LCD Track Buddy (Timer And Stopwatch)

Shawheen Ghezavat

Spring, May 22, 2024

Behavior Description

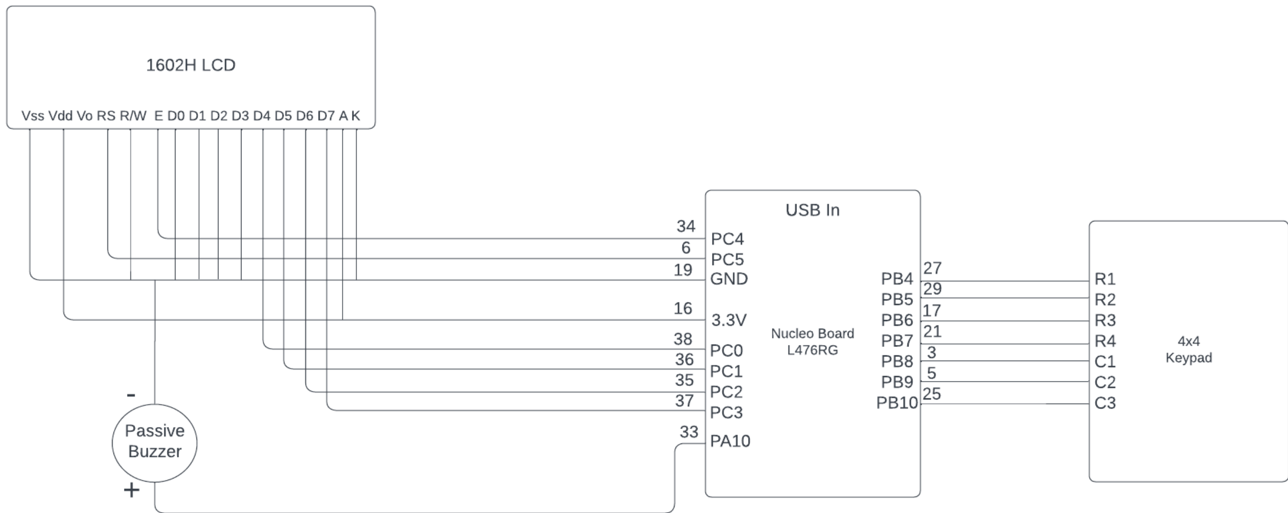
This device was inspired by my love for track & field and cooking. It behaves as an LCD timer and stopwatch using the STM32 NUCLEO-L476RG microcontroller, a keypad, an LCD screen, and a passive buzzer. It provides accurate timing for both modes. The stopwatch mode counts up and has a “lapping” feature, allowing users to store up to 3 different times if they’d like; they can view them with buttons on the keypad. The timer mode counts down from the desired time set by the user and sets off an external alarm once the time has finished. Users can easily start, stop, or reset the time in both modes.

System Specifications

Specification	Details
Power Supply Voltage	3.3V
Maximum Time	1 hour
Battery Life	N/A (powered via USB)
Display	LCD Screen (1602H) 16x2 characters
MCU Clock Speed	4 MHz
Overall Size	85.0 mm x 36.0 mm
Weight	Approx. 150 grams
Environmental Tolerance	Operating temperature 0°C to 70°C
Keypad Input	12-key keypad (9 keys used)
Buzzer	Activated at 1000 Hz for timer alarm
Modes	Stopwatch and Timer modes
Operating Conditions	Suitable for continuous operation with USB power
Response Time	Max Rise: 250 ms Max Fall: 300 ms Typical Rise: 150 ms Typical Fall: 200 ms
Display Size	66.0 mm x 16.2 mm
Resolution	5x8 dot matrix per character

Table 1

System Schematic



Software Architecture

Timer Calculations

- Timer Clock Frequency (F_{CLK}): 4 MHz
- Desired Time Interval (T_{OUT}): 1 second
- Prescaler (PSC): 0

The formula to calculate the ARR value for a given timer interval is:

$$T_{\text{OUT}} = \frac{(\text{ARR}+1)(\text{PSC}+1)}{F_{\text{CLK}}}$$

For PSC = 0:

$$T_{\text{OUT}} = \frac{(\text{ARR}+1)}{F_{\text{CLK}}}$$

Rearranging to solve for ARR:

$$\text{ARR}+1 = T_{\text{OUT}} \times F_{\text{CLK}}$$
$$\text{ARR} = (T_{\text{OUT}} \times F_{\text{CLK}}) - 1$$

Substituting the given values:

$$\text{ARR} = (1\,\text{s} \times 4,000,000\,\text{Hz}) - 1$$
$$\text{ARR} = 4,000,000 - 1$$
$$\text{ARR} = 3,999,999$$

PWM Calculations

- Timer Clock Frequency: 4 MHz

The formula to calculate the ARR value for a given PWM frequency is:

$$\text{ARR} = \left(\frac{\text{Timer Clock Frequency}}{\text{PWM Frequency}}\right) - 1$$

The CCR value for a 50% duty cycle is:

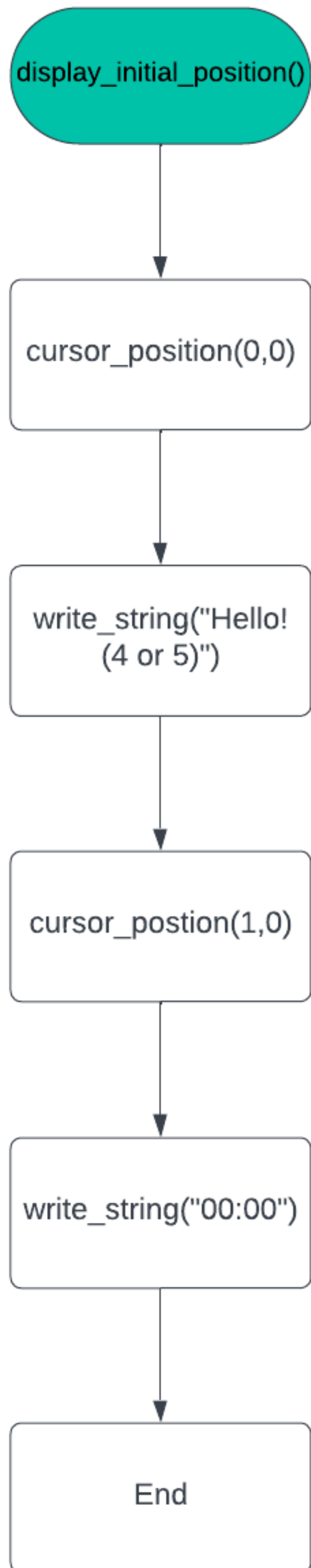
$$\text{CCR} = \left(\frac{\text{ARR}+1}{2}\right)$$

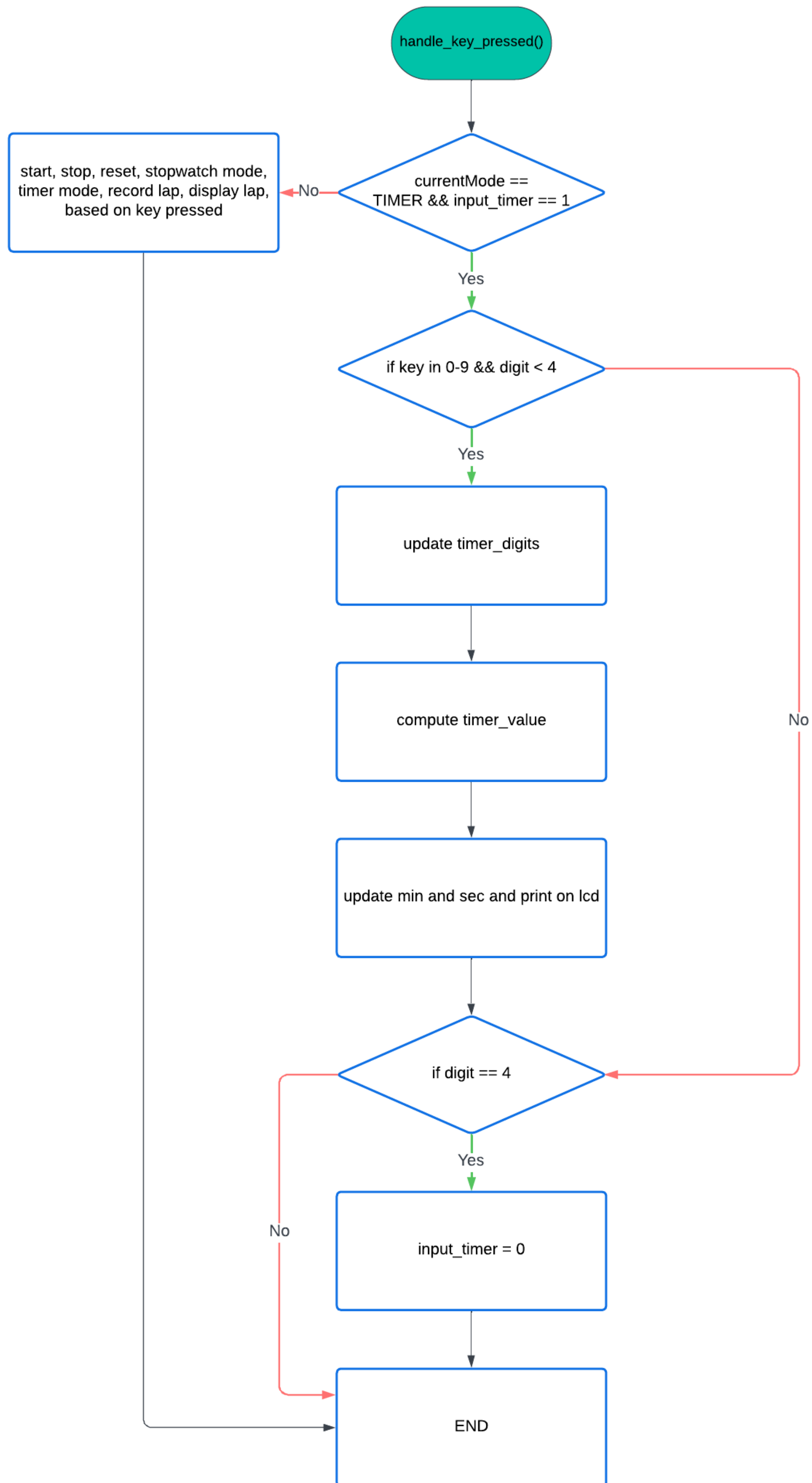
How It Works

User input is received via the keypad, where each key press is processed. The user can toggle between stopwatch and timer modes by pressing 4 or 5, respectively. The current mode and count values are clearly displayed on the LCD. TIM2 is set in order to time the 1 second count properly as well as frequently check to see what mode the device is operating in to run the timer or stopwatch accordingly.

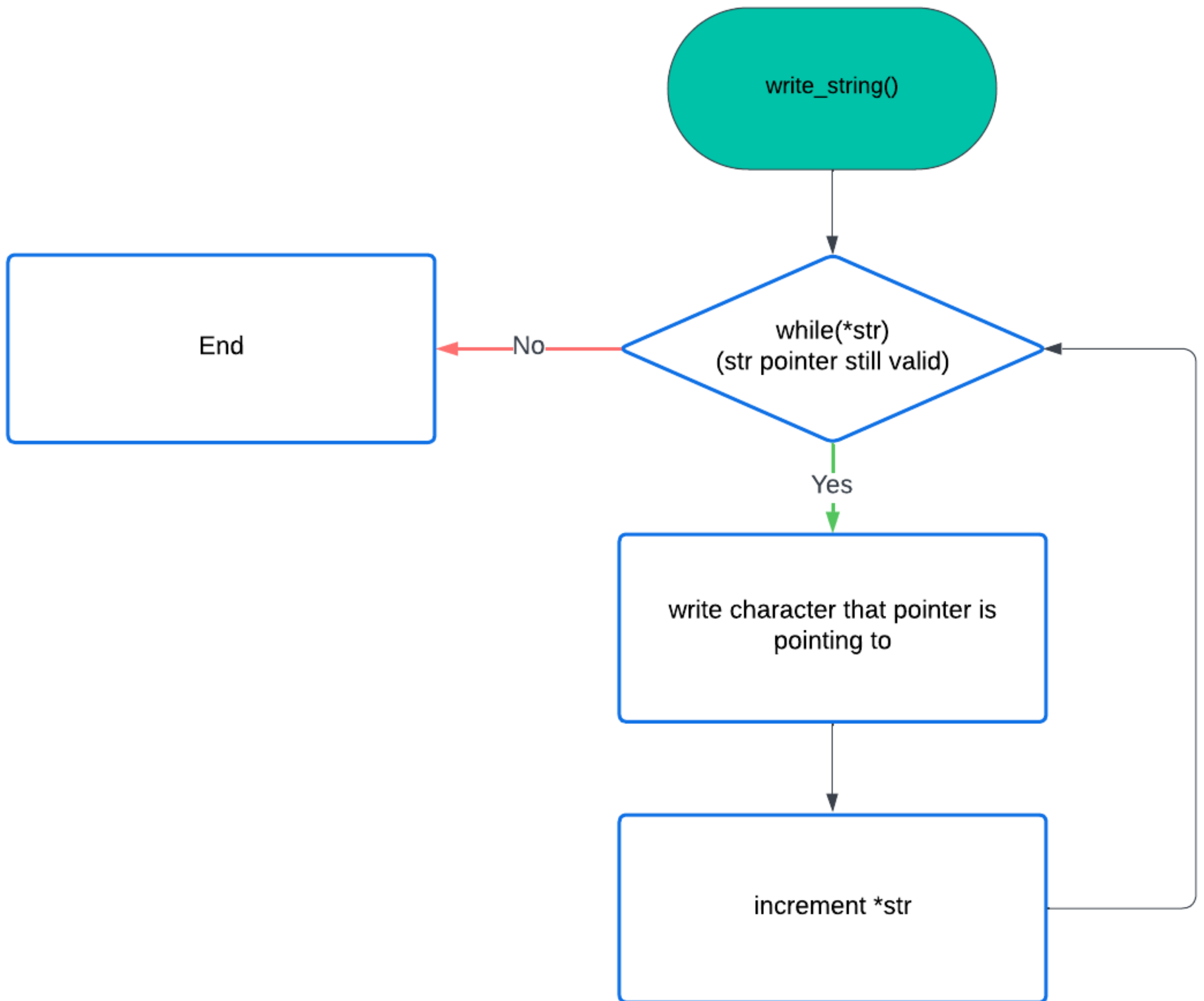
In stopwatch mode, the system can increment time, start/stop the count, and record lap times, which can be recalled with key 6 and displayed on the LCD using keys 7,8, and 9. The increment_count_update_LCD function updates the time displayed every second, while record_lap and display_lap manage the lap functionalities.

In timer mode, the user sets a countdown time using the keypad, and the start_timer function initiates the countdown. The decrement_count_update_LCD function handles the countdown, updating the display each second. When the timer reaches zero, the buzzer is activated with an 1000 Hz input wave, providing a loud alarm sound.

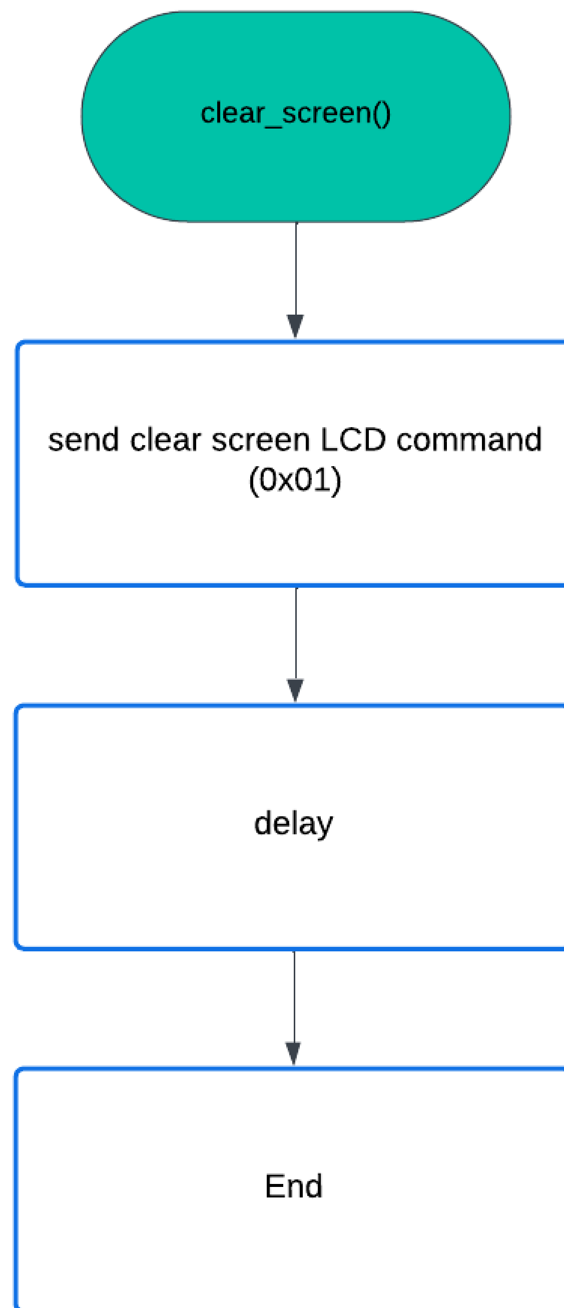

display_initial_position()
cursor_position(0,0)
write_string("Hello!
(4 or 5)")
cursor_postion(1,0)
write_string("00:00")
End

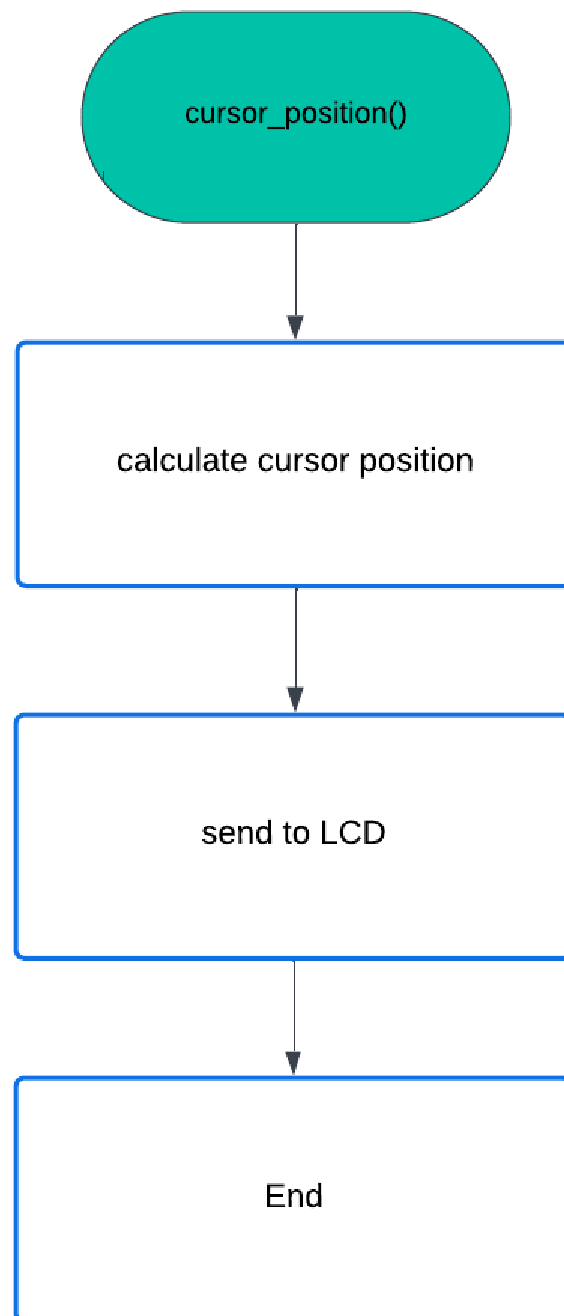


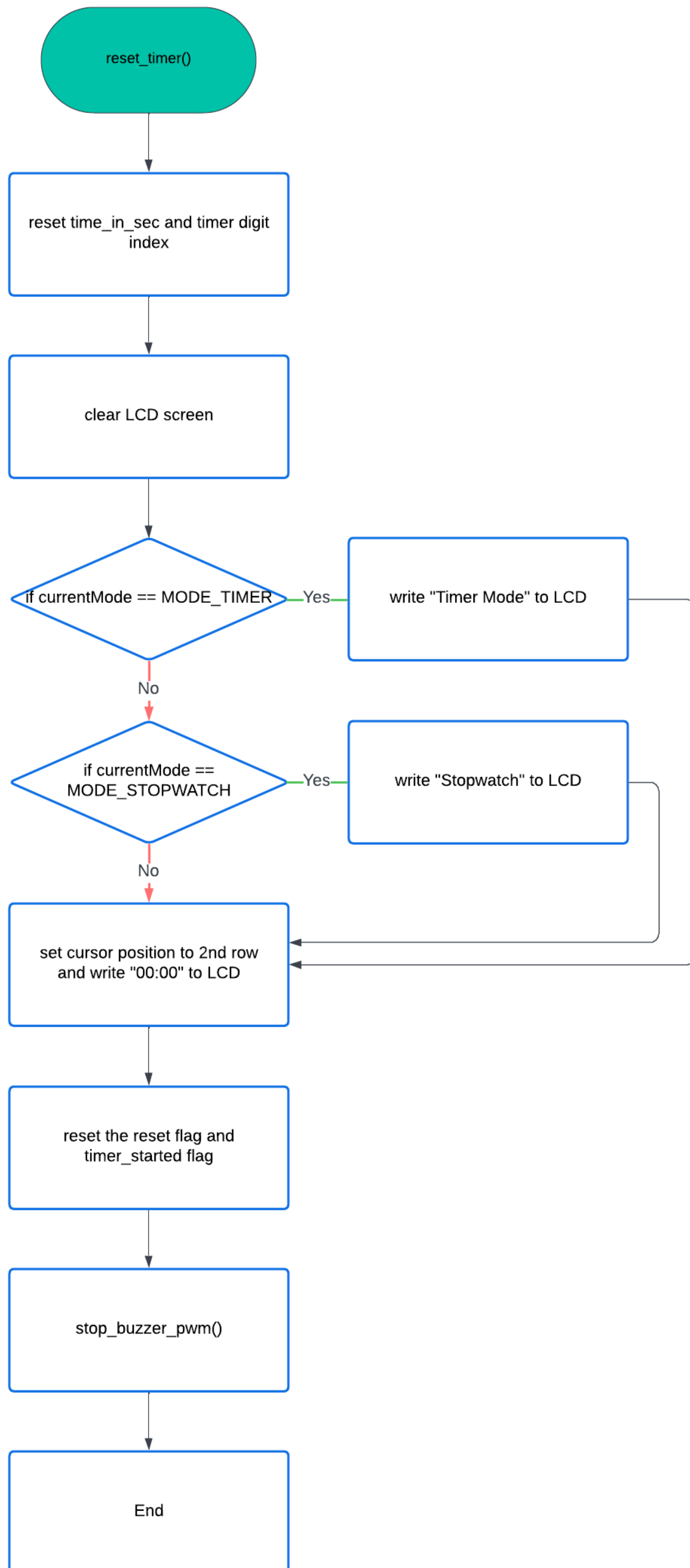
handle_key_pressed()
currentMode == TIMER && input_timer == 1
No
start, stop, reset, stopwatch mode, timer mode, record lap, display lap, based on key pressed
Yes
if key in 0-9 && digit < 4
Yes
update timer_digits
compute timer_value
update min and sec and print on lcd
No
if digit == 4
Yes
input_timer = 0
No
END

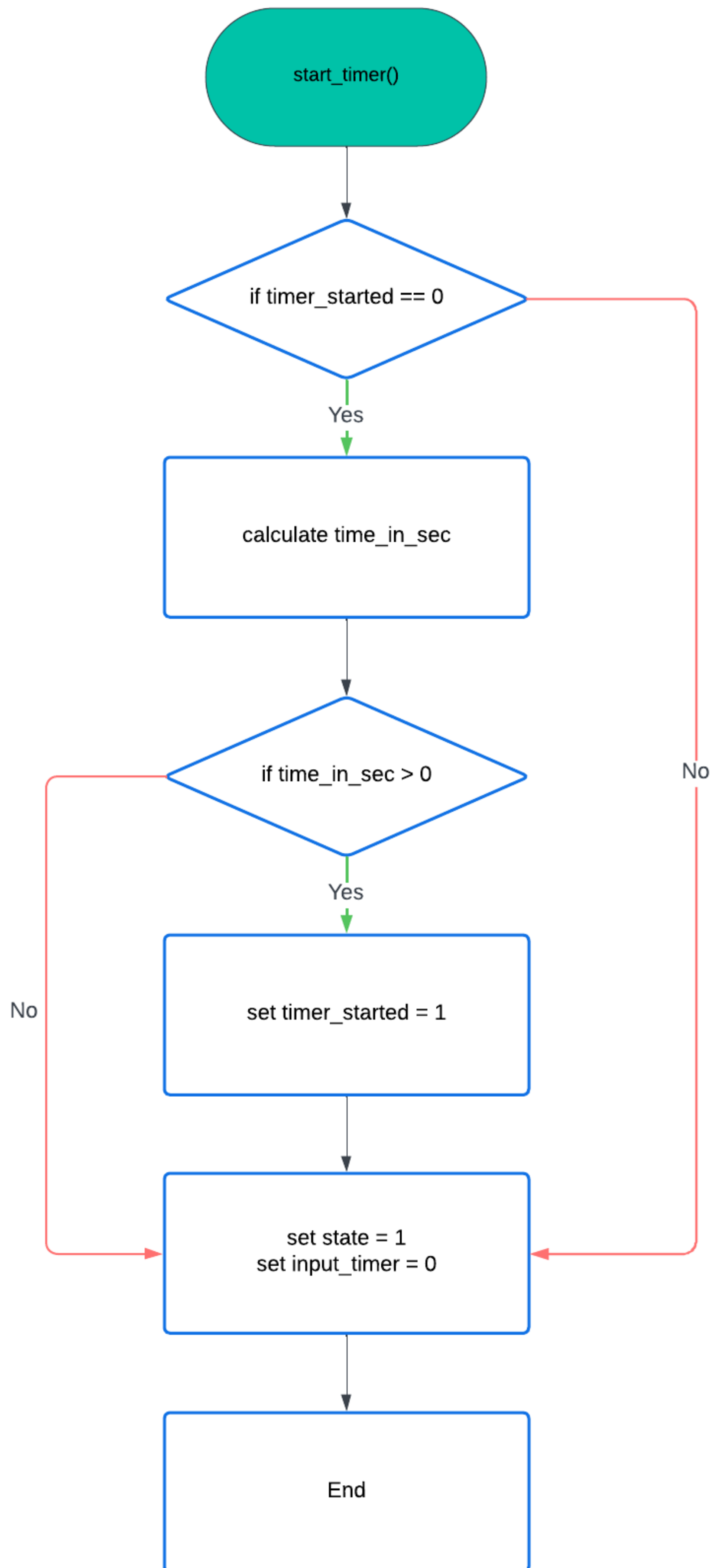


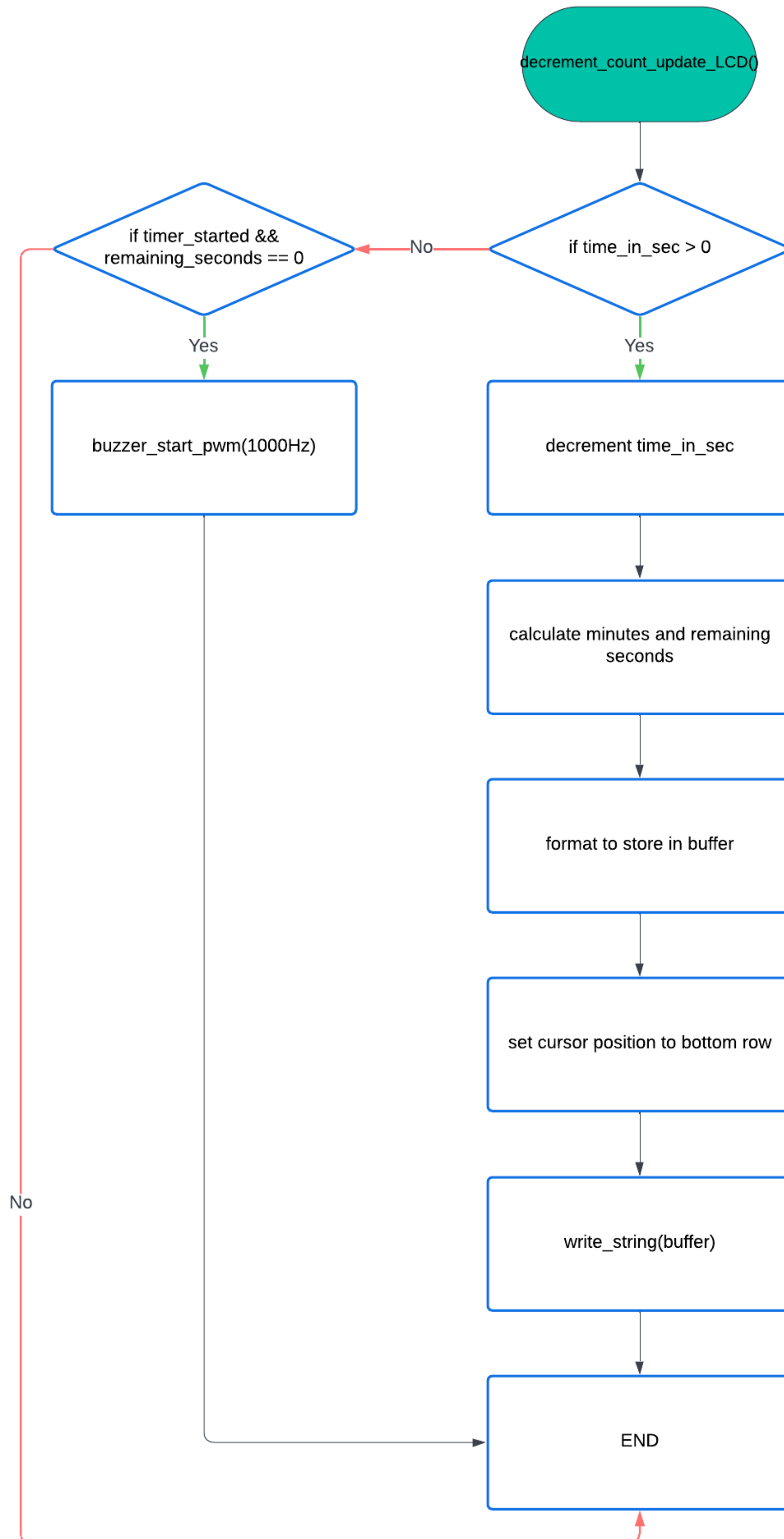
write_string()
End
No
while(*str)
(str pointer still valid)
Yes
write character that pointer is pointing to
increment *str


clear_screen()
send clear screen LCD command (0x01)
delay
End

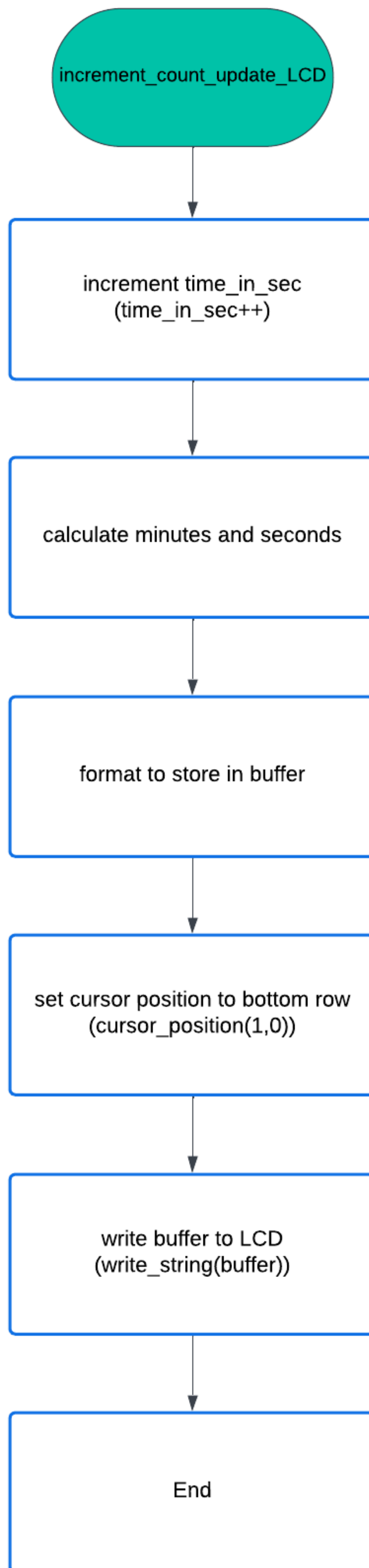

cursor_position()
calculate cursor position
send to LCD
End

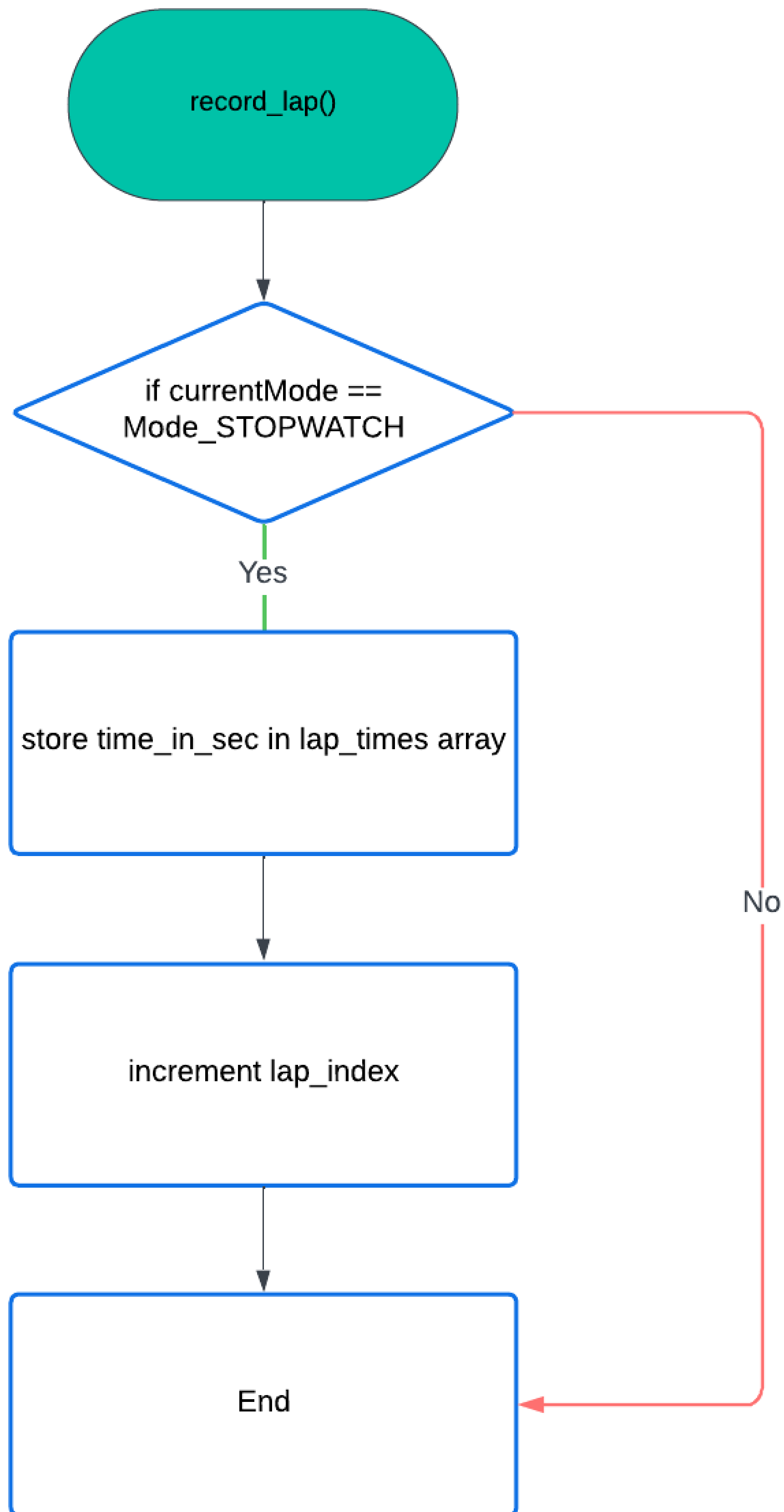

reset_timer()
reset time_in_sec and timer digit index
clear LCD screen
if currentMode == MODE_TIMER
Yes
write "Timer Mode" to LCD
No
if currentMode == MODE_STOPWATCH
Yes
write "Stopwatch" to LCD
No
set cursor position to 2nd row and write "00:00" to LCD
reset the reset flag and timer_started flag
stop_buzzer_pwm()
End

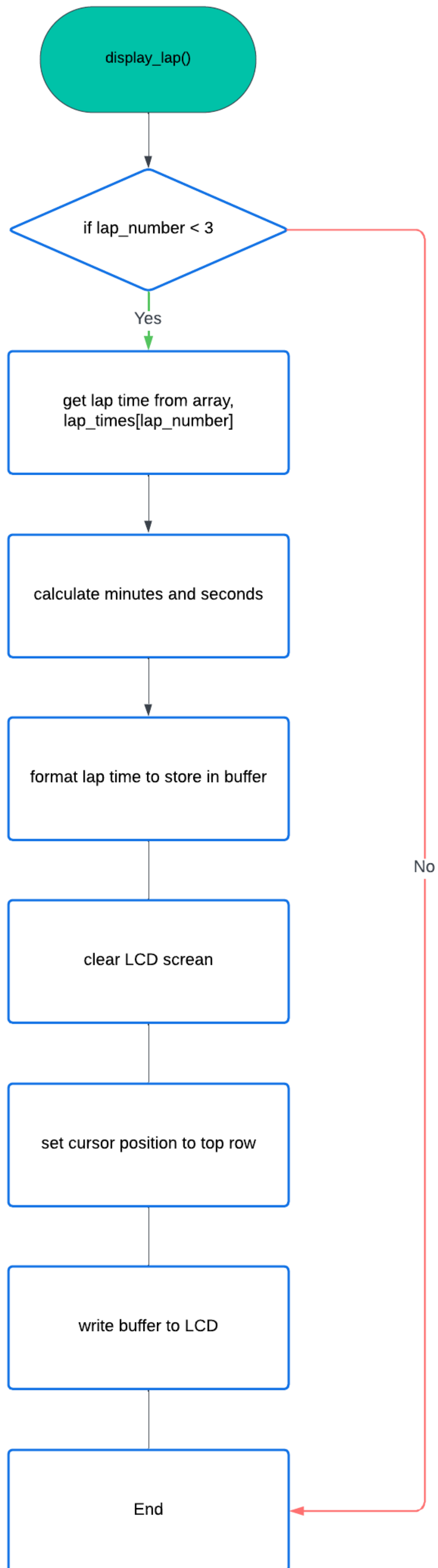

start_timer()
if timer_started == 0
Yes
calculate time_in_sec
if time_in_sec > 0
Yes
set timer_started = 1
No
No
set state = 1
set input_timer = 0
End

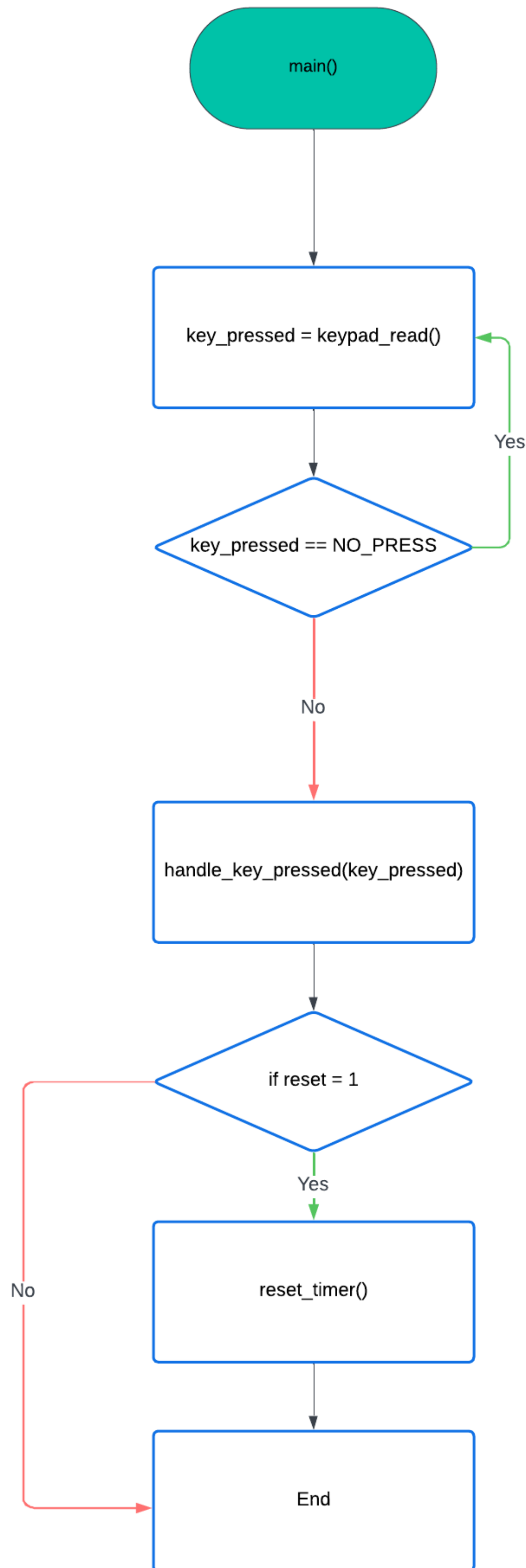


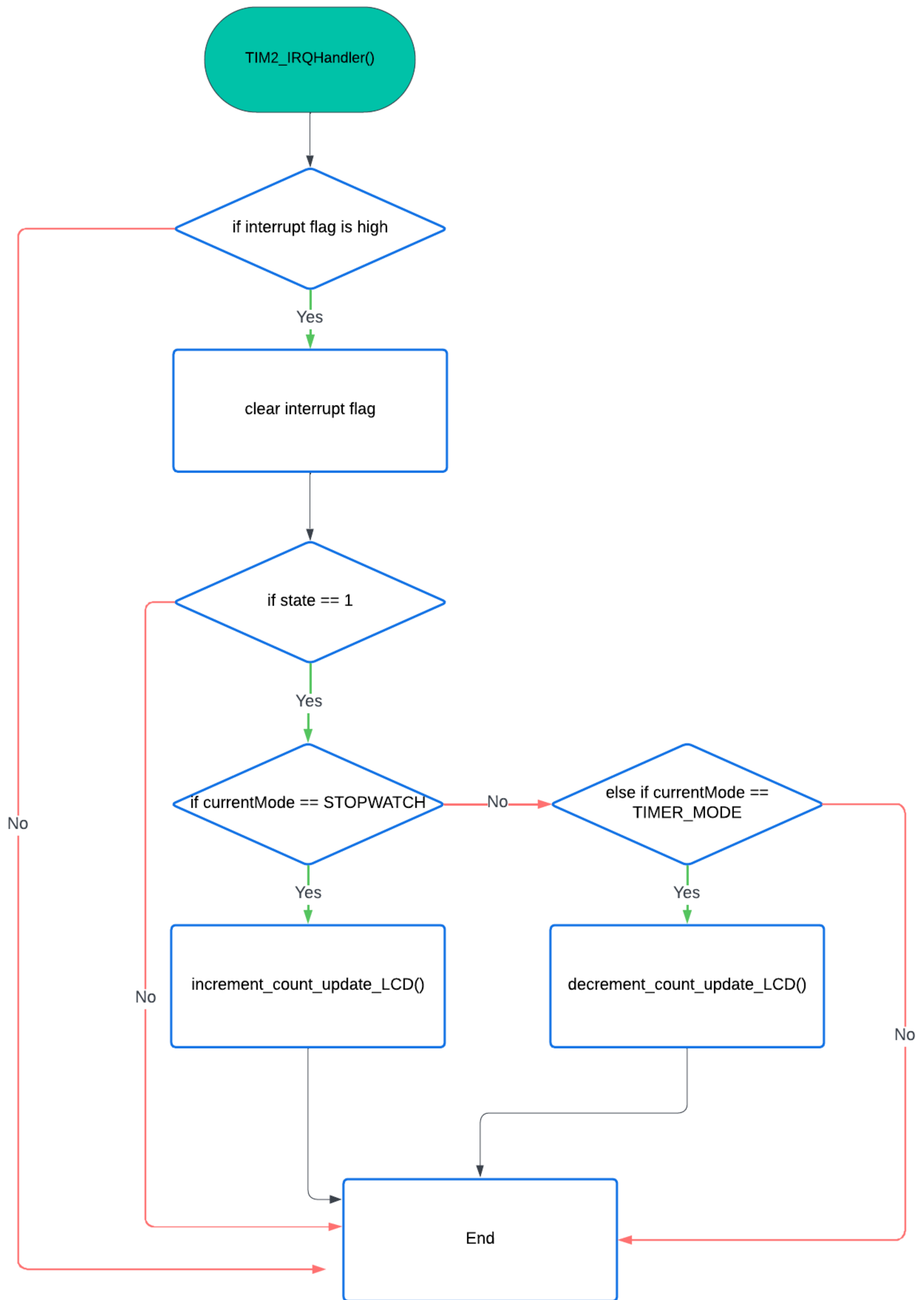
decrement_count_update_LCD()
if time_in_sec > 0
No
if timer_started &&
remaining_seconds == 0
Yes
Yes
buzzer_start_pwm(1000Hz)
decrement time_in_sec
calculate minutes and remaining
seconds
format to store in buffer
set cursor position to bottom row
No
write_string(buffer)
END

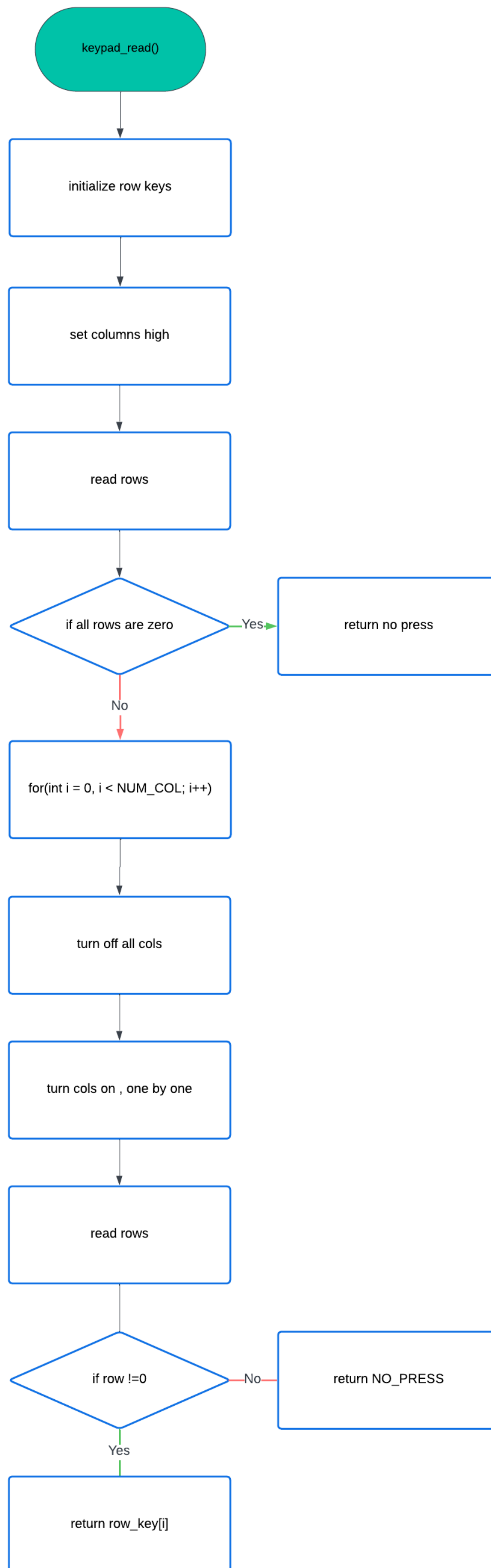

increment_count_update_LCD
increment time_in_sec
(time_in_sec++)
calculate minutes and seconds
format to store in buffer
set cursor position to bottom row
(cursor_position(1,0))
write buffer to LCD
(write_string(buffer))
End


record_lap()
if currentMode ==
Mode_STOPWATCH
Yes
No
store time_in_sec in lap_times array
increment lap_index
End


display_lap()
if lap_number < 3
Yes
get lap time from array,
lap_times[lap_number]
calculate minutes and seconds
format lap time to store in buffer
clear LCD screan
set cursor position to top row
write buffer to LCD
End
No


main()
key_pressed = keypad_read()
Yes
key_pressed == NO_PRESS
No
handle_key_pressed(key_pressed)
if reset = 1
Yes
No
reset_timer()
End


TIM2_IRQHandler()
if interrupt flag is high
Yes
clear interrupt flag
if state == 1
Yes
if currentMode == STOPWATCH
No
else if currentMode == TIMER_MODE
Yes
Yes
increment_count_update_LCD()
decrement_count_update_LCD()
No
No
No
End


keypad_read()
initialize row keys
set columns high
read rows
if all rows are zero
Yes
return no press
No
for(int i = 0, i < NUM_COL; i++)
turn off all cols
turn cols on , one by one
read rows
if row !=0
No
return NO_PRESS
Yes
return row_key[i]

Code

main.c

```
#include "main.h"
#include "lcd.h"
#include "stdio.h"
#include "keypad.h"
#include "buzzer.h"
#include "stopwatch.h"
#include <stdio.h>

///* Private variables ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void decrement_count_update_LCD(void);
void increment_count_update_LCD(void);
void TIM2_IRQHandler(void);
void timer_config(void);
void record_lap(void);
void display_lap(int lap_number);
void reset_timer(void);
void start_timer(void);
/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
	//necessary initializations and config
	HAL_Init();
	SystemClock_Config();
	LCD_init();
	keypad_init();
	timer_config();
	buzzer_init();
	buzzer_pwm_config();
	display_initial_message();
	while (1)
	{
		// check for press & debounce
		key_pressed = keypad_read();
		while (key_pressed == NO_PRESS) {
		  key_pressed = keypad_read();
		}
		while (keypad_read() != NO_PRESS);
		// handle press
		if (key_pressed != NO_PRESS) {
		  handle_key_pressed(key_pressed);
		}
		// if user resets
		if (reset) {
		  reset_timer();
		}
	}
}

void TIM2_IRQHandler(void) {
    if (TIM2->SR & TIM_SR_UIF) {
        TIM2->SR &= ~TIM_SR_UIF; // clear interrupt flag
        if (state == 1) {
		// handle modes accordingly
            if (currentMode == MODE_STOPWATCH) {
                increment_count_update_LCD();
            } else if (currentMode == MODE_TIMER) {
                decrement_count_update_LCD();
            }
        }
    }
}
```

keypad.h

```c
#ifndef SRC_KEYPAD_H_
#define SRC_KEYPAD_H_

#include "main.h"
#include <math.h>

#define NUM_COL 3
#define STAR 10
#define POUND 11
#define NO_PRESS 15

void keypad_init(void);
void config_rows(void);
void config_col(void);
uint8_t keypad_read(void);
void delay(uint32_t milliseconds);


#endif /* SRC_KEYPAD_H_ */
```

keypad.c

```
#include "keypad.h"
// initialization for keypad
void keypad_init(void)
{
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN; // Enable GPIOB clock
    config_rows();
    config_col();
}
// row config
void config_rows(void)
{
    // Configure rows as input with pull-down resistors
    GPIOB->MODER &= ~(GPIO_MODER_MODE4_Msk | GPIO_MODER_MODE5_Msk | GPIO_MODER_MODE6_Msk | GPIO_MODER_MODE7_Msk); // Set mode to input
    GPIOB->PUPDR &= ~(GPIO_PUPDR_PUPD4_Msk | GPIO_PUPDR_PUPD5_Msk | GPIO_PUPDR_PUPD6_Msk | GPIO_PUPDR_PUPD7_Msk); // Clear pull-down
    GPIOB->PUPDR |= (GPIO_PUPDR_PUPD4_1 | GPIO_PUPDR_PUPD5_1 | GPIO_PUPDR_PUPD6_1 | GPIO_PUPDR_PUPD7_1); // Enable pull-down
}
// col config
void config_col(void)
{
    // Configure columns as output, push-pull, low speed, no pull-up/down resistors
    GPIOB->MODER &= ~(GPIO_MODER_MODE8_Msk | GPIO_MODER_MODE9_Msk | GPIO_MODER_MODE10_Msk); // Clear mode
    GPIOB->MODER |= (1 << GPIO_MODER_MODE8_Pos) | (1 << GPIO_MODER_MODE9_Pos) | (1 << GPIO_MODER_MODE10_Pos); // Set mode to output
    GPIOB->OTYPER &= ~(GPIO_OTYPER_OT8_Msk | GPIO_OTYPER_OT9_Msk | GPIO_OTYPER_OT10_Msk); // Push-pull output
    GPIOB->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED8_Msk | GPIO_OSPEEDR_OSPEED9_Msk | GPIO_OSPEEDR_OSPEED10_Msk); // Low speed
    GPIOB->PUPDR &= ~(GPIO_PUPDR_PUPD8_Msk | GPIO_PUPDR_PUPD9_Msk | GPIO_PUPDR_PUPD10_Msk); // No resistor connected
}
// reading presses
uint8_t keypad_read(void)
{
    uint8_t row0_keys[NUM_COL] = {1, 2, 3};
    uint8_t row1_keys[NUM_COL] = {4, 5, 6};
    uint8_t row2_keys[NUM_COL] = {7, 8, 9};
    uint8_t row3_keys[NUM_COL] = {STAR, 0, POUND};

    // Set all columns to high
    GPIOB->ODR |= GPIO_ODR_OD8_Msk | GPIO_ODR_OD9_Msk | GPIO_ODR_OD10_Msk;

    // Read the rows
    uint8_t row0 = GPIOB->IDR & GPIO_IDR_ID4_Msk;
    uint8_t row1 = GPIOB->IDR & GPIO_IDR_ID5_Msk;
    uint8_t row2 = GPIOB->IDR & GPIO_IDR_ID6_Msk;
    uint8_t row3 = GPIOB->IDR & GPIO_IDR_ID7_Msk;

    // Check if any rows were pressed
    if ((row0 == 0) && (row1 == 0) && (row2 == 0) && (row3 == 0)) {
        return NO_PRESS;
    }

    for (int i = 0; i < NUM_COL; i++) {
        // Turn off all columns
        GPIOB->ODR &= ~(GPIO_ODR_OD8_Msk | GPIO_ODR_OD9_Msk | GPIO_ODR_OD10_Msk);

        // Turn on the column you want
        GPIOB->ODR |= (1 << (i + 8));

        // Small delay
        delay(10);

        // Read the rows
        row0 = GPIOB->IDR & GPIO_IDR_ID4_Msk;
        row1 = GPIOB->IDR & GPIO_IDR_ID5_Msk;
        row2 = GPIOB->IDR & GPIO_IDR_ID6_Msk;
        row3 = GPIOB->IDR & GPIO_IDR_ID7_Msk;

        if (row0 != 0) {
            return row0_keys[i];
        } else if (row1 != 0) {
            return row1_keys[i];
        } else if (row2 != 0) {
            return row2_keys[i];
        } else if (row3 != 0) {
            return row3_keys[i];
        }
    }

    return NO_PRESS;
}

void delay(uint32_t milliseconds)
{
    for (int i = 0; i < milliseconds; i++);
}
```

buzzer.h

```
#ifndef SRC_BUZZER_H_
#define SRC_BUZZER_H_
#include "main.h"
void buzzer_init(void);
void buzzer_pwm_config(void);
void buzzer_start_pwm(uint32_t frequency);
void buzzer_stop_pwm(void);
#endif /* SRC_BUZZER_H_ */
```

```c
#include "buzzer.h"

void buzzer_init(void)
{
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN; // enable GPIOA clock
    GPIOA->MODER &= ~(GPIO_MODER_MODE10); // clear mode for PA10
    GPIOA->MODER |= GPIO_MODER_MODE10_1;  // set PA10 to alternate function mode
    GPIOA->OTYPER &= ~(GPIO_OTYPER_OT10); // set PA10 as push-pull
    GPIOA->OSPEEDR |= GPIO_OSPEEDR_OSPEED10; // set PA10 to high speed
    GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPD10); // no pull-up, pull-down for PA10
    GPIOA->AFR[1] |= (1 << GPIO_AFRH_AFSEL10_Pos); // set alternate function 1 (TIM1_CH3) for PA10
}

void buzzer_pwm_config(void)
{
    RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; // enable TIM1 clock
    TIM1->PSC = 0; // no prescaler
    TIM1->ARR = 3999; // ARR val
    TIM1->CCR3 = 2000; // CCR for 50% duty
    TIM1->CCMR2 &= ~(TIM_CCMR2_OC3M); //clear output compare mode bits
    TIM1->CCMR2 |= (TIM_CCMR2_OC3M_1 | TIM_CCMR2_OC3M_2); // set output compare mode to PWM mode 1
    TIM1->CCER |= TIM_CCER_CC3E; // enable the capture/compare channel 3
    TIM1->BDTR |= TIM_BDTR_MOE; // output enable
}
// pulse with desired frequency for buzzer
void buzzer_start_pwm(uint32_t frequency)
{
    TIM1->ARR = (1000000 / frequency) - 1; // set arr based on desired frequency
    TIM1->CCR3 = (TIM1->ARR + 1) / 2; // set duty cycle to 50%
    TIM1->CR1 |= TIM_CR1_CEN; // enable timer
}
// stop buzzer
void buzzer_stop_pwm(void)
{
    TIM1->CR1 &= ~TIM_CR1_CEN; // Disable the timer
}
```

lcd.h

```c
#include <stdint.h>
#ifndef SRC_LCD_H_
#define SRC_LCD_H_


#define LCD_PORT GPIOC
#define LCD_RS GPIO_PIN_5
#define LCD_EN GPIO_PIN_4
#define DATA_BITS (GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3)

void LCD_init(void);
void enable(void);
void nybble(uint8_t command);
void write_char(uint8_t ch);
void cursor_position(uint8_t row, uint8_t col);
void write_string(char str[]);
void LCD_clear(void);
void delay_us(const uint32_t us);
void SysTick_Init(void);
void handle_key_pressed(int key);
void display_initial_message(void);
void high_nybble(uint8_t command);
void LCD_GPIO_Config(void);

#endif /* SRC_LCD_H_ */
```

lcd.c

```
// references: referred to EE329 lab manual and LCD data sheet for initialization guidance
#include "lcd.h"
#include "main.h"
#include "stdio.h"
#include "stopwatch.h"
#include "timer.h"
#include <string.h>

void LCD_init( void )  {
        LCD_GPIO_Config();
    delay_us(50000); // power-up wait more than 40 ms
    for (int i = 0; i < 3; i++){  // wake up #1,#2,#3
        high_nybble(0x30);// 0x30 on output port
        delay_us(200); // delay more than 160us
    }
    high_nybble(0x20); // put 20 on the output port
    delay_us(5000);
    nybble(0x28); //4 bit 2 line
    nybble(0x10); //set cursor
    nybble(0x0F); //display on and cursor blink
    nybble(0x06); //entry mode set
    nybble(0x01); // clear
    delay_us(50000);
}

// start message
void display_initial_message(void) {
    cursor_position(0, 0);
    write_string("Hello! (4 or 5)"); // stopwatch is 4, timer is 5
    cursor_position(1, 0);
    write_string("00:00");
}

// handle key pressed on LCD
void handle_key_pressed(int key)
{
        // check if in timer mode
    if (currentMode == MODE_TIMER && input_timer == 1) {
        // check if digit is in range and count of timer inputs is less than 4
        if (key >= 0 && key <= 9 && digit < 4) {
        // store timer value, get time and print to lcd
            timer_digits[digit++] = key;
            int timer_value = timer_digits[0] * 600 + timer_digits[1] * 60 + timer_digits[2] * 10 + timer_digits[3];
            minutes = timer_value / 60;
            seconds = timer_value % 60;
            char buffer[6];
            sprintf(buffer, "%02d:%02d", minutes, seconds);
            cursor_position(1, 0);
            write_string(buffer);
        }
        // timer done taking inputs
        if (digit == 4){
        input_timer = 0;
        }
    } else {
        switch (key) {
        // start
            case 1:
                if (currentMode == MODE_TIMER && digit == 4) {
                    start_timer();
                } else {
                    state = 1;
                }
                break;
            // stop
            case 2:
                state = 0;
                break;
            // reset
            case 3:
                reset = 1;
                break;
            // start stopwatch case
            case 4:
                 start_stopwatch();
                break;
            // timer mode
            case 5:
                currentMode = MODE_TIMER;
                reset = 1;
                input_timer = 1;
                digit = 0;
                break;
            // recording and displaying laps 6-9
            case 6:
                record_lap();
                break;
            case 7:
                display_lap(0);
```

```c
                break;
            case 8:
                display_lap(1);
                break;
            case 9:
                display_lap(2);
                break;
            default:
                break;
        }
    }
}

// enable pulse
void enable(void)  {
   LCD_PORT->ODR |=(LCD_EN); // set enable high
   delay_us(30); // wait
   LCD_PORT->ODR &= ~(LCD_EN); // set enable low
   delay_us(30);
}

// mainly for config/initialization purposes
void high_nybble( uint8_t command )  {
   LCD_PORT->ODR &= ~DATA_BITS; // clear data bits
   LCD_PORT->ODR |= (command >> 4); // set upper bits
   delay_us(30);
   enable(); //enable pulse to send command
}
// high and low nybble
void nybble( uint8_t command )  {
   LCD_PORT->ODR &= ~(DATA_BITS); // clear data bits
   LCD_PORT->ODR |= ((command>>4)&DATA_BITS); // high nybble
   delay_us(30); // short delay for setup
   enable(); // pulse enable to latch the high nybble in order for LCD to read
   LCD_PORT->ODR   &= ~(DATA_BITS ); //clear data bits
   LCD_PORT->ODR |= (command & DATA_BITS); // low nybble
   delay_us(30); // short delay for setup
   enable(); // pulse enable for LCD to read low nybble
}

//write single char
void write_char(uint8_t ch){
   LCD_PORT->ODR |= (LCD_RS);// set RS high for incoming data
   delay_us(30);
   nybble(ch); // call nybble with character to print
   LCD_PORT->ODR &= ~(LCD_RS);// set RS low afterwards
}
// write string
void write_string(char *str) {
    while (*str) {
        write_char(*str++);
    }
}

// enable systick timer, selecting cpu clk, and disabling interrupt
void SysTick_Init(void) {
    SysTick->CTRL |= (SysTick_CTRL_ENABLE_Msk | SysTick_CTRL_CLKSOURCE_Msk);
    SysTick->CTRL &= ~(SysTick_CTRL_TICKINT_Msk);
}

void delay_us(const uint32_t us) {
    // set the counts for the specified delay
    SysTick->LOAD = (uint32_t)((us * (SystemCoreClock / 1000000)) - 1);
    SysTick->VAL = 0; // clear timer cnt
    SysTick->CTRL &= ~(SysTick_CTRL_COUNTFLAG_Msk); // clear cnt flag
    while (!(SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk)); // wait for flag
}

//clear screen
void LCD_clear(void) {
    nybble(0x01); //clear display command
    delay_us(5000); //delay
}

// setting cursor position
void cursor_position(uint8_t row, uint8_t col) {
    uint8_t position = (row == 0) ? (0x80 + col) : (0xC0 + col);
    nybble(position);
}
// lcd port gpio config
void LCD_GPIO_Config(void){
    RCC->AHB2ENR   |=   (RCC_AHB2ENR_GPIOCEN);   //enable clock

    // gpio for data bits and RS and EN
    LCD_PORT->MODER &= ~(GPIO_MODER_MODE0 | GPIO_MODER_MODE1 |
                                         GPIO_MODER_MODE2 | GPIO_MODER_MODE3 |
                                         GPIO_MODER_MODE4 | GPIO_MODER_MODE5);

    LCD_PORT->MODER |= (GPIO_MODER_MODE0_0 | GPIO_MODER_MODE1_0 |
                                        GPIO_MODER_MODE2_0 | GPIO_MODER_MODE3_0 |
                                        GPIO_MODER_MODE4_0 | GPIO_MODER_MODE5_0);
    LCD_PORT->OSPEEDR |= ((3 << GPIO_OSPEEDR_OSPEED0_Pos) | (3 << GPIO_OSPEEDR_OSPEED1_Pos) | (3 << GPIO_OSPEEDR_OSPEED2_Pos) |
```

```
                                                      (3 << GPIO_OSPEEDR_OSPEED3_Pos) |(3 << GPIO_OSPEEDR_OSPEED4_Pos) | (3 <<
GPIO_OSPEEDR_OSPEED5_Pos));

    LCD_PORT->PUPDR &= ~(GPIO_PUPDR_PUPD0 | GPIO_PUPDR_PUPD1 |
                                            GPIO_PUPDR_PUPD2 | GPIO_PUPDR_PUPD3 |
                                            GPIO_PUPDR_PUPD4 | GPIO_PUPDR_PUPD5);
    // all outputs start off
    LCD_PORT->BRR = (DATA_BITS | LCD_EN | LCD_RS);
}
```

```
#ifndef SRC_STOPWATCH_H_
#define SRC_STOPWATCH_H_
#include <stdio.h>
#include "lcd.h"
void display_lap(int lap_number);
void record_lap(void);
void increment_count_update_LCD(void);
void start_stopwatch(void);
extern int key_pressed; // stores the integer of the keypress
extern int state; // state of start button
extern int digit; // used to track presses after timer mode set
extern int reset; // rest flag
extern int timer_started; // timer started flag
extern int time_in_sec; // time
extern int lap_times[3]; // store lapping times
extern int lap_index; // index used for lap times
extern const int SECONDS_PER_MINUTE; // 60 seconds in a minute
extern int remaining_seconds;
extern int minutes;
extern int seconds;
typedef enum {
    MODE_STOPWATCH,
    MODE_TIMER
} Mode;
extern Mode currentMode;

#endif /* SRC_STOPWATCH_H_ */
```

```c
#include "stopwatch.h"
int key_pressed = 0; // store keypress
int state = 0; // state of "start" button
int digit = 0;
int reset = 0;// used as a flag that resets digit, time_in_sec key and LCD
int timer_started = 0; // Flag to indicate if the timer was started with a valid time
int time_in_sec; // stores the total timer value in seconds
int lap_times[3] = {0, 0, 0};
int lap_index = 0; // Index to store the next lap time
const int SECONDS_PER_MINUTE = 60; // 60 seconds = minute
int remaining_seconds;
int minutes;
int seconds;
Mode currentMode;

// increment time by 1 second
void increment_count_update_LCD(void) {
    time_in_sec++;
    minutes = time_in_sec / SECONDS_PER_MINUTE;
    seconds = time_in_sec % SECONDS_PER_MINUTE;
    // store in buffer and print
    char buffer[6];
    sprintf(buffer, "%02d:%02d", minutes, seconds);
    cursor_position(1, 0);
    write_string(buffer);
}
void start_stopwatch(void){
    currentMode = MODE_STOPWATCH;
    LCD_clear();
    cursor_position(0, 0);
    write_string("Stopwatch");
    cursor_position(1, 0);
    write_string("00:00");
    reset = 1;
}
// recording laps
void record_lap(void)
{
        if(currentMode == MODE_STOPWATCH){
                lap_times[lap_index] = time_in_sec;
                lap_index = (lap_index + 1) % 3; // continuosly index thru array of 3 to rewrite new laps
        }
}
// displaying laps
void display_lap(int lap_number)
{
    if (lap_number < 3) {
        int lap_time = lap_times[lap_number]; // grab lap time from array
        int minutes = lap_time / SECONDS_PER_MINUTE;
        int remaining_seconds = lap_time % SECONDS_PER_MINUTE;
        // store in buffer and print
        char buffer[32];
        sprintf(buffer, "Lap %d: %02d:%02d", lap_number + 1, minutes, remaining_seconds);
        LCD_clear();
        cursor_position(0, 0);
        write_string(buffer);
    }
}
```

Timer.h

```
#ifndef SRC_TIMER_H_
#define SRC_TIMER_H_
void reset_timer(void);
void start_timer(void);
void decrement_count_update_LCD(void);
#include "stopwatch.h"
#include "lcd.h"
#include "buzzer.h"
extern int input_timer; // flag to check if timer has received full 4 digit input yet
extern int timer_digits[4]; // to store 4 digits for timer
void timer_config(void);
#endif /* SRC_TIMER_H_ */
```

Timer.c

```c
#include "timer.h"
int input_timer = 0; // flag for when timer's time is set
int timer_digits[4] = {0, 0, 0, 0}; // stores user input timer time

// reset time to 00:00
void reset_timer(void) {
  // reset counts
  time_in_sec = 0;
  digit = 0;
  LCD_clear();
  // check mode
  if(currentMode == MODE_TIMER){
         write_string("Timer Mode");
  }
  if(currentMode == MODE_STOPWATCH){
         write_string("Stopwatch");
  }
  cursor_position(1, 0);
  write_string("00:00");
  reset = 0;
  timer_started = 0;
  buzzer_stop_pwm();

}

// timer configuration to count up and down by one second properly
void timer_config(void) {
    RCC->APB1ENR1 |= RCC_APB1ENR1_TIM2EN; // enable TIM2 clock

    TIM2->CR1 &= ~(TIM_CR1_CEN); // disable counter
    TIM2->CR1 &= ~(TIM_CR1_DIR); // set upcounting
    TIM2->CR1 &= ~(TIM_CR1_CMS); // set to edge-aligned mode

    TIM2->SR &= ~TIM_SR_UIF; // clear update interrupt flag
    TIM2->DIER |= TIM_DIER_UIE; // enable update interrupt

    TIM2->PSC = 0; // no prescaler
    TIM2->ARR = 4000000 - 1; // set arr for 1 second (considering 4MHz clock)
        NVIC->ISER[0]= (1 << (TIM2_IRQn & 0x1F)); //enable interrupt
    TIM2->CR1 |= TIM_CR1_CEN; // enable
}

// start the timer if ready
void start_timer(void) {
        // check if timer started, set timer value, set flags
        if(!timer_started){
                time_in_sec = timer_digits[0] * 600 + timer_digits[1] * 60 + timer_digits[2] * 10 + timer_digits[3];
                if(time_in_sec > 0){
                timer_started = 1;
                }
        }
        state = 1;
        input_timer = 0;
}

// decrement time for timer
void decrement_count_update_LCD(void) {
        // decrememnt until 00:00 reached
        if (time_in_sec > 0) {
         time_in_sec--;
         // get time, store in buffer, print
         minutes = time_in_sec / SECONDS_PER_MINUTE;
         remaining_seconds = time_in_sec % SECONDS_PER_MINUTE;
         char buffer[6];
         sprintf(buffer, "%02d:%02d", minutes, remaining_seconds);
         cursor_position(1, 0);
         write_string(buffer);
         // if timer done, set off buzzer @ 1kHz
    } else if(timer_started && remaining_seconds == 0){
        buzzer_start_pwm(1000);
    }
}
```

Owner's Manual for Stopwatch and Timer

Contents

1. Introduction
2. Safety Instructions
3. Setup
4. Operation
5. Switching Between Modes
6. Troubleshooting
7. Specifications
8. Customer Support

Introduction

Congratulations on your new Stopwatch and Timer! This manual will guide you through the setup and usage of your device. Please read this manual in its entirety to ensure the best possible performance.

Components

Here are the main components used in this project:

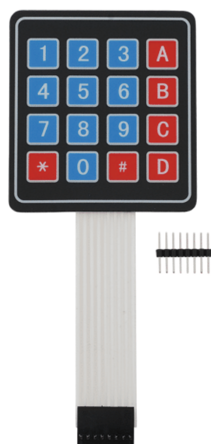


Figure 1: Keypad

Figure 2: Buzzer

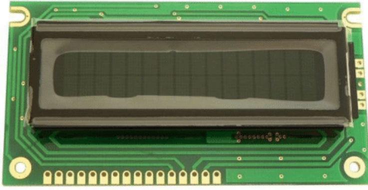


Figure 3: LCD

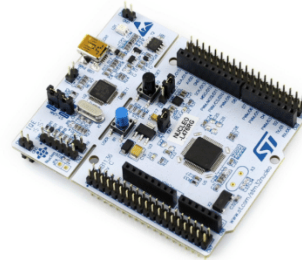


Figure 4: Nucleo Board

Safety Instructions

1. Read these instructions carefully before use.
2. Keep these instructions for future reference.
3. Do not use the device near water.
4. Clean only with a dry cloth.

Setup

1. Plug the STM32 Nucleo L476RG board into your computer using a USB cable.
2. Upload the provided code to the STM32 board using your preferred development environment (e.g., STM32CubeIDE).
3. Once the code is uploaded and running, the LCD screen will prompt you to select between stopwatch mode or timer mode by pressing 4 or 5 on the keypad.

Operation

Stopwatch Mode

To enter stopwatch mode, press **4** on the keypad. The following operations are available:

- **1** - Start the stopwatch.
- **2** - Stop the stopwatch.
- **3** - Reset the stopwatch.
- **6** - Record a lap time (up to three different lap times can be recorded).
- **7**, **8**, **9** - View the recorded lap times.

Timer Mode

To enter timer mode, press **5** on the keypad. You will then be prompted to enter a 4-digit countdown time. After entering the time, the following operations are available:

- **1** - Start the timer.
- **2** - Stop the timer.
- **3** - Reset the timer.

Switching Between Modes

To switch between stopwatch and timer modes at any time, simply press **4** for stopwatch mode or **5** for timer mode.

Troubleshooting

- **LCD not displaying correctly:** Ensure that the STM32 board is properly connected and the code is correctly uploaded.
- **Keypad inputs not responding:** Check the connections between the keypad and the STM32 board.
- **Buzzer not sounding:** Verify the buzzer connections and ensure that the buzzer is correctly initialized in the code.

Specifications

- **Microcontroller:** STM32 Nucleo L476RG
- **Display:** 16x2 LCD Screen
- **Input:** 4x4 Keypad
- **Output:** Passive Buzzer

Customer Support

If you have any questions or need further assistance, please contact our customer support team at `sghezava@calpoly.edu`.

References

1. STMicroelectronics, "UM1724 User manual - STM32 Nucleo-64 boards (MB1136)," STMicroelectronics. [Online]. Available: https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf. [Accessed: Jun. 5, 2024].
2. STMicroelectronics, "STM32L476xx Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 1MB Flash, 128 KB SRAM, USB OTG FS, LCD, ext. SMPS," STMicroelectronics. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32l476rg.pdf. [Accessed: Jun. 5, 2024].
3. STMicroelectronics, "RM0351 Reference manual - STM32L47xxx, STM32L48xxx, STM32L49xxx, and STM32L4Axxx advanced Arm®-based 32-bit MCUs," STMicroelectronics. [Online]. Available: https://www.st.com/resource/en/reference_manual/rm0351-stm32l47xxx-stm32l48xxx-stm32l49xxxand-stm32l4axxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf. [Accessed: Jun. 5, 2024].
4. EE329, "EE329 Lab Manual," Google Docs. [Online]. Available: https://docs.google.com/document/d/1xA1AfZJOFKy3r1e9WllB8o4h1oyOlbHH/edit?rtpof=true. [Accessed: Jun. 5, 2024].
5. CPE316, "STM32 Lab Manual," Google Docs. [Online]. Available: https://docs.google.com/document/d/1Btl--IQGtYRRn8naFpLwn64Av9y7no5OkXmoK1pFN4g/edit. [Accessed: Jun. 5, 2024].
6. POWERTIP, "PC 1602-H Datasheet," [Online]. Available: https://www.powertipusa.com/pdf/pc1602h.pdf. [Accessed: Jun. 5, 2024].
7. Newhaven Display, "NHD-0216HZ-FSW-FBW-33V3C Datasheet," [Online]. Available: https://newhavendisplay.com/content/specs/NHD-0216HZ-FSW-FBW-33V3C.pdf. [Accessed: Jun. 5, 2024].