UNIVERSITI TEKNOLOGI MALAYSIA

SEMESTER 2024/2025-1
SKEE 3223-08 MICROPROCESSOR
PROJECT TITLE- SMART FLUORESCENT LAMP

No.	Full Name	Picture	Matric No.	Section	Tasks
1	DESMOND HO PIN FENG		A23KE0081	08	- Document Manager - Problem Identify - Problem Solving - Co-leader
2	LIM KAI MING		A23KE0169	08	- Researcher - Hardware Built - Software Prototype - Solution developer coordinator
3	OOI CHUN JIE		A23KE0301	08	- Leader - Software Prototype - Project Flow - Circuit Diagram
4	YEOH JIA RUI		A23KE0364	08	- Video Editor - Researcher - Problem Identify - Solution developer coordinator

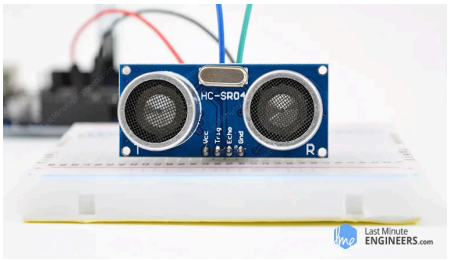
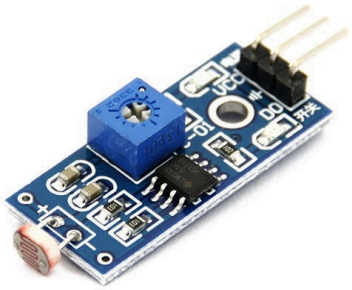
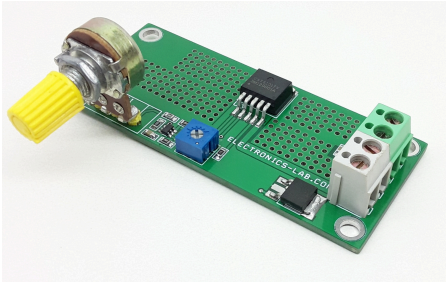
Youtube Link: https://youtu.be/6-sAWO5dNFM

Github Link: https://github.com/ChunJie888/Smart-Fluorescent-Lamp-.git

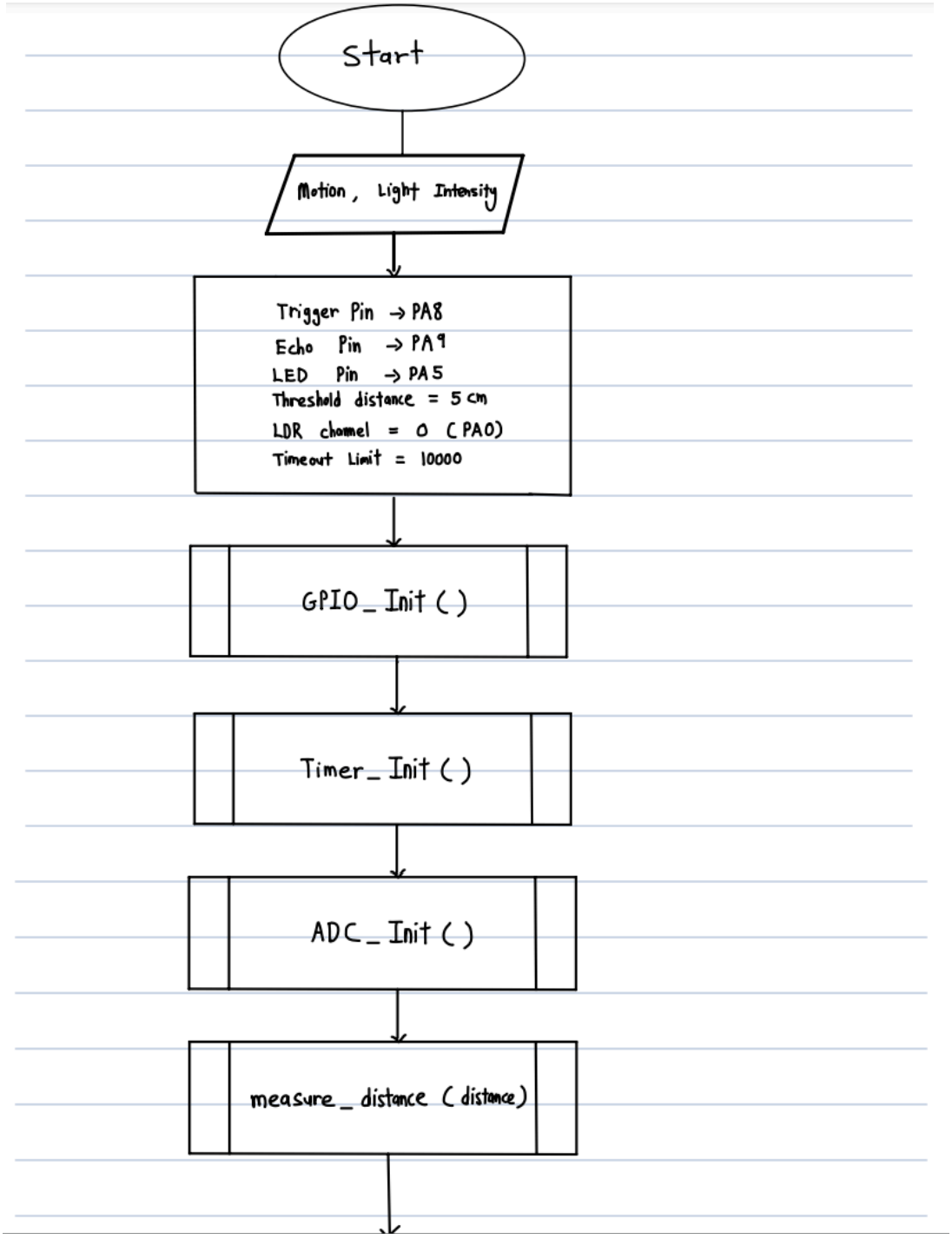
1.0 ABSTRACT

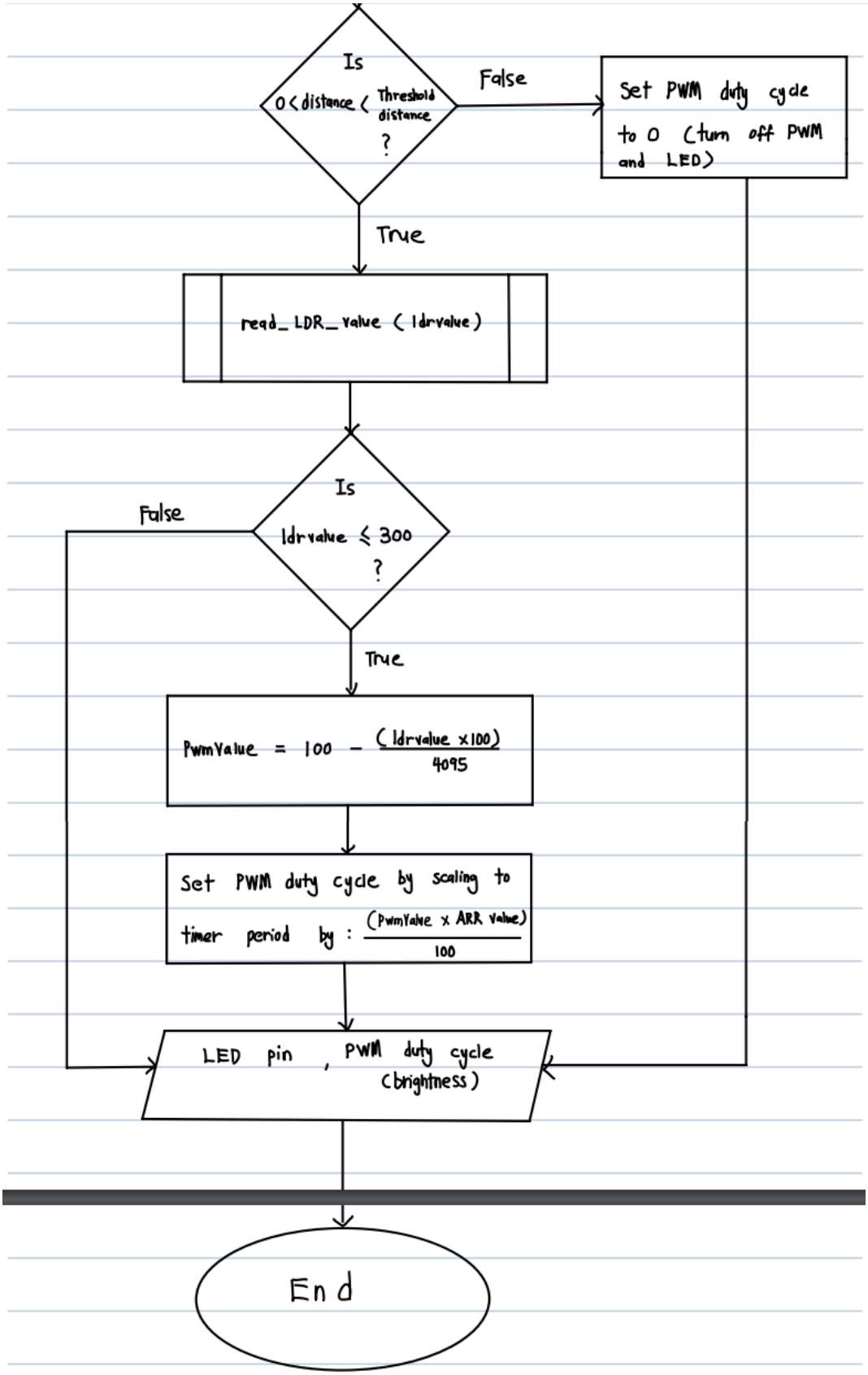
This project is to design and implement a Smart Fluorescent Lamp System aimed at enhancing energy and user convenience.The system is design by 4 main components STM-32 Microcontroller, an ultrasonic sensor, a LDR(Light Dependent Resistor) and PWM-Controlled LED to enable automatic operation based on certain criteria and ambient light levels. STM-32 Microcontroller is the central control unit responsible for interfacing with the sensors and controlling the LED. Ultrasonic sensor is a motion detector that detects the presence of a person within a predefined range, the lamp is triggering to active when needed. LDR (Light Dependent Resistor) detects the intensity of the light. The LDR continuously monitors ambient light intensity, allowing the system to dynamically adjust the lamp's brightness via pulse-width modulation (PWM) to maintain optimal illumination and conserve energy. The implementation of the STM32's Microcontroller, ADC, and timer peripherals to ensure precise sensor readings and responsive control. With potential applications in homes, offices, and public spaces, the Smart Fluorescent Lamp System is a sustainable and user-friendly lighting solution that leverages advanced sensor technology and precise microcontroller operations. By dynamically responding to motion and ambient light, the system optimizes energy consumption and enhances user convenience. This innovative design has significant potential for deployment in various residential, commercial, and public settings.

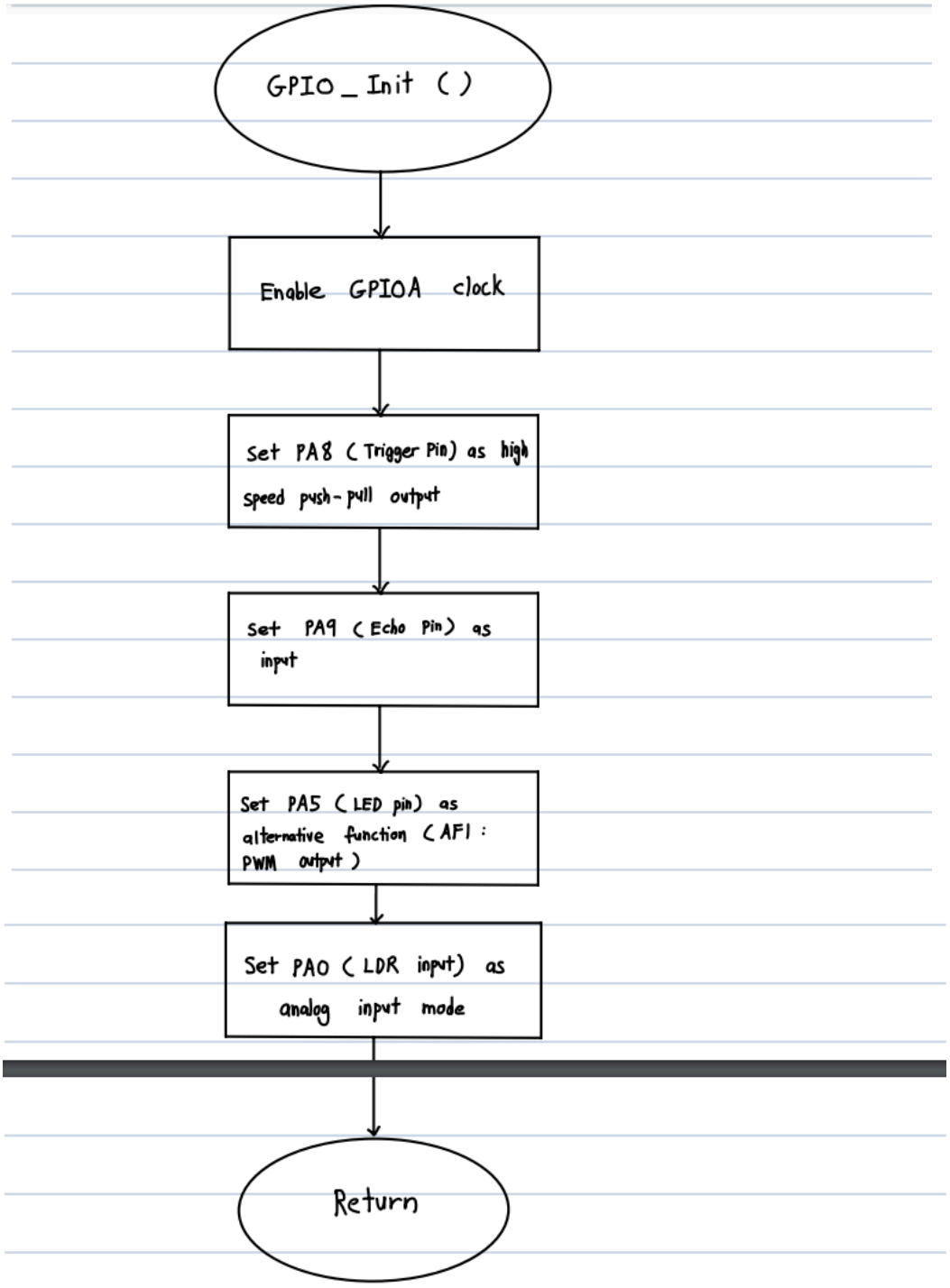
2.0 COMPONENTS

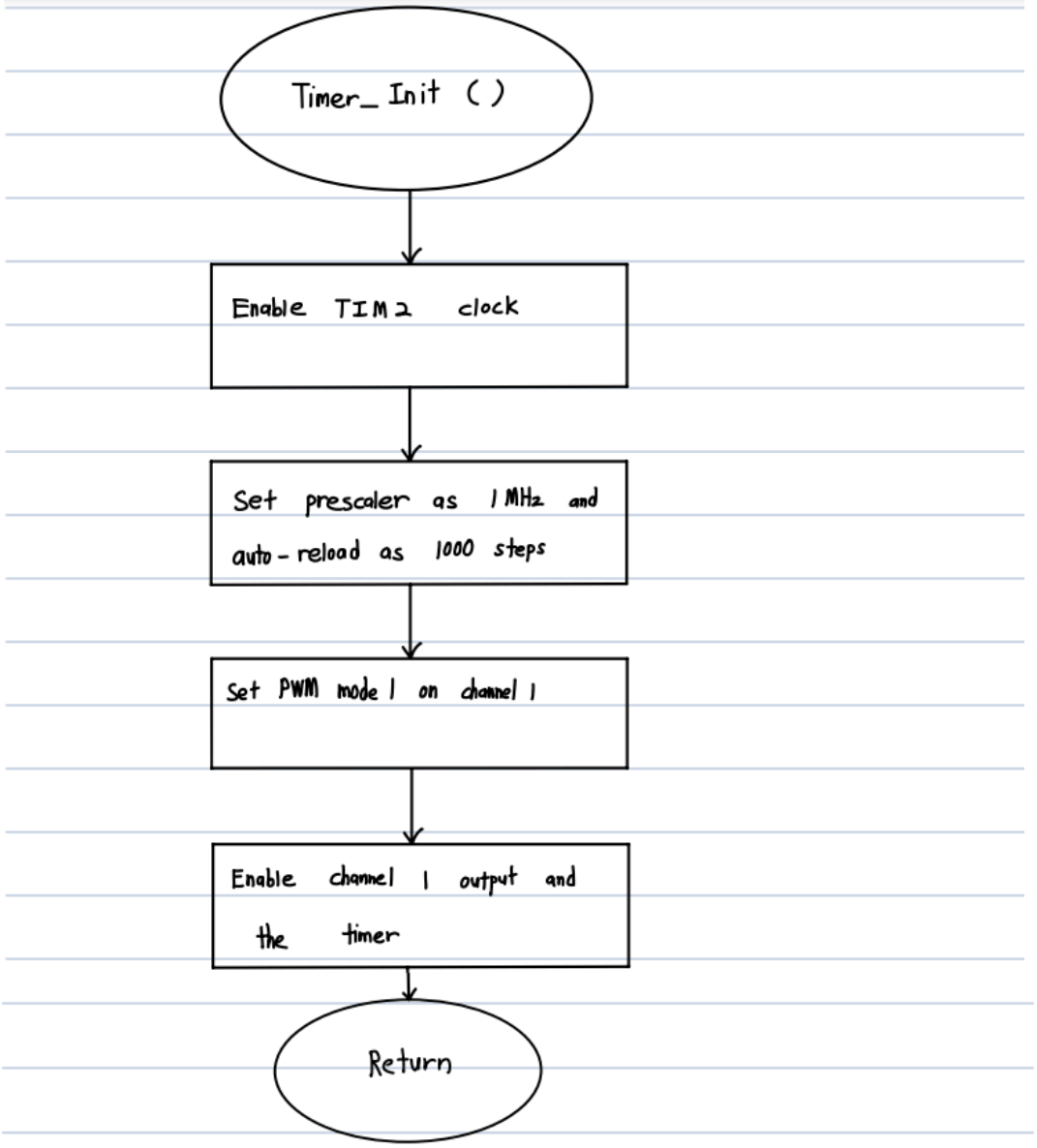
Components	Picture	Description
STM-32 Microcontroller		• Central control unit responsible for interfacing with the sensors and controlling the LED.
Ultrasonic Sensor (HC-SR04)		• Motion Sensor • Measures the distance to an object using sound waves. • Sends a pulse via a trigger pin and measures the time taken for the echo to return.
LDR (Light Dependent Resistor)		• Measures ambient light intensity using an ADC channel.
PWM-Controlled LED		• Brightness is controlled based on the LDR value.

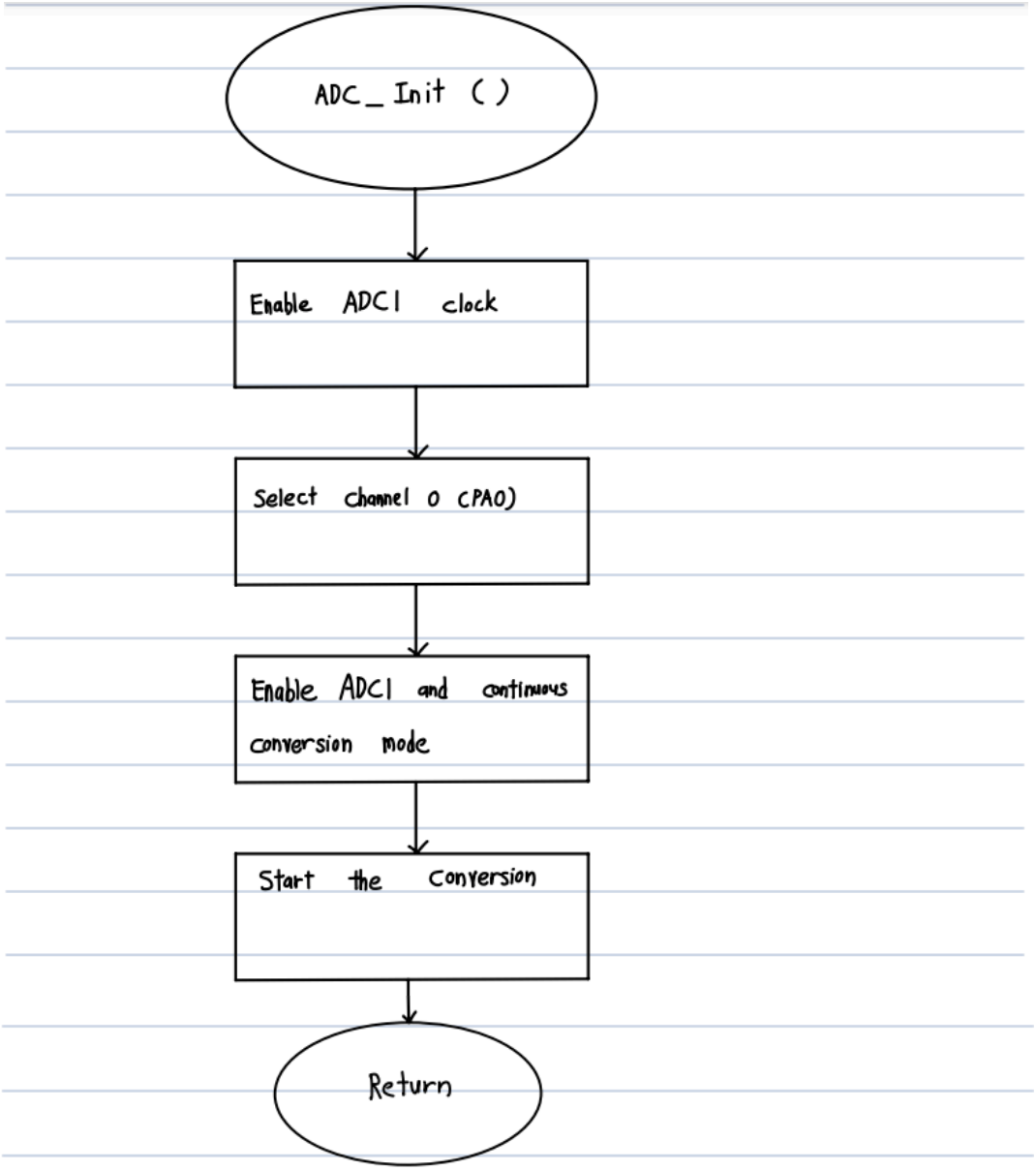
3.0 FLOWCHART

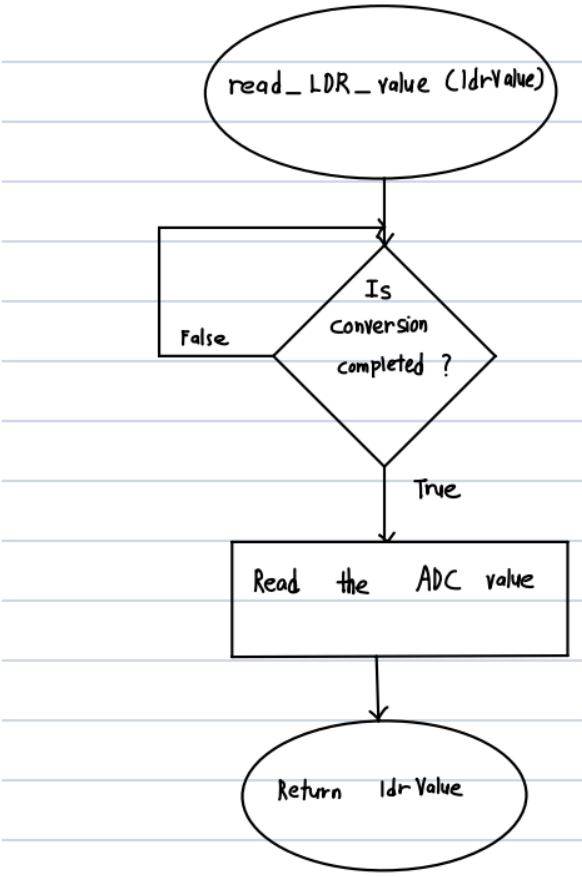


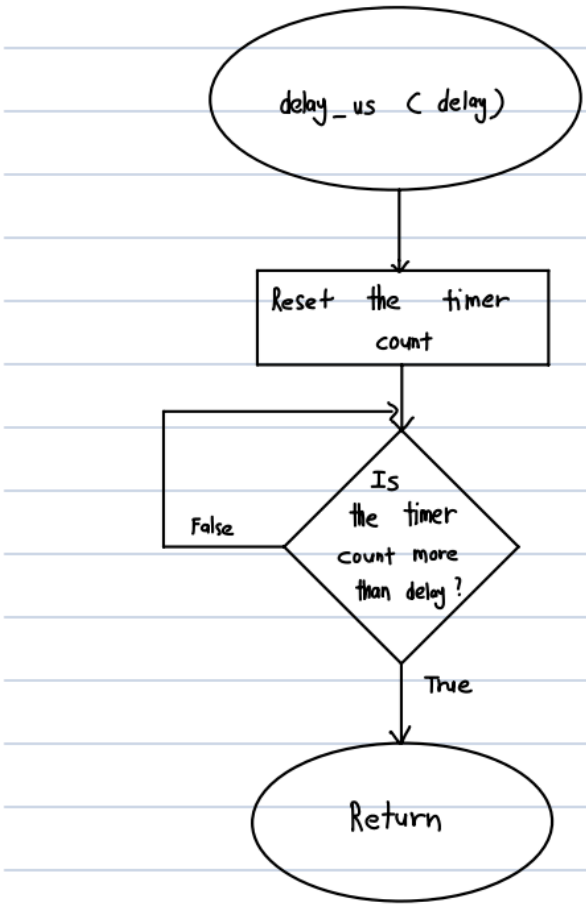

Is
0 < distance < Threshold distance
?
False
Set PWM duty cycle to 0 (turn off PWM and LED)
True
read_LDR_value (ldrvalue)
Is
ldrvalue ≤ 300
?
False
True
PwmValue = 100 − (ldrvalue x100) / 4095
Set PWM duty cycle by scaling to timer period by : (PwmValue x ARR value) / 100
LED pin , PWM duty cycle (brightness)
End

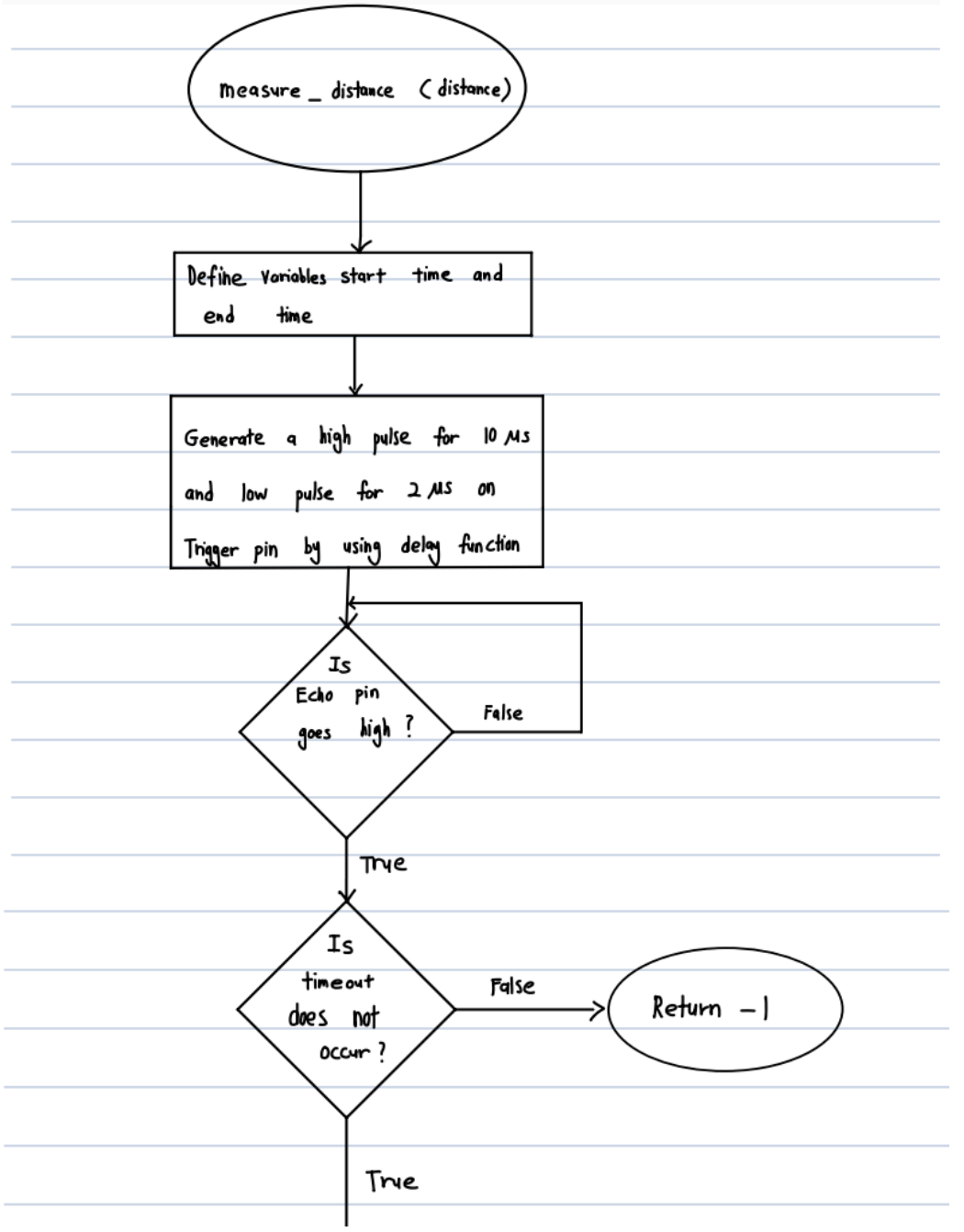

GPIO_Init ()
Enable GPIOA clock
Set PA8 (Trigger Pin) as high speed push-pull output
Set PA9 (Echo Pin) as input
Set PA5 (LED pin) as alternative function (AF1 : PWM output)
Set PA0 (LDR input) as analog input mode
Return

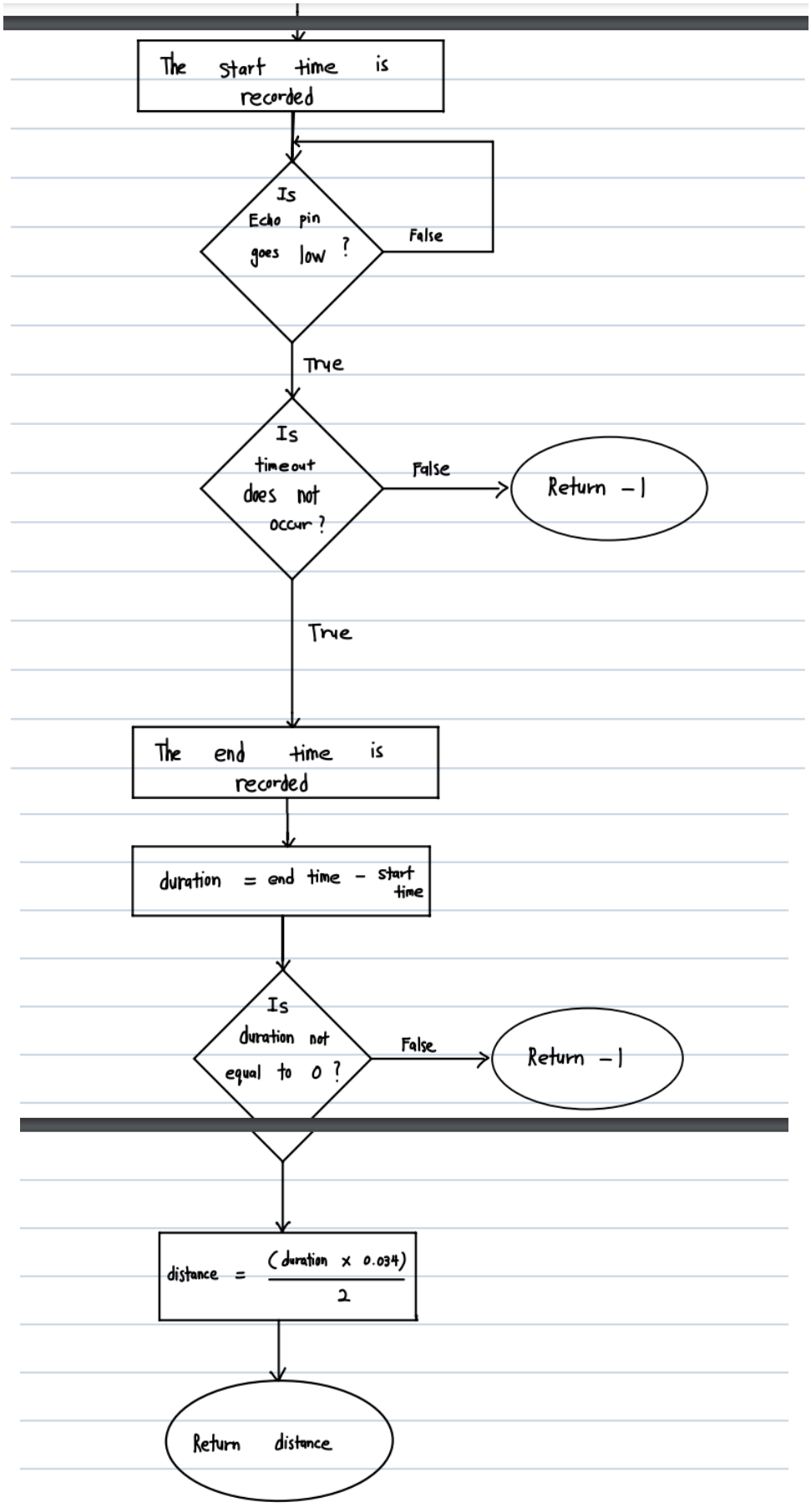

Timer_ Init ( )
Enable TIM2 clock
Set prescaler as 1 MHz and auto-reload as 1000 steps
Set PWM mode 1 on channel 1
Enable channel 1 output and the timer
Return


ADC_Init ( )
Enable ADC1 clock
Select channel 0 (PA0)
Enable ADC1 and continuous conversion mode
Start the conversion
Return


read_LDR_value (ldrValue)
Is conversion completed ?
False
True
Read the ADC value
Return ldrValue


delay_us (delay)
Reset the timer count
Is the timer count more than delay ?
False
True
Return


measure_distance (distance)
Define variables start time and end time
Generate a high pulse for 10 µs and low pulse for 2 µs on Trigger pin by using delay function
Is Echo pin goes high ?
False
True
Is timeout does not occur ?
False
Return -1
True


The start time is recorded
Is Echo pin goes low ?
False
True
Is timeout does not occur ?
False
Return -1
True
The end time is recorded
duration = end time - start time
Is duration not equal to 0 ?
False
Return -1
distance = (duration x 0.034) / 2
Return distance

4.0 CIRCUIT DIAGRAM

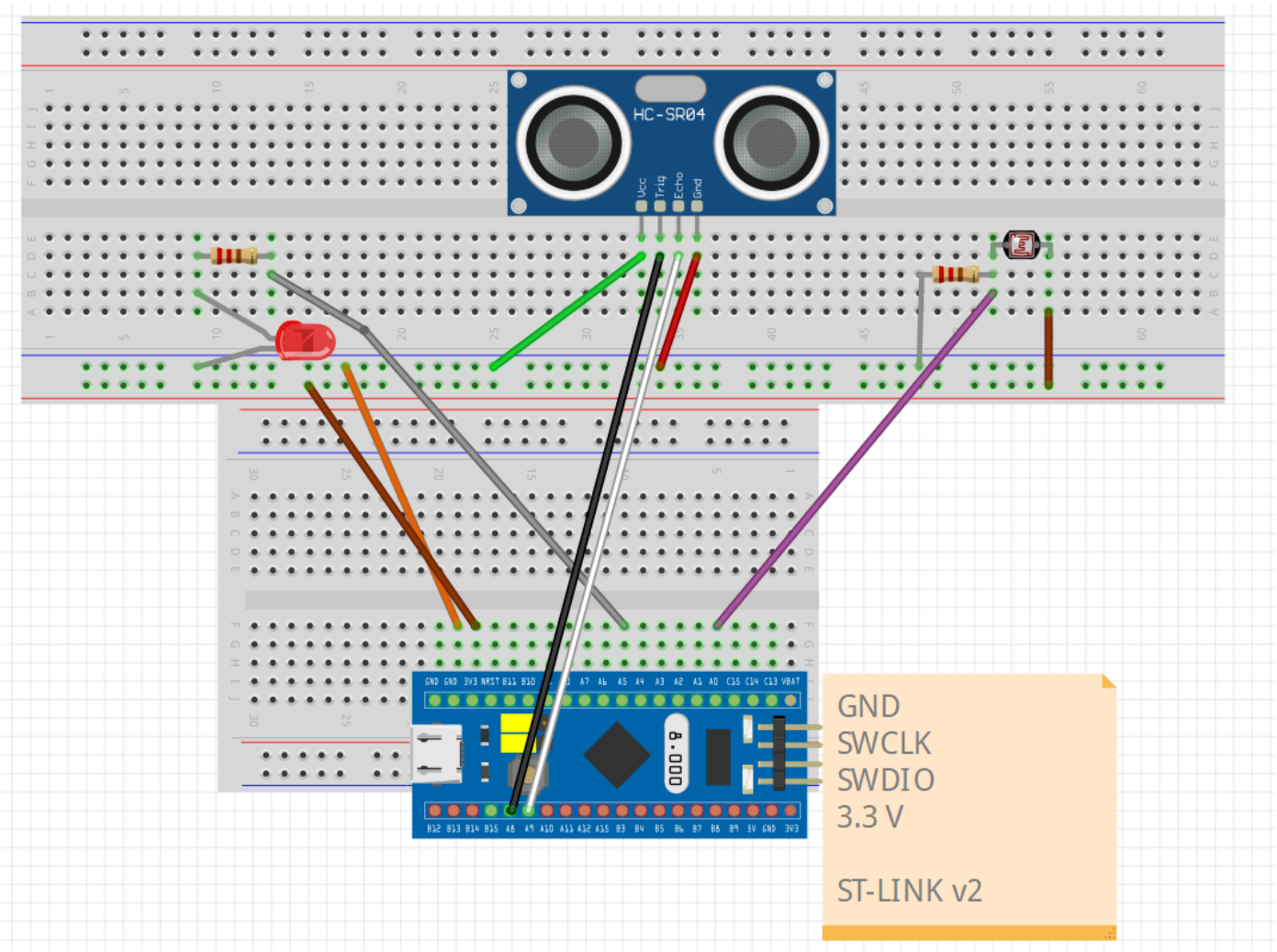


Figure 1: Circuit Diagram of the prototype by using Fritzing.

5.0 HARDWARE BUILT

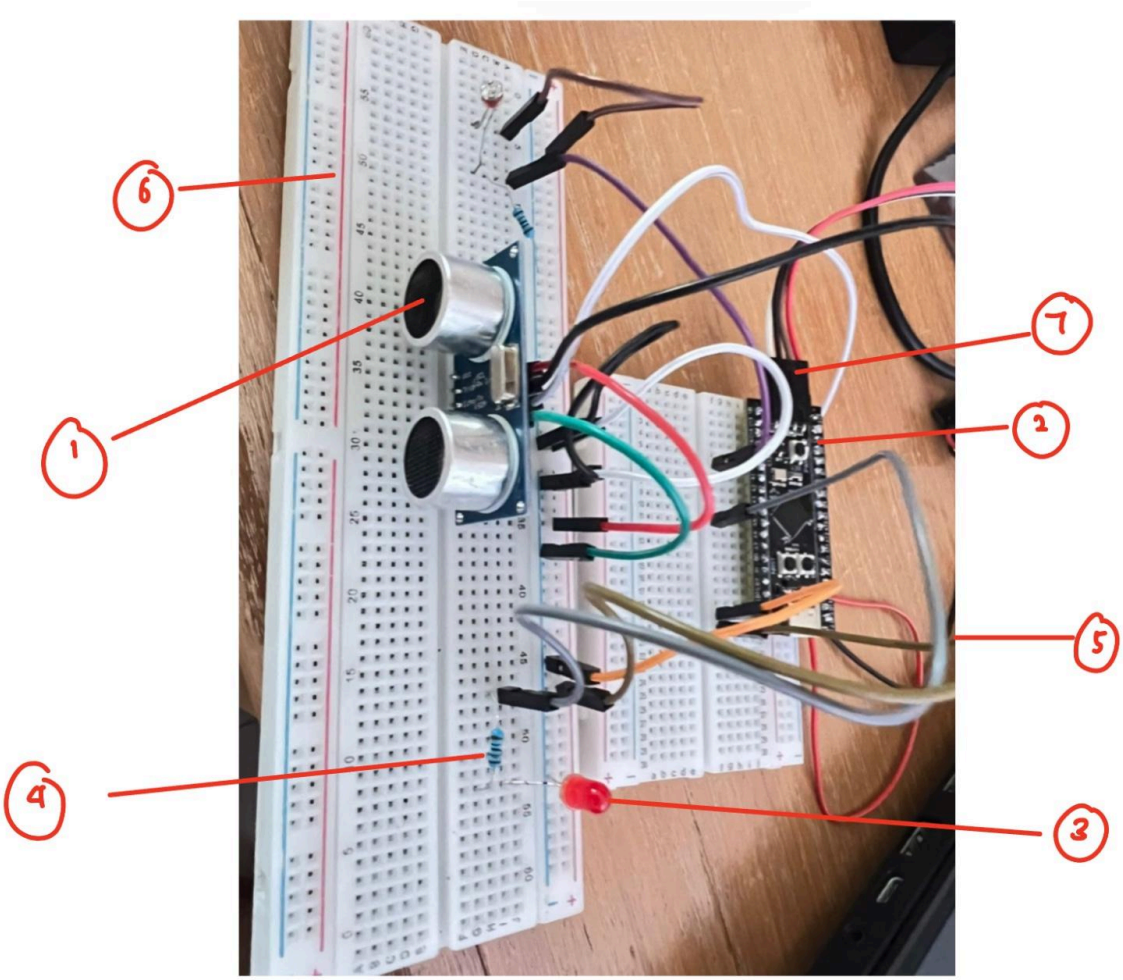


Figure 2: Hardware build with labelling

No.	Components	Function
1	Ultrasonic Distance Sensor	• Two cylindrical transducers: one transmitter, one receiver • Used to measure distances by sending and receiving ultrasonic waves
2	STM32 Development Board	• Microcontroller for controlling the circuit • Provides GPIO pins for input/output operations •

3	LED (Red)	● Used for visual output indication
4	Resistor	● Connected in series with the LED ● Limits the current to the LED to protect it
5	Jumper Wires	● Connecting components to power, ground, and GPIO pins
6	Breadboard	● For prototyping without soldering
7	Power Supply (via USB)	● Connected to the microcontroller for powering the circuit

6.0 DESIGN FUNCTIONALITY

```c
#include "stm32f4xx.h"

// Define pins
#define TRIG_PIN (1 << 8)   // PA8 - Trigger Pin
#define ECHO_PIN (1 << 9)   // PA9 - Echo Pin
#define LED_PIN (1 << 5) // PA5 (PWM for LED)
#define DISTANCE_THRESHOLD 5 // Distance threshold in cm
#define LDR_CHANNEL 0    // ADC Channel 0 (LDR connected to PA0)
#define TIMEOUT_LIMIT 10000   // Timeout limit for echo signal in cycles

// Function prototypes
void GPIO_Init(void);
void Timer_Init(void);
void ADC_Init(void);
uint16_t read_LDR_value(void);
float measure_distance(void);
void delay_us(uint32_t delay);

int main(void) {
    GPIO_Init();
    Timer_Init();
    ADC_Init();

    while (1) {
        // Measure the distance
        float distance = measure_distance();

        // Check if the distance is below the threshold
        if (distance > 0 && distance <= DISTANCE_THRESHOLD) {
            // Read LDR value
            uint16_t ldrValue = read_LDR_value();


                    if (ldrValue <= 300){
            // Map the LDR value (0-4095) to a PWM duty cycle (0-100%)
            uint16_t pwmValue = 100 - (ldrValue * 100) / 4095;
            // Set PWM duty cycle
            TIM2->CCR1 = (pwmValue * TIM2->ARR) / 100; // Scale to timer period
                    }
        } else {
            // Distance exceeds the threshold, turn off PWM and LED
            TIM2->CCR1 = 0;  // Set PWM duty cycle to 0
        }
    }
}

void GPIO_Init(void) {
    // Enable GPIOA clock
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;

    // Configure PA8 (TRIG_PIN) as output
    GPIOA->MODER |= (1 << 16);     // Set PA8 as output
    GPIOA->OTYPER &= ~TRIG_PIN;    // Set PA8 as push-pull
    GPIOA->OSPEEDR |= (3 << 16);   // Set PA8 as high speed

    // Configure PA9 (ECHO_PIN) as input
    GPIOA->MODER &= ~(3 << 18);    // Set PA9 as input

    // Configure PA5 (LED_PIN) as alternate function (PWM output)
    GPIOA->MODER |= GPIO_MODER_MODER5_1; // Alternate function mode
    GPIOA->AFR[0] |= (1 << 20);          // AF1 for TIM2_CH1

    // Configure PA0 (LDR input) as analog input
    GPIOA->MODER |= GPIO_MODER_MODER0;   // Analog mode
}
```

```

void Timer_Init(void) {
    // Enable TIM2 clock
    RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;

    // Configure TIM2 for PWM
    TIM2->PSC = 84 - 1;         // Prescaler: 1 MHz (1 µs per tick)
    TIM2->ARR = 1000 - 1;       // Auto-reload: 1000 steps (1 kHz PWM frequency)
    TIM2->CCMR1 |= (6 << 4);    // PWM mode 1 on channel 1
    TIM2->CCER |= TIM_CCER_CC1E; // Enable channel 1 output
    TIM2->CR1 |= TIM_CR1_CEN;   // Enable the timer
}

void ADC_Init(void) {
    // Enable ADC1 clock
    RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;

    // Configure ADC1 for single-channel continuous conversion
    ADC1->SQR3 = LDR_CHANNEL;  // Select channel 0 (PA0)
    ADC1->CR2 |= ADC_CR2_ADON; // Enable ADC1
    ADC1->CR2 |= ADC_CR2_CONT; // Continuous conversion mode
    ADC1->CR2 |= ADC_CR2_SWSTART; // Start conversion
}

uint16_t read_LDR_value(void) {
    while (!(ADC1->SR & ADC_SR_EOC)); // Wait for conversion to complete
    return ADC1->DR;                  // Return ADC value
}


// Delay function in microseconds
void delay_us(uint32_t delay) {
    TIM2->CNT = 0;                      // Reset the timer count
    while (TIM2->CNT < delay);          // Wait until the timer reaches the delay
}

// Function to measure distance using the ultrasonic sensor
float measure_distance(void) {
    uint32_t start_time, end_time;

    // Send a 10 µs pulse on the trigger pin
    GPIOA->ODR &= ~TRIG_PIN;    // Trigger low
    delay_us(2);
    GPIOA->ODR |= TRIG_PIN;     // Trigger high
    delay_us(10);
    GPIOA->ODR &= ~TRIG_PIN;    // Trigger low

    // Wait for echo pin to go high (start of the pulse)
    uint32_t timeout = TIMEOUT_LIMIT;
    while (!(GPIOA->IDR & ECHO_PIN) && timeout--) {
        // Timeout protection
    }
    if (timeout == 0) return -1;  // Timeout, return error code

    start_time = TIM2->CNT;   // Record the start time

    // Wait for echo pin to go low (end of the pulse)
    timeout = TIMEOUT_LIMIT;
    while (GPIOA->IDR & ECHO_PIN && timeout--) {
        // Timeout protection
    }
    if (timeout == 0) return -1;  // Timeout, return error code
```

```
126
127         end_time = TIM2->CNT;     // Record the end time
128
129         // Calculate the duration and convert to distance
130         uint32_t duration = end_time - start_time;
131         if (duration == 0) return -1;  // Avoid divide by zero
132
133         // Calculate the distance based on the duration
134         float distance = (duration * 0.034) / 2.0; // Speed of sound = 0.034 cm/µs
135
136         return distance;
137     }
```

Table 1: Description of C code line by line.

Line	Description
Initialization	
1	Include the STM32F4 peripheral library. The library provides access to register definitions specific to STM32F4 microcontrollers. It also includes macros to simplify hardware configuration and control.
4	Define the Trigger Pin at PA8 for the ultrasonic sensor.
5	Define the Echo Pin at PA9 for the ultrasonic sensor.
6	Define the LED Pin which indicate the Pulse Width Modulation frequency for LED dimming at PA5
7	Set a threshold for activating PWM based on distance which is 5 cm.
8	Specifie the ADC channel connected to the light-dependent resistor (LDR). The channel used is Channel 0.
9	Set a timeout limit for echo signal in cycles which is 10000 cycles to avoid the infinite loops during signal detection.
12-17	Function prototype declaration which consists of return type, function name and formal parameter list for each function that is used in the main program. These functions handle hardware initialization, sensor readings, and delays.

Main Program	
19	Function header for main program.
20-22	Initialize the GPIO pins, timer, and ADC peripherals for use by calling their respective function.
24-26	Measure the distance from the ultrasonic sensor *continuously* and assign it as variable named *"distance"* by calling a function that is defined for measuring distance (lines 101-137).
28-31	Check whether the distance is valid *and* below the threshold distance. If the condition is fulfilled, read the LDR value and assign it as a variable named *"ldrValue"* by calling the function that is defined for reading the LDR value (lines 89-92).
33-38	If the LDR value is less than or equal to 300 (the threshold that signifies a low-light condition which will trigger the response) : a) Map the LDR value to a PWM duty cycle by scaling down the value to a percentage ranging from 0% until 100% relative to its maximum possible value of 4095 (For 12-bit ADC resolution). b) Invert the mapping by using subtraction by 100, so that the lower LDR value results in higher PWM value. c) Convert the PWM value in percentage to an absolute value that corresponds to the required pulse width by multiplying with the timer's period. The calculated value is assigned to the CCR1 register (Capture/Compare Register 1) which determines the duty cycle for the timer channel.
39-42	Else statement when LDR value is more than 300 (threshold). The PWM and LED will be turned off by setting the PWM duty cycle to 0 in the CCR1 register.
GPIO Setup and Configuration	

46	Function definition header for GPIO initialization.
48	Enables the clock for the GPIOA peripheral by setting the GPIOAEN bit in the AHB1 peripheral clock enable register.
51-53	Configure PA8 (Trigger pin) as a *high-speed push-pull output* by: 1) Write 01 (output mode) to bits 16-17 in Mode Register. 2) Ensure the bit 8 in Output Type Register is Push-Pull mode (00). 3) Write 11 (high-speed mode) to bits 16-17 in the Speed Register.
56	Configure PA9 (Echo pin) as input by ensuring the bits 19-18 is 00 (input mode) in Mode Register.
59-60	Configure PA5 (LED pin) as alternative function which is used for PWM output: 1) Set PA5 to 10 (alternate function mode) at bits 11-10 in Mode Register. 2) Assign AF1 (TIM2_CH1) to PA5 by setting 0001 at bits 23-20 in the Alternative Function Low Register.
63	Configure PA0 (LDR input) as analog input by setting 11 (analog mode) at bits 1-0 in Mode register.
Timer Initialization and Configuration	
66	Function definition header for Timer initialization.
68	Enables the clock for the Timer 2 peripheral by setting the TIM2EN bit in the APB1ENR register of RCC (Reset and Clock Control).
71	Set a Prescaler which divides the timer's input clock frequency to reduce the effective timer frequency. System clock is assumed to run at 84 MHz and Prescaler is set to reduce the timer clock to 1 MHz (1 µs per tick).
72	Set a period of the timer in Auto-Reload Register. The timer overflows every 1000 µs (1 ms) with a 1 MHz clock and an ARR value of 999. It can shown by the

	formula which: $PWM\ frequency\ = \frac{Timer\ Clock}{ARR+1} = \frac{1MHz}{1000} = 1kHz$ (PWM period = 1 ms)
73	Capture/Compare Mode Register 1 is used to control the behavior of Timer 2's channels. Set PWM Mode 1 on channel 1 by writing 110 to bits 6-4 in CCMR1. In PWM Mode 1, the output is active when the counter is less than the compare value and inactive otherwise.
74	Enable the output on channel 1 of Timer 2. The CC1E bit is set in the Capture/Compare Enable Register (CCER) to connect the compare output to the corresponding GPIO pin.
75	Enable the timer by setting the CEN bit in Control Register 1 (CR1). Now the timer starts to count and generate PWM.
ADC Initialization and Configuration	
78	Function definition header for ADC initialization.
80	Enable the clock for ADC1 peripheral by setting the ADC1EN bit in the APB2ENR Register.
83	Select the channel 0 (LDR connected to PA0) to be converted in the ADC's regular conversion sequence by setting Regular Sequence Register 3 (SQR3).
84	Enable ADC1 by setting the ADON bit in the Control Register 2 (CR2).
85	Enable continuous conversion mode by setting CONT bit in Control Register 2 (CR2). In this mode, the ADC automatically restarts conversions on its own. It does not require a new trigger to begin the next conversion cycle.
86	Start the ADC conversion by setting SWSTART bit in Control Register 2 (CR2).
Program for reading the LDR value	

89	Function definition header for the program that is designed to read LDR value. The program will return an unsigned integer value which contains the LDR value to the main program.
90	Loops used to check the EOC (End of Conversion) flag in the Status Register (SR).The loop ensures the function waits until the ADC finishes converting the current sample. When conversion is finished, EOC will be set as 1 by hardware and the loops stop.
91	During the loops, the result of ADC conversion is holded by Data Register (DR). The function reads and returns this value to the main program. The value will range from 0 to 4095 for a 12-bit ADC.
Program for Time Delay in Microseconds	
95	Function definition header for the program that is designed for time delay in microseconds with an input unsigned integer argument which indicates the time needed to delay.
96	Counter Register (CNT) of TIM2 is reset to 0 to ensure the timer starts counting from zero for the delay.
97	The loops continue and will stop when it meets the condition that the value in Counter Register is exceeding the desired delay value in microseconds. This performs the time delay operation.
Program for measuring distance using ultrasonic sensor	
101-102	Function definition header for the program that is designed to measure the distance using an ultrasonic sensor and return a value (distance measured) to the main program in float type number. Two variables named *"start_time"* and *"end_time"* are defined in unsigned integer data types.

105-109	Generate a high pulse for 10 µs on the Trigger Pin (PA8) by setting 1 at the bit 8 in the Output Data Register (ODR) with a delay function of argument = 10. Before and after the high pulse, there is a low pulse for 2 µs on the Trigger pin for stabilization purposes by setting 0 at the bit 8 in the ODR with a delay function of argument = 2. The delay function (delay_us) ensures precise timing.
112-113	Assign a variable named *"timeout"* with a value of *"TIMEOUT_LIMIT"* which is 10000. The loop is used to check if the Echo Pin (PA9) is 1 (High) at bit 9 in Input Data Register (IDR). A timeout counter is used in order to prevent the function from hanging indefinitely if no echo signal is received.
116	If timeout occurs (timeout counter reaches 0), the function exits early with an error code (-1).
118	The current timer value from the Counter Register (CNT) of Timer 2 is recorded as the start time once the echo signal is detected (Echo Pin in IDR is High).
121-125	The operation is similar to lines 112-116 but now the loop is used to check if the Echo Pin (PA9) is 0 (Low) at bit 9 in Input Data Register (IDR). A timeout counter now is used to prevent the function from hanging if the signal doesn't end. If timeout occurs (timeout counter reaches 0), the function exits early with an error code (-1).
127	The current timer value from the Counter Register (CNT) of Timer 2 is recorded as the end time once the echo signal is ended (Echo Pin in IDR is Low).
130-131	The time difference is calculated by subtracting the end time with the start time. The time difference represents the duration of the echo signal in microseconds. An error code (-1) is returned if the duration is 0 in order to avoid divide-by-zero errors in the distance calculation.
134	The distance is calculated by applying the formula:

	$Distance = \frac{Duration * 0.034}{2}$ where 0.034 is the speed of sound in unit cm/μs with assumption standard air temperature. And divided by 2 to account for the round trip.
136	Return the calculated distance in unit cm to the main program.

7.0 SIMULATION

Case 1: Object Distance = 3 cm, LDR Value = 200

Condition:

```
C

if (distance > 0) && distance <= DISTANCE_THRESHOLD) //true condition
if (ldrValue <=300) // true condition
```

Result:

When the object distance is greater than 0 cm and is less than equal to 5 cm. Besides, the ldrValue is less than equal to 300.The LED brightness is controlled by PWM based on ldrValue.

Case 2: Distance = 4cm, LDR Value = 350

Condition:

```
C

if (distance > 0) && distance <= DISTANCE_THRESHOLD) //true condition
if (ldrValue <=300) // false condition
```

Result:

When the object distance is greater than 0 and is less than equal to 5 cm. Besides, the ldrValue is greater than 350. No PWM control and the LED is unaffected. PWM is not adjusted and in default state.

Case 3: Distance = 7cm, LDR Value= 150

Condition:

```
C

if (distance > 0) && distance <= DISTANCE_THRESHOLD) //false condition
}
else {
        TIM2->CCR1 = 0; // Set PWM duty cycle to 0
}
```

Result:

When the object distance is greater than 5 cm. Therefore, the program enters else condition. The LED will turn off. The duty cycle is explicitly set to 0%, so no PWM signal is generated. Since the distance check is false, the code never reads or considers the LDR value (ldrValue = 150 in this case becomes irrelevant).

Case 4: Distance = -1cm (error condition)

Condition:

```
C

if (timeout == 0) return -1; //Timeout error
if (duration == 0) return -1; // Sensor error
```

Result:

When distance is negative value (error condition), the LED will be turned off , as distance = -1cm does not satisfy the if condition.

8.0 RESULT AND DISCUSSION

The design of the Smart Fluorescent Lamp system integrates STM32 Microcontroller, ultrasonic sensor, LDR and PWM-controlled LED is to enhance the energy efficiency and user convenience. The system dynamically adjusts the brightness of the lamp based on the motion detection and the ambient light intensity. According to the results as shown above, this system successfully demonstrates dynamic brightness control using motion detection and ambient light sensing. The energy usage is effectively optimized by the system by adjusting the LED brightness based on the real-time environmental conditions.

The system was tested under different conditions, and the results shown are aligned with the expected functionality. Each test provides valuable insights into the efficiency and effectiveness of the Smart Fluorescent Lamp system. In Case 1, the microcontroller successfully detects the motion and the low light intensity, which then triggers the LED. The PWM signal also dynamically controls the LED brightness depending on the LDR reading. This confirms that the mapping function works properly. While in Case 2, the results showed that the system prevents the unnecessary energy consumption when the ambient lighting is already sufficient. The LDR threshold is appropriately set to differentiate between the dark and bright environments, while the microcontroller processes multiple sensor inputs accurately before execution. In Case 3, it shows that the system prioritizes motion detection first, even though there is a dark environment detected by the LDR. The LED will only turn on if there is an object existing within the range. This proves that the distance threshold is functioning correctly and the LED control logic is well-implemented, which ensures that the system does not waste energy by unnecessarily activating the LED. Lastly in Case 4, the error-handling mechanism prevents faulty sensor readings from affecting the system operation. The implementation of timeout checks ensures that the program does not get stuck in an infinite loop due to sensor failures and the system's stability is improved through well-defined conditions for handling erroneous input.

Generally, according to the system functionality and performance, we can conclude that this Smart Fluorescent Lamp system was designed correctly. From the aspects of the implementation of microcontroller, the STM32 Microcontroller is effectively and well-programmed in order to

handle sensor data processing and LED control. Its ADC and timer peripherals ensure the precise sensor readings and responsive PWM control for the brightness of the LED.While from the aspects of the sensor efficiency and responses, we can observe that the ultrasonic sensors are able to detect the motion within the predefined range accurately. The LDR continuously monitors ambient light, enabling real-time brightness adjustments. Lastly, for the implementation of the PWM-based LED control, the brightness of the LED is able to be adjusted dynamically. The system successfully and effectively prevents unnecessary energy consumption in well-fit environments.

9.0 CONCLUSION

The Smart Fluorescent Lamp System, STM-32 based project integrates multiple peripherals to create a functional distance and light-sensitive LED control system. The ultrasonic sensor measures distance using pulse-echo techniques, while the LDR (Light Dependent Resistor) provides analog light intensity readings. The system dynamically adjusts the LED brightness via PWM based on the ambient lighting conditions when an object is detected within a defined threshold distance.

Key functions such as measure_distance ( ) , read_LDR_value ( ) , and delay_us ( ) are implemented to ensure precise timing and data acquisition. The Timer_Init ( ) function configures TIM2 for both PWM generation and microsecond delay functionality, essential for accurate ultrasonic distance measurements. Additionally, the ADC_Init ( ) function enables continuous analog-to-digital conversion to monitor the LDR values.

By mapping the LDR's 12-bit ADC output to a corresponding PWM duty cycle, the system efficiently controls LED brightness to respond to changing light conditions. This feature demonstrates a practical application for energy-efficient smart lighting systems**.**

This project serves as an excellent starting point for embedded system enthusiasts seeking hands-on experience with STM32 microcontrollers, ADC operations, PWM generation, and sensor integration. Potential improvements include enhanced signal processing techniques for noise filtering, integration of low-power modes, and communication interfaces for remote monitoring and control.

10.0 REFERENCES

1) Dejan. (n.d.). Ultrasonic Sensor HC-SR04 and Arduino (Complete Guide) https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/

2) STMicroelectronics. (n.d.). Getting started with STM32: STM32 step-by-step. Retrieved from https://wiki.st.com/stm32mcu/wiki/STM32StepByStep:Getting_started_with_STM32_:_STM32_step_by_step

3) Instructables. (n.d.). *SmartLamp Project*. Retrieved from https://www.instructables.com/SmartLamp-Project/

4) Smart Solutions for Home. (n.d.). *How to program STM32 – The Complete Guide*. Retrieved from https://smartsolutions4home.com/how-to-program-stm32/

5) DeepBlue Embedded. (n.d.). *STM32 Tutorials. ARM Programming – STM32 Course*. Retrieved from https://deepbluembedded.com/stm32-arm-programming-tutorials/