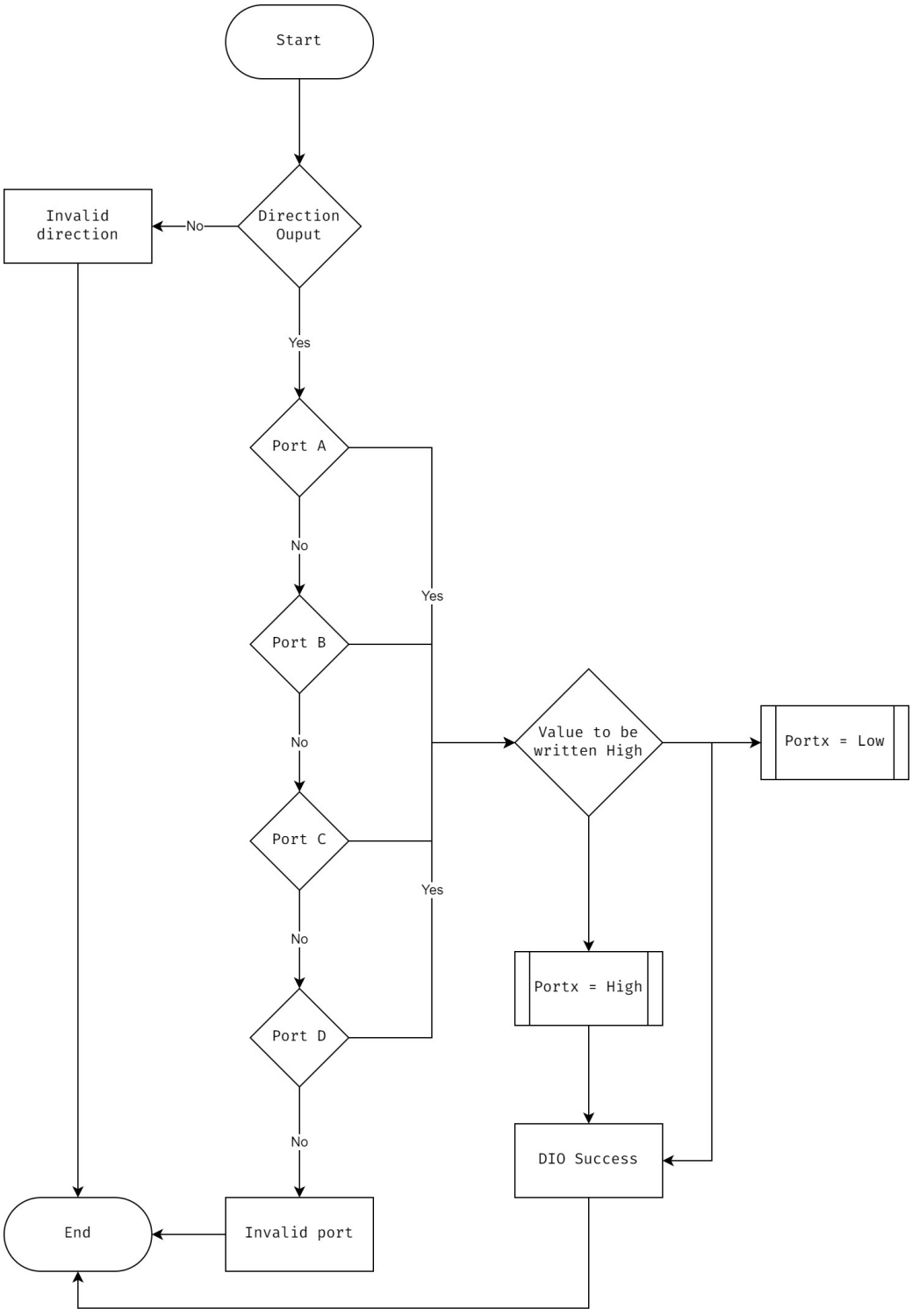
GPIO Initialization



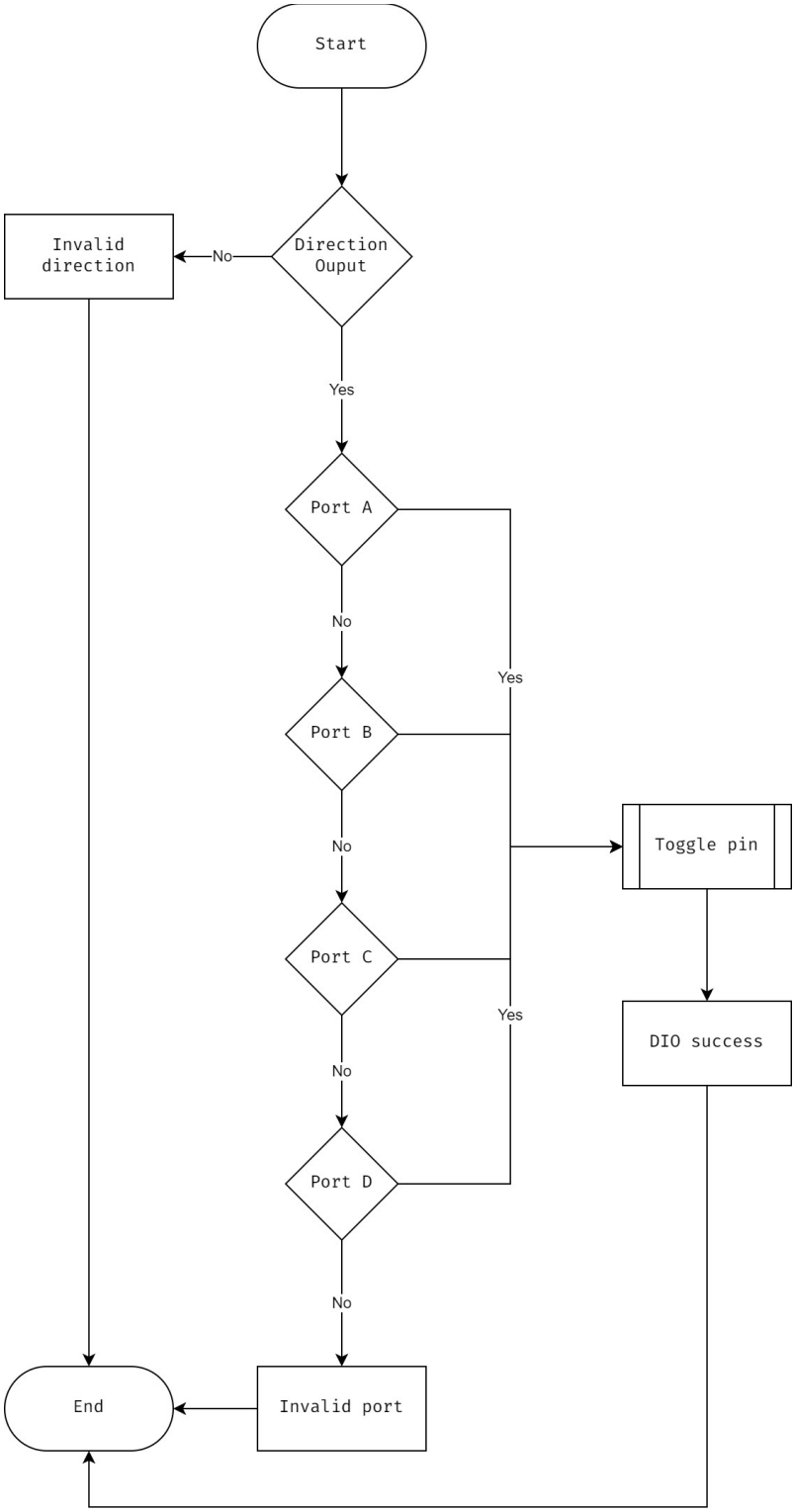
GPIO Read Pin



GPIO Write Pin

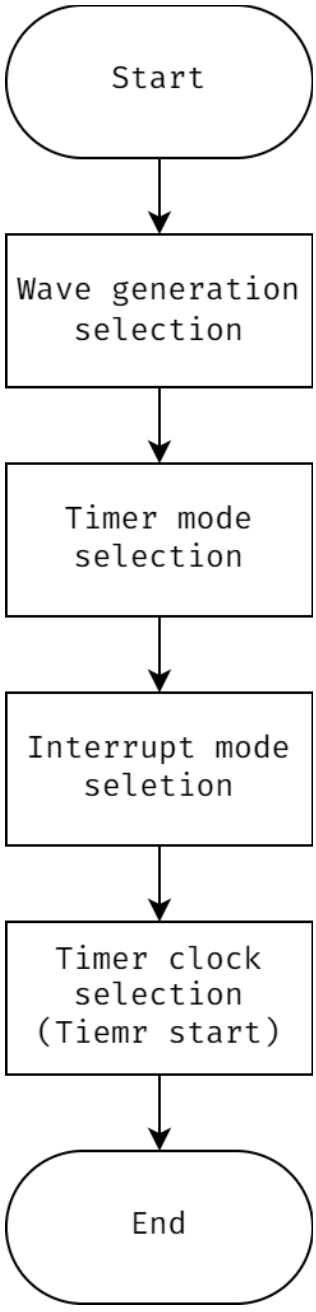


GPIO Toggle Pin



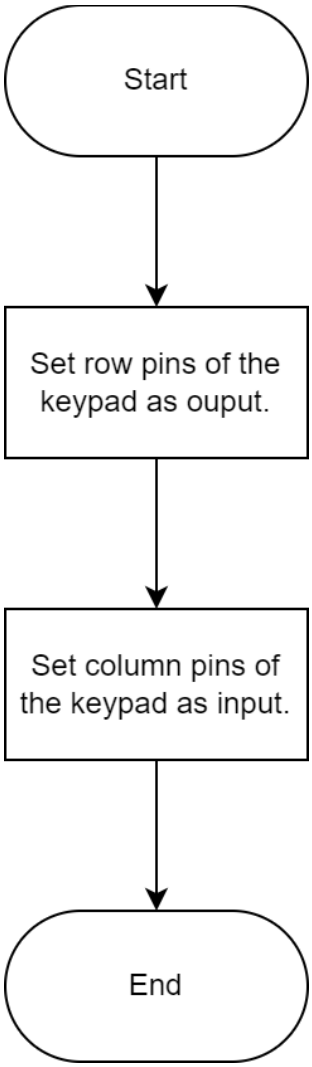
Timer

Timer Initialization

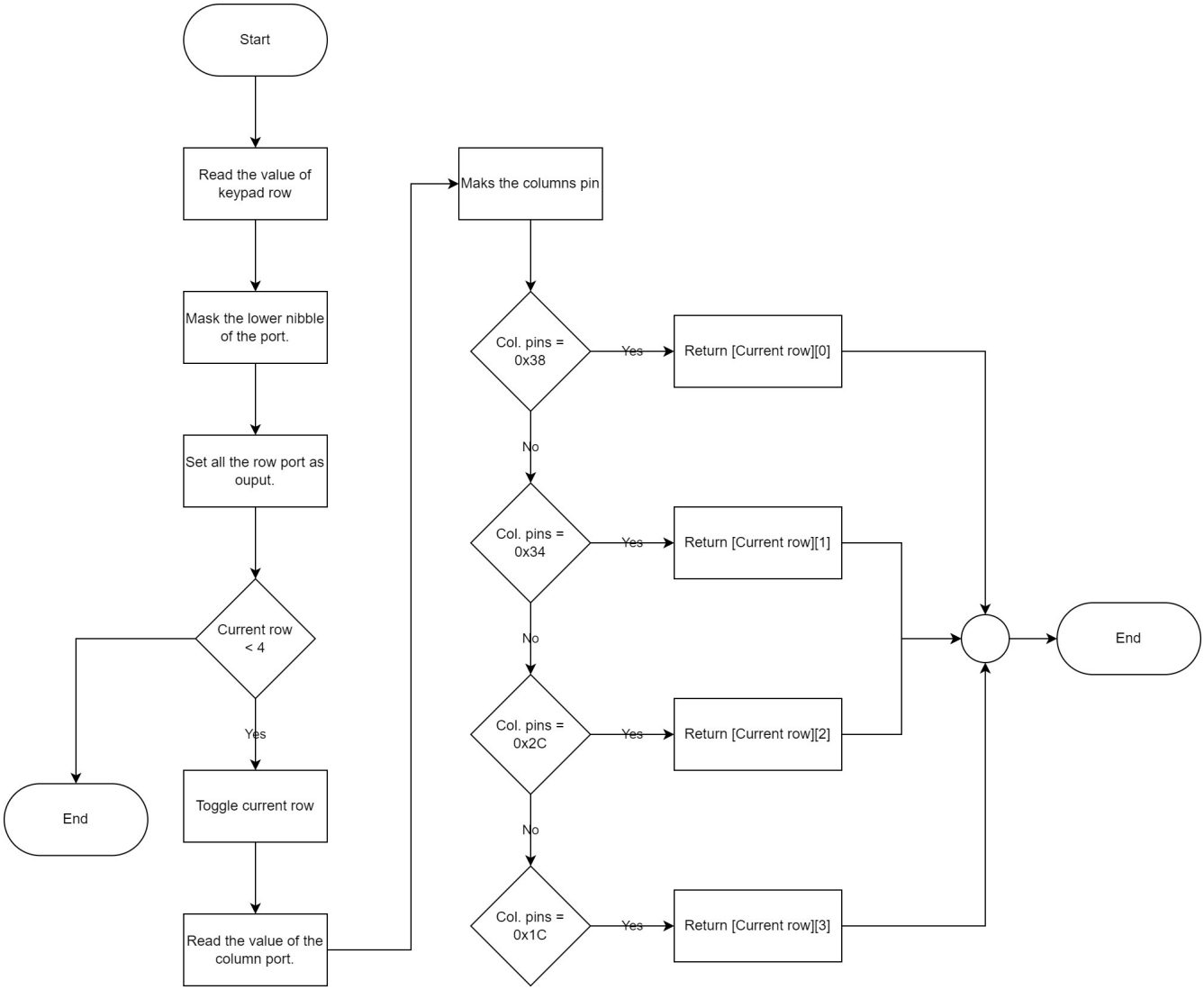


Keypad Driver

Keypad Initialization

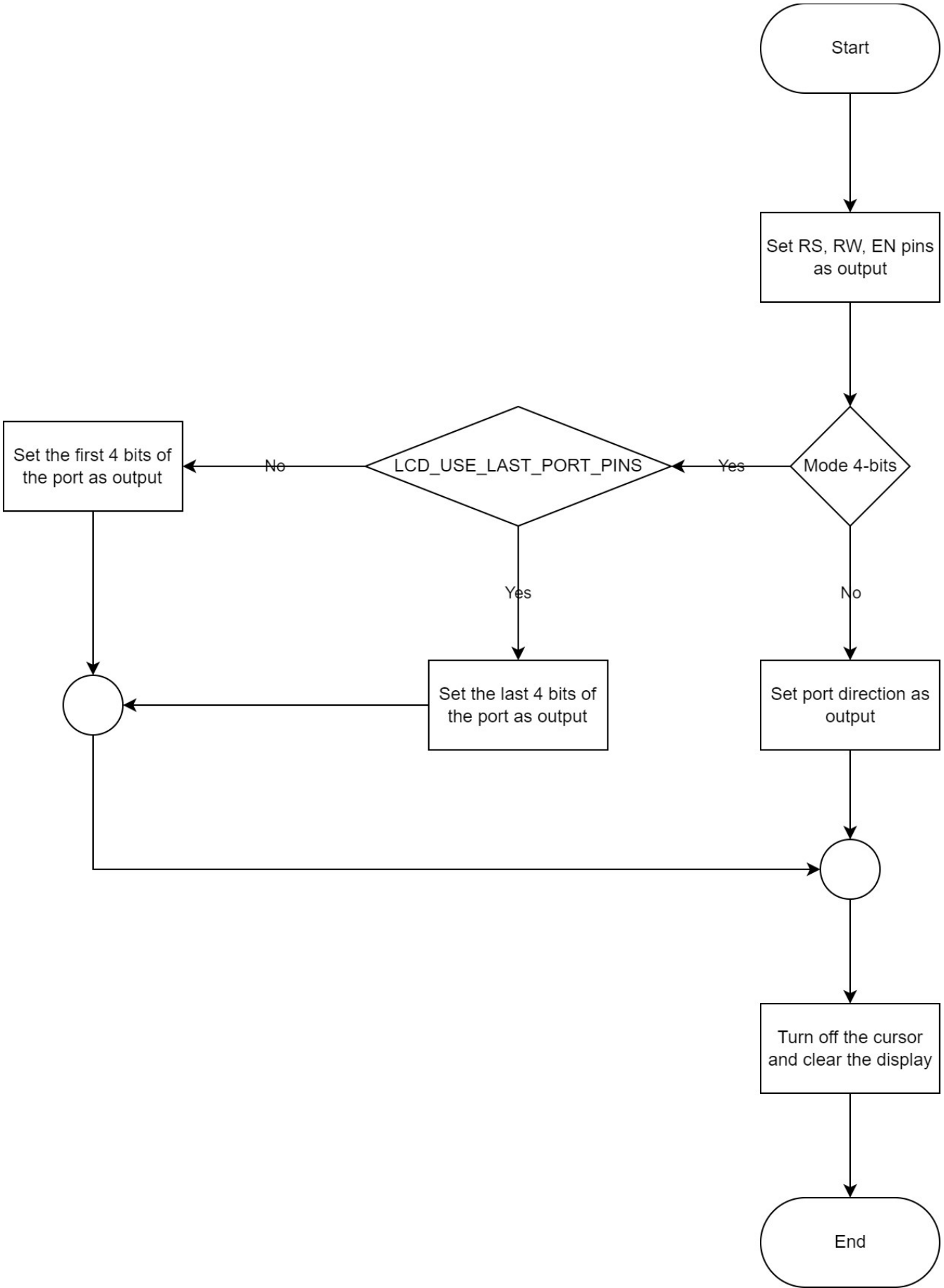


Keypad get Key pressed

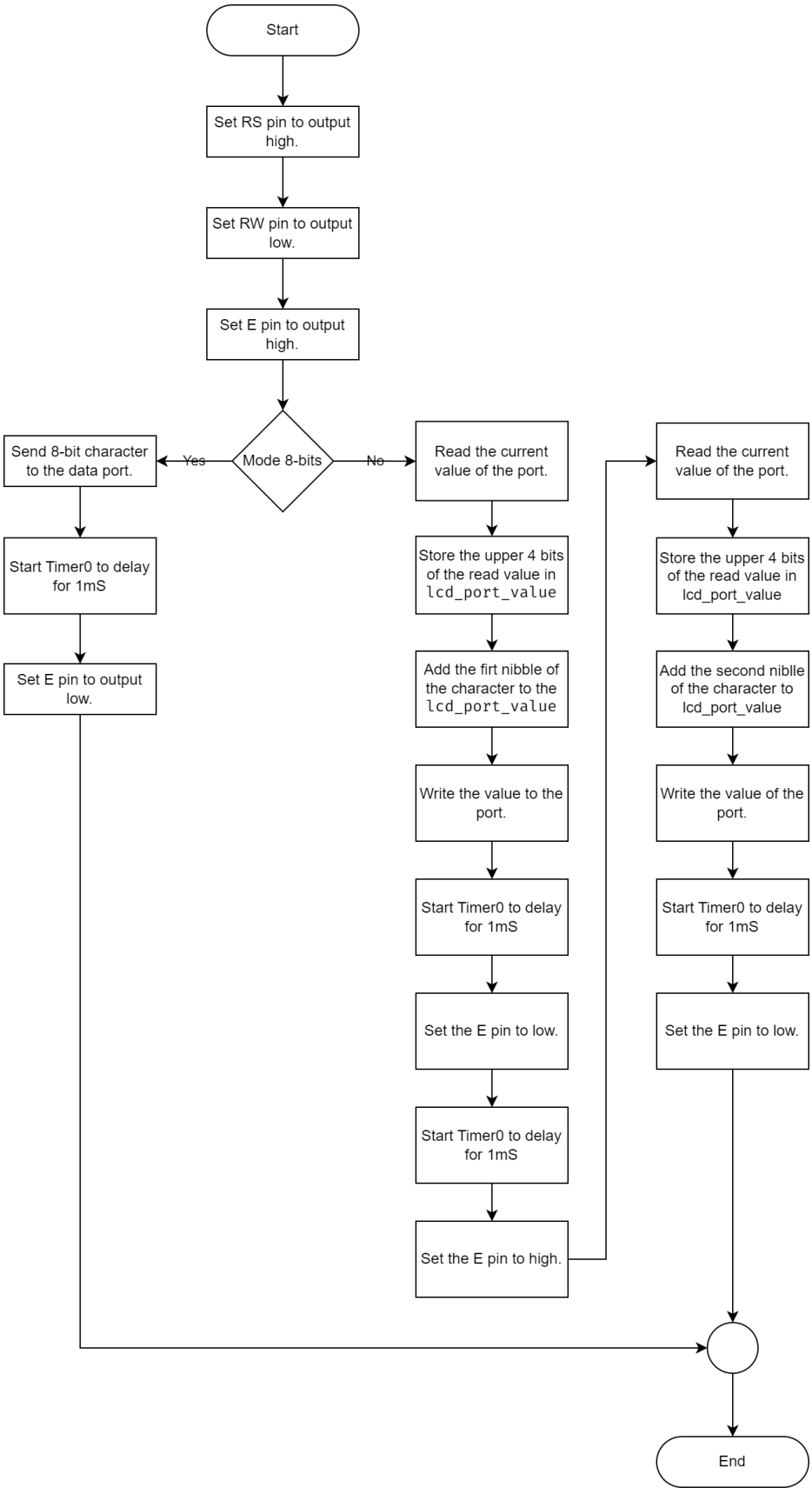


LCD Driver

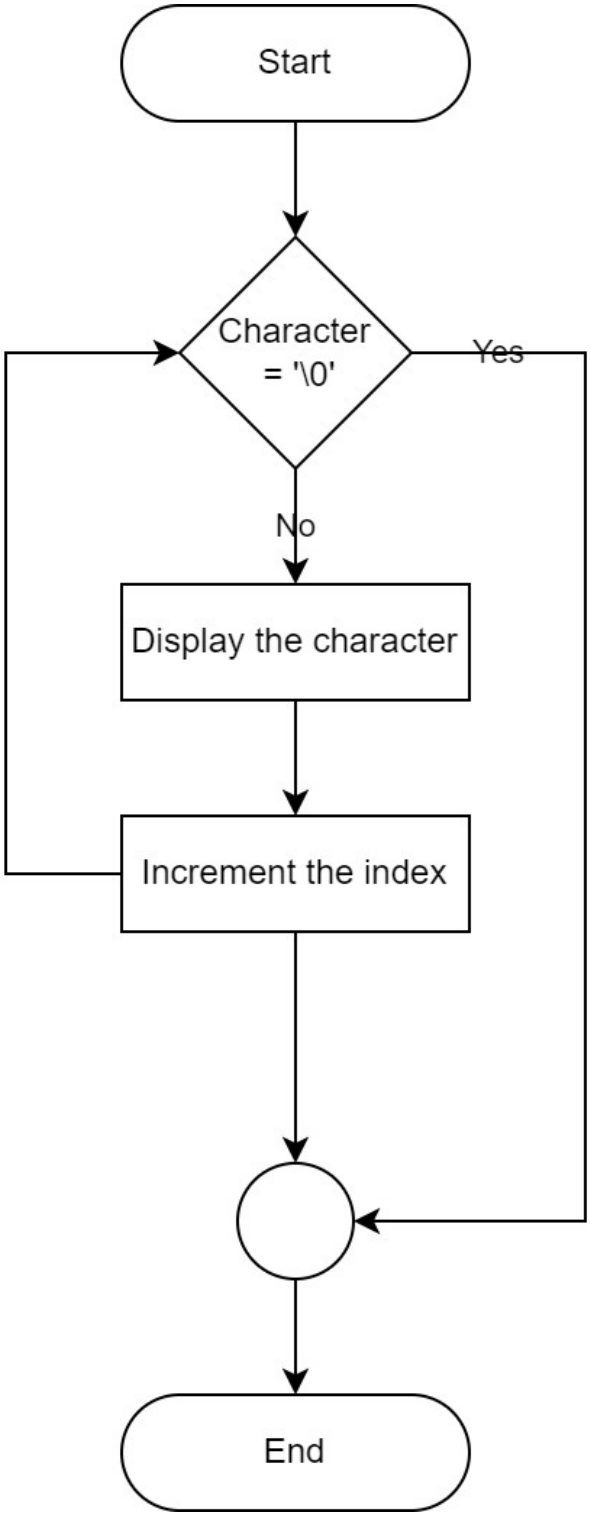
LCD Initialization



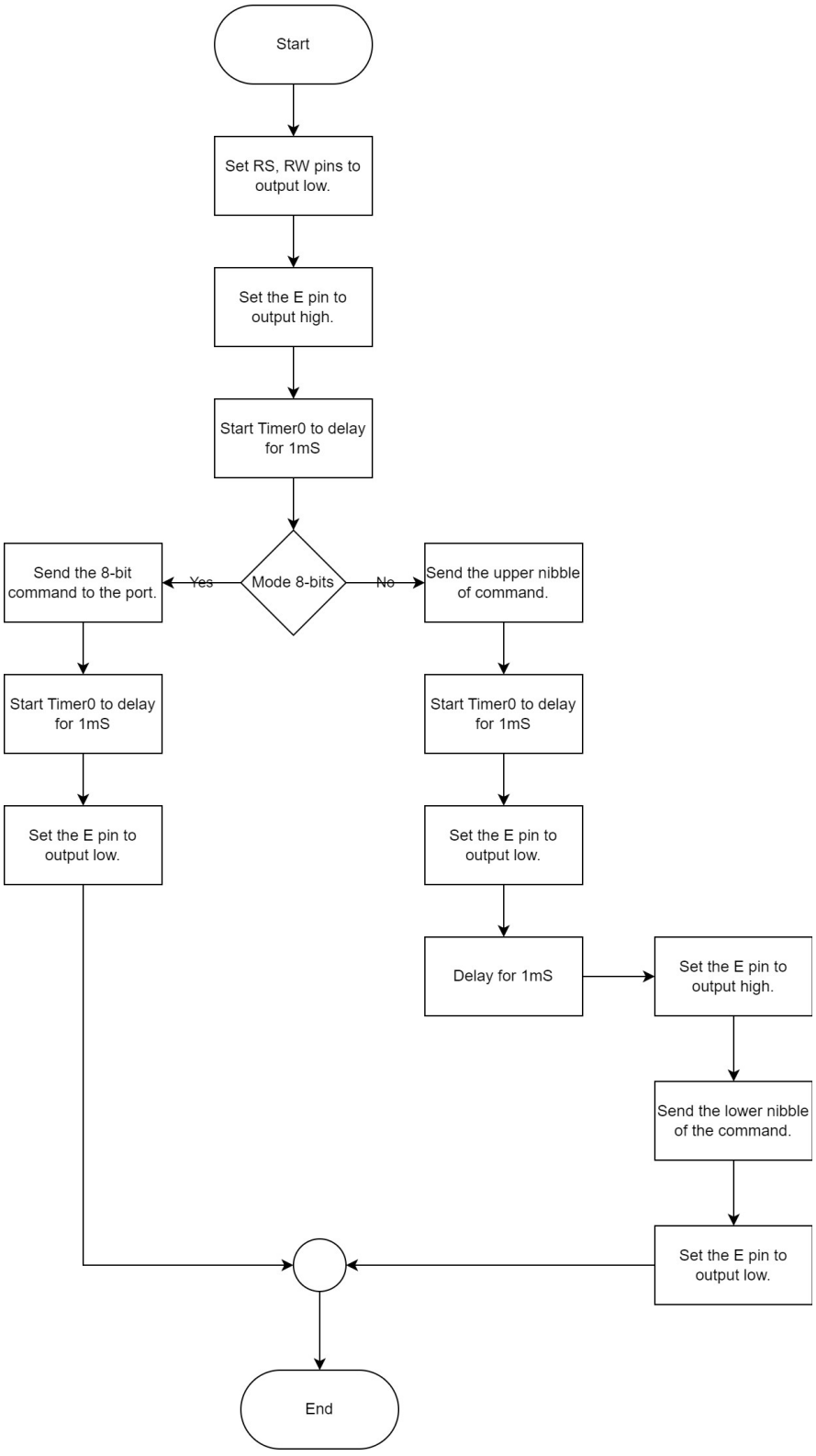
LCD Display Character



LCD Display String



LCD Send command



LCD Move Cursor