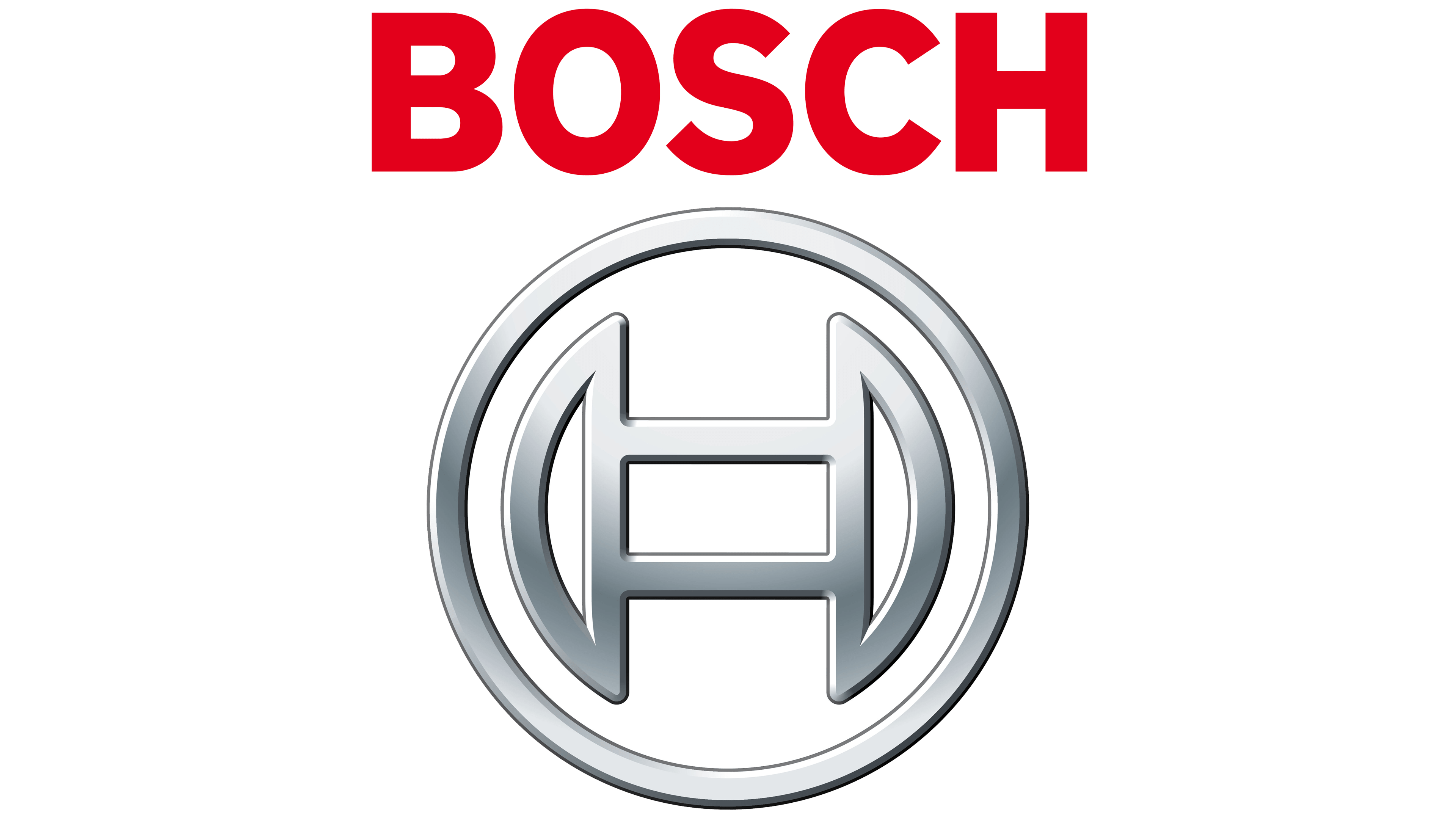
REMOTE CONTROLLERD HOME AUTOMATION



PROJECT REPORT

submitted by

Anchal Dev

Vishawjeet Singh

Surya Prakash Singh

Ruban Mariappan

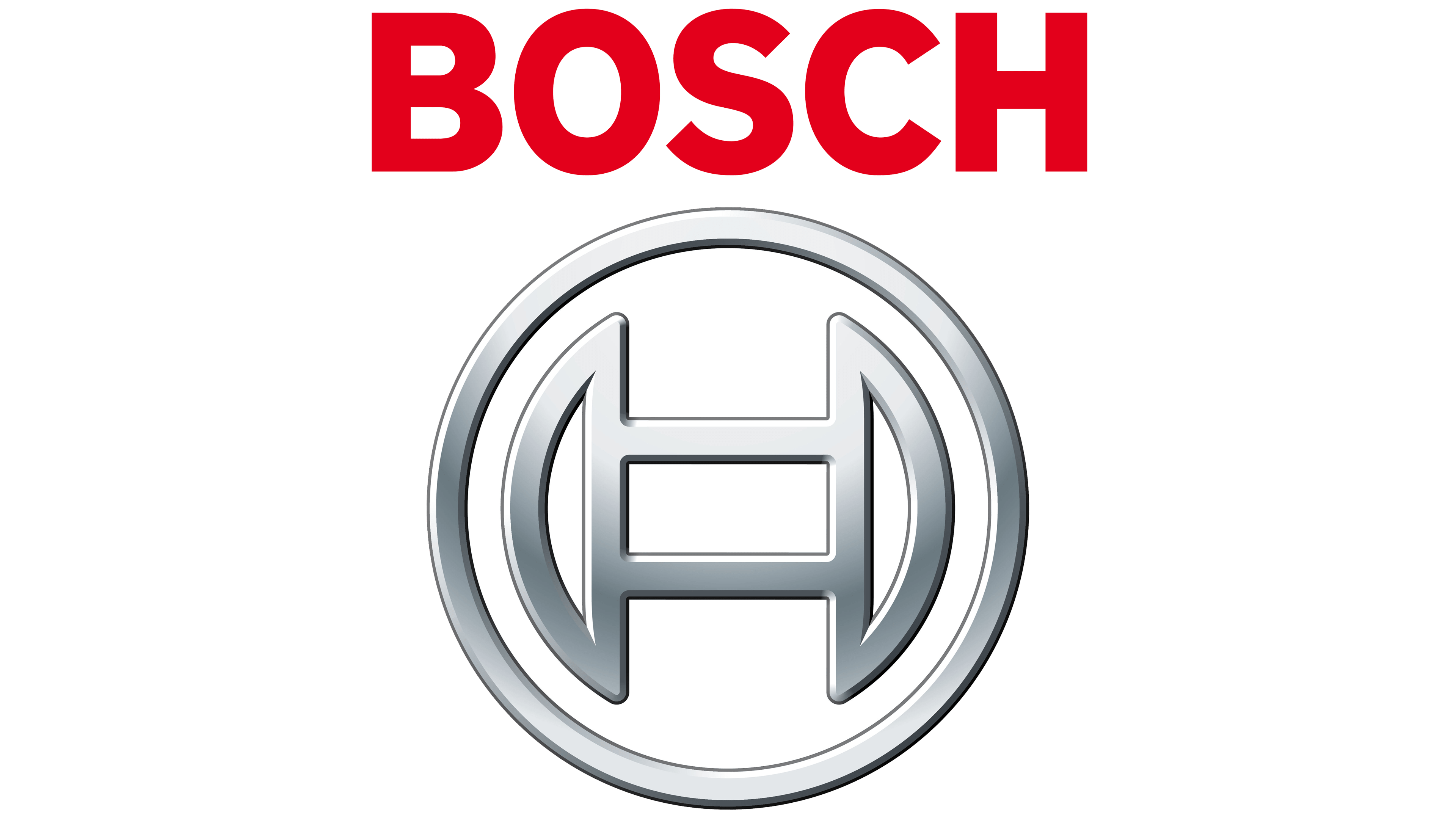
BOSCH GLOBAL SOFTWARE TECHNOLOGIES

Pune

SEPT. 2025

BOSCH GLOBAL SOFTWARE TECHNOLOGIES

Pune



Certificate

This is to certify that the project entitled **“Remote Controlled Home Automation”** using **STM32F405** has been successfully carried out by **TEAM 4** under the supervision of **Kishore Bhagat**, at Bosch Global Software Technologies, Pune. The project was conducted as part of the corporate apprentice program and demonstrated the application of embedded system design and real-time control of home appliances.

The STM32F405 microcontroller was used in the design, development, and implementation of a remote-controlled home automation system incorporating various sensors and actuators such as temperature sensors, relays, and communication modules. The system enables efficient monitoring and control of home devices like lights, fans, and other appliances remotely, enhancing convenience and energy efficiency.

This report presents original work and has not been submitted for any other purpose

BGSW,Pune

Acknowledgement

We would like to take this opportunity to sincerely thank Mr. Kishore Bhagat for his invaluable mentorship, guidance, and encouragement throughout the duration of our project. His constant support, technical expertise, and insightful feedback played a critical role in helping us understand complex concepts and successfully navigate the challenges we encountered during each stage of development. We are also profoundly grateful to the technical team at BGSW for their continuous support and contribution. The provision of essential resources, development boards, and the technical knowledge shared during training sessions were instrumental in shaping our understanding and enhancing the overall quality of our project. Their professionalism and responsiveness helped us stay focused and productive. Finally, we express our deep appreciation to our heartfelt thanks to our peers and colleagues for their teamwork, cooperation, and constructive inputs, which enriched our learning experience and helped us achieve our objectives more efficiently.

Contents

List of Figures

List of Tables

Abstract

1 Introduction 10

1.1 Objectives 11

2 System Overview

2.1 System Description 12

2.1.1 Sensor Module 12

2.1.2 Microcontroller Module 13

2.1.3 User Interface 14

2.1.4 Actuation Module 14

3 Hardware Requirements

3.1 STM32F405GT6 15

3.1.1 Core and Performance 15

3.1.2 Memory 16

3.1.3 Timers 16

3.1.4 GPIO pins 17

3.1.5 ADC 17

3.1.6 I2C 17

3.2 IR Remote control without battery 18

3.3 Infrared IR Sensor Receiver 19

3.4 4x4 Matrix 20

3.5 Breadboard and wires 21

3.5.1 Breadboard 21

3.5.2 Jumper wires 22

4. Software Tools

4.1 STMCUBEIDE 24

4.2 CMSIS 25

5. System Design and Flowcharts

5.1 Theory of working 26

5.2 Flowchart

5.2.1 Flowchart 28

5.2.2 Use case chart 29

5.2.3 Data flow chart 30

5.3 Final Sequence 31

6. Software Implementation

6.1 GPIO config 32

6.2 Software ckt. Diagram 33

7. Hardware Implementation

7.1 hardware setup 37

8. Source code

8.1 main.c 39

8.2 mpu.c 58

8.3 ultrasonic 64

8.4 DHT11.c 68

8.5 Soil moisture 73

8.5 lcd.c 75

9. Conclusion 78

List of Figures

Figure 3.1 14

Figure 3.2 17

Figure 3.3 18

Figure 3.4 19

Figure 3.5.1 20

Figure 3.5.2 21

Figure 3.6 22

Figure 5.1 27

Figure 5.2 28

Figure 5.3 29

Figure 5.4 31

Figure 6.2 32

Figure 7.1 34

List of tables

Table 6.1.1 32

Table 6.1.2 32

Table 6.1.3 32

Table 6.1.4 32

Table 6.1.5 33

ABSTRACT

The increasing demand for convenience, energy efficiency, and security in modern homes has driven the development of advanced Remote Control Home Automation Systems. This project focuses on designing and implementing a comprehensive system that allows remote monitoring and control of various home appliances and environmental parameters. The STM32 microcontroller acts as the central controller, managing all operations with the help of multiple sensors and actuators. A 16x2 LCD is used to display real-time status and system information. The DHT11 sensor monitors temperature and humidity levels to ensure a comfortable indoor environment. Relays are used to control devices such as lights and fans remotely. An IR sensor receiver detects commands from a remote control, enabling users to switch appliances on or off conveniently. This integration of sensors and remote control functionality makes the system efficient, user-friendly, and enhances home automation capabilities.

Chapter 1

Introduction

1.1 Background

In today’s rapidly evolving smart home environment, ensuring convenience, energy efficiency, and safety has become more important than ever. Traditional home management methods, which rely on manual operation and monitoring, often lack the ability to provide real-time control and feedback, limiting responsiveness and efficiency. This gap has led to the development of Remote Control Home Automation Systems, which offer a proactive and convenient approach by continuously monitoring and controlling various home appliances and environmental parameters. This system uses a DHT11 sensor for temperature and humidity sensing, which operates on the principle of capacitance measurement for humidity and thermistor for temperature sensing, ensuring comfortable indoor climate control. Relays are used to control home appliances like lights and fans remotely. An IR sensor receiver detects signals from a remote control, enabling users to conveniently switch appliances ON or OFF without physical interaction. Additionally, motion sensors can be integrated to enhance security and automate lighting. This integrated system enables efficient energy management, user comfort, and improved home safety through continuous monitoring and remote operation.

1.2 Objectives

As mentioned above, it’s a control and monitoring device equipped with all necessary features for home automation. This system especially covers –

1. Remote ON/OFF control of bulbs – using an IR receiver and remote control
2. Remote ON/OFF control of fans – with speed control to increase or decrease fan speed
3. Curtain control – remotely open and close curtains using the IR remote
4. Energy efficiency – by controlling appliances remotely to reduce unnecessary power usage
5. Convenience and safety – allowing users to operate multiple home devices without physical interaction

Chapter 2

System Overview

2.1 System Description

An embedded system called Vehicle Health Monitoring Systems is made to continually track important health metrics like Engine temperature of the vehicle, Humidity inside the cabin of vehicle, Fuel level of vehicle, Tilt in the axis of axle, Suspension health check as well as detection of obstacle while parking to avoid accidents with any wall or human in real time. An STM32F405 microcontroller (ARM Cortex-M4) and sophisticated sensors are combined to deliver precise, responsive, and easy-to-use vehicle health tracking.

The system consists of three main modules:

The **IR receiver module** receives signals from the IR remote control. It decodes the remote commands such as turning the bulb and fan ON/OFF, adjusting fan speed, and controlling curtain movement.

The **IR remote control** acts as the user interface, allowing the user to send commands wirelessly to control various home appliances conveniently.

The **servo motor** is used for precise control of mechanical components such as opening and closing curtains smoothly. The microcontroller drives the servo motor based on commands received via the IR receiver.

These modules together enable efficient and remote control of home devices, improving convenience and automation without the need for manual switching.

The STM32F405 microcontroller acts as the brain of the system. Incoming sensor data is processed, findings are shown on many output modules, and alert systems like buzzers and LED indicators are managed

2.1.1 Sensor Module

Home automation control is achieved using three key modules:

1. **IR Receiver Module**: Receives signals from the IR remote control to interpret user commands for controlling appliances.
2. **IR Remote Control**: Acts as the wireless interface for the user to send ON/OFF commands, adjust fan speed, and control curtain movement.
3. **Servo Motor**: Provides precise mechanical control for operations like opening and closing curtains smoothly based on commands received.

These modules provide digital input and control signals to the microcontroller, enabling dynamic and remote management of home appliances for improved convenience and automation.

2.1.2 Microcontroller Module

The STM32F405 microcontroller serves as the central control unit. It reads sensor inputs using its ADC and digital I/O channels, processes logic for threshold-based decision-making, and controls outputs such as LEDs, the LCD. It also manages the keypad scanning.

2.1.3 User Interface Module

The user interface includes:

• A 16x2 LCD for displaying temperature, humidity and fuel level.

• A 4x4 keypad for user input (if required).

2.1.4 Actuation Module

The Actuation Module responds directly to commands received via the IR remote control:

* The IR sensor receives signals from the IR remote, which are processed by the microcontroller to control electrical appliances like bulbs and fans, turning them ON or OFF according to user commands.
* Servo motors control mechanical devices such as curtains, enabling smooth opening and closing movements.

The microcontroller processes the signals decoded by the IR sensor and immediately triggers the appropriate actuators. For example, when the remote command to increase fan speed is received, the microcontroller adjusts the fan accordingly. This real-time control ensures convenient and efficient home automation.

Chapter 3

Hardware Component

3.1 **STM32F405RGT6**

3.1.1 Core and Performance

An ARM Cortex-M4 CPU with an integrated Floating-Point Unit (FPU) power the STM32F405RGT6, which can operate at up to 168 MHz It offers about 210 DMIPS and is intended for high-performance real-time applications. The core ensures efficiency and power by supporting the ARMv7-M instruction set with Thumb-2 technology. It is appropriate for a range of embedded system designs because it runs on a supply voltage range of 1.8V to 3.6V.

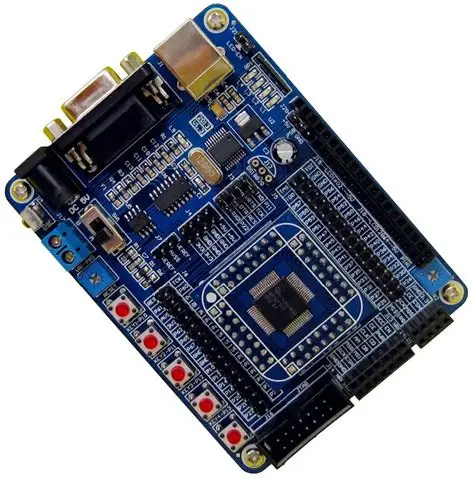


Fig.1 STM microcontroller

3.1.2 Memory

This microcontroller has 192 KB of SRAM, which is split into 64 KB of tightly coupled CCM (Core Coupled Memory) SRAM, 112 KB of SRAM1, and 16 KB of SRAM2. It also has 1 MB of Flash memory for program storage. Flash memory can be utilized to simulate EEPROM capability, when necessary, even if it lacks genuine EEPROM.

3.1.3 Timers

Numerous timers that are appropriate for a variety of timing and control purposes are included with the STM32F405RGT6. It has a four-channel PWM advanced-control timer (TIM1) that is perfect for motor control and supports complementary outputs and dead time insertion. 32-bit counters with up to four channels each are supported by general-purpose timers TIM2 through TIM5. Additionally, there are 16-bit timers TIM9, TIM10, and TIM11 for simpler PWM or input capture requirements, as well as the basic timers TIM6 and TIM7 for timing bases or DAC triggering. System-level timekeeping and RTOS support are provided by a specialized SysTick timer.

3.1.4 GPIO Pins

GPIO ports A through I are supported by the STM32F405RGT6, albeit not all of them are available in the 64-pin package. There are roughly 51 GPIOs in this package. Several modes, such as input, output, alternate function, and analog, can be set for each pin. Additionally, they offer output configurations like push-pull or open-drain, as well as internal pull-up/pull-down resistors.

3.1.5 ADC and DAC

Each of the three 12-bit ADCs (ADC1, ADC2, and ADC3) can function alone or in tandem. These ADCs support multiple conversion modes, including single, continuous, scan, and discontinuous, and they can share up to 24 channels. Software or timers can initiate conversions, and sampling duration can be adjusted for the best balance between speed and accuracy. Two 12-bit DAC channels on the chip can produce analog voltages. These can be applied to analog control, audio applications, or signal production. For accurate output timing, DACs can be activated by timers or software.

3.1.6 I2C

I²C (Inter-Integrated Circuit) is a serial communication protocol used to connect low-speed peripherals to microcontrollers. It was developed by Philips (now NXP) and is widely used in embedded systems due to its simplicity and efficiency.

Key Features of I²C:

Two-wire interface:

SDA (Serial Data Line) – for data transfer

SCL (Serial Clock Line) – for synchronization

Master-slave architecture:

One master control the bus (usually a microcontroller).

Multiple slaves (sensors, displays, etc.) can be connected.

3.2 **IR REMOTE CONTROL WITHOUT BATTERY**

The IR remote control is a commonly used wireless device designed to send commands to electronic appliances using infrared light signals. It works by emitting infrared light pulses encoded with specific digital data representing various commands such as ON, OFF, increase speed, or open/close. The remote itself typically contains an IR LED that transmits these signals. When operated without a battery, the remote relies on external power sources or energy harvesting methods to emit the IR signals, though most traditional remotes require a battery to function.

The IR receiver module, often paired with microcontrollers, detects the incoming IR signals using a photodiode or phototransistor that is sensitive to infrared light. The receiver then demodulates and decodes these signals into digital data which the microcontroller can process to execute the corresponding command.

This IR communication uses a simple, line-of-sight method with a limited range, making it suitable for controlling devices within a room. While IR remotes offer a straightforward and cost-effective way to control appliances wirelessly, their dependence on unobstructed paths and the need for a power source (battery) are some practical limitations.



Figure 3.2 IR Remote Control without Battery

3.3 **INFRARED IR SENSOR RECIEVER**

The MPU-6050 is a widely used inertial measurement unit (IMU) sensor developed by InvenSense. It combines a 3-axis gyroscope and a 3-axis accelerometer on a single chip, making it ideal for motion tracking and orientation sensing in embedded systems and robotics.

This sensor communicates via the I2C protocol, allowing easy integration with microcontrollers like Arduino, Raspberry Pi, and STM32. It also features an onboard Digital Motion Processor (DMP) that can process complex motion algorithms, reducing the computational load on the host processor.

Main features of MPU 6050:

1. 3-axis accelerometer for detecting linear acceleration.
2. 3-axis gyroscope for measuring angular velocity.
3. 16-bit ADCs for high-resolution measurements.
4. Operating voltage: 3.3V to 5V.
5. Compact size and low power consumption.

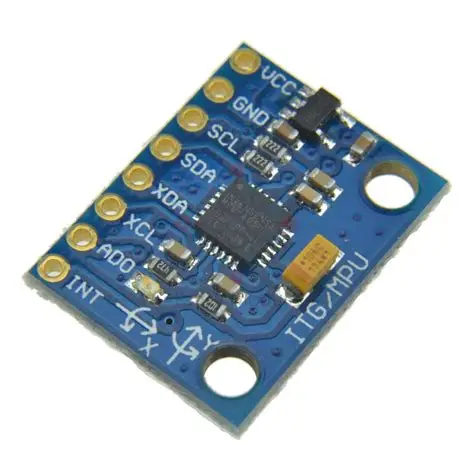


Figure 3.3 MPU 6050

3.4 **16X2 LCD**

The 16x2 LCD is a commonly used alphanumeric display module that can show 16 characters per line across 2 lines, making it ideal for simple text-based output in embedded systems and microcontroller projects. It can operate both 4-bit and 8-bit data modes, allowing flexible interfacing with devices like Arduino, Raspberry Pi, and other microcontrollers. Each character is displayed using a 5x8 pixel matrix, and the module typically includes a backlight for visibility in low-light conditions. It can display letters, numbers, and some custom characters, making it useful for showing sensor readings, system status, or user instructions in real-time applications.

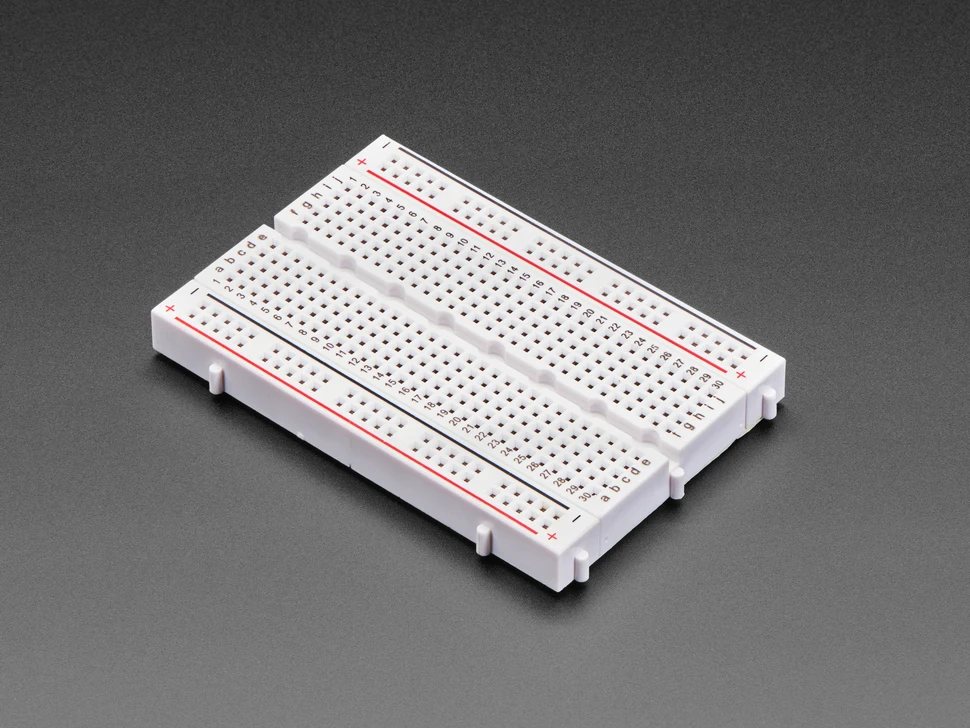


Figure 3.4 16x2 LCD

**3.5 Breadboards and Jumper wires**

3.5.1 Breadboard

A breadboard is a solderless prototyping tool that makes it easy and quick to construct and test electronic circuits. It is made up of a grid of spring-loaded holes that can accommodate component leads and standard 22 AWG solid-core jumper wires. The breadboard’s interior is made up of rows and columns of conductive strips. The vertical rails on the sides, designated + and -, are used to distribute power, while the horizontal rows in the center, usually arranged in sets of five, are electrically connected. Full-size breadboards have 830 tie points. Breadboard and jumper wires whereas half-size breadboards have 400 tie points. When temporary circuits are built before they are completed on a PCB, they are perfect. Although they cannot handle high-current or high-frequency circuits, breadboards are ideal for digital logic applications, sensor testing, and microcontroller interface.

Figure 3.5.1 Breadboard

3.5.2 Jumper Wires

Jumper wires are crucial parts for creating short-term, solderless connections between modules and on breadboards. They facilitate simple connections of microcontroller headers, sensors, modules, and breadboards and are available in three primary varieties: male-to-male, male-to-female, and female-to-female. The connections are made to fit snuggly into conventional breadboard sockets or headers with a pitch of 2.54 mm (0.1 inch), and the wires are usually 20 to 30 cm long. Although the colors do not have any intrinsic electrical value, jumper wires are color-coded for circuit structure and clarity. They are perfect for circuit debugging, prototyping, and instructional setups since they are flexible and reusable.

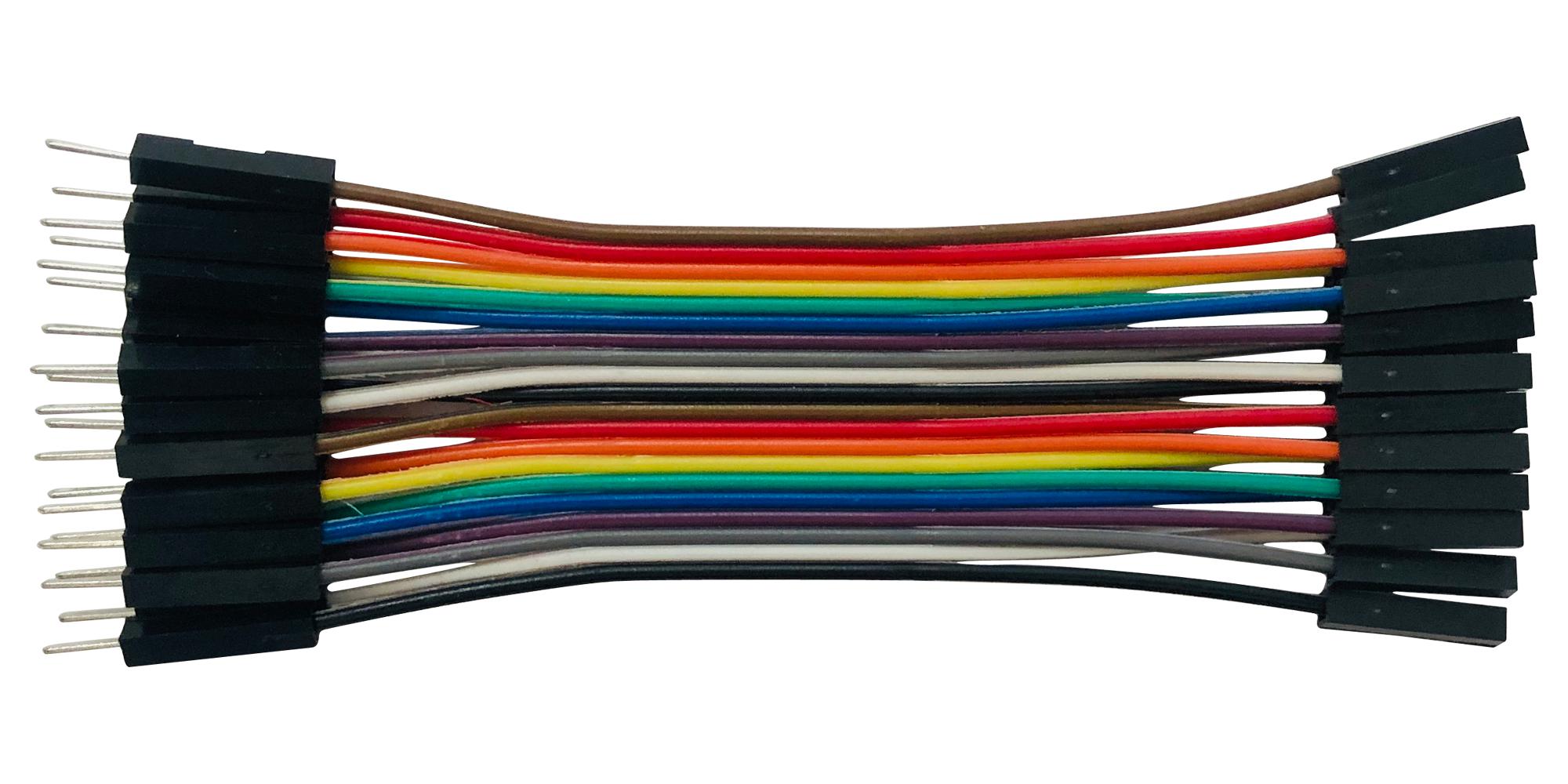


Figure 3.5.2 Jumper Wire

3.6 **Soil moisture Sensor**

A soil moisture sensor is an electronic device used to measure the water content in soil, making it essential for agricultural automation, gardening, and environmental monitoring. It typically consists of two probes that are inserted into the soil to measure its electrical conductivity—wet soil conducts electricity better than dry soil, allowing the sensor to estimate moisture levels. The sensor outputs either analog or digital signals, depending on the model, which can be read by microcontrollers like Arduino or ESP32. Analog sensors provide a variable voltage corresponding to moisture levels, while digital sensors give a binary output based on a set threshold.

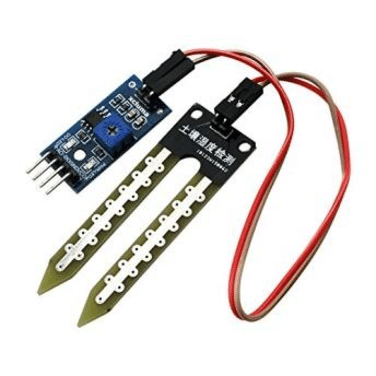


Figure 3.6 Soil moisture Sensor

3.7 **Ultrasonic Sensor**

An ultrasonic sensor is a device used to measure distance by emitting high-frequency sound waves and calculating the time it takes for the echo to return after hitting an object. It typically consists of a transmitter (which sends out ultrasonic pulses) and a receiver (which detects the reflected waves). The sensor uses the time-of-flight principle to determine the distance to an object based on the speed of sound in air. Ultrasonic sensors are widely used in applications such as obstacle detection, level measurement, robot navigation, and parking assistance systems. They are effective in detecting objects regardless of color or transparency, but their performance can be affected by environmental factors like temperature, humidity, and surface texture. Most commonly used modules, like the HC-SR04, provide digital output and are easy to interface with microcontrollers. These sensors are reliable, low-cost, and suitable for both indoor and outdoor use, though they may struggle with soft or angled surfaces that absorb or deflect sound waves.



Figure 3.7 Ultrasonic Sensor

CHAPTER 4

Software Tools

4.1 STM32CubeIDE

STMicroelectronics’ STM32CubeIDE is a comprehensive development environment that combines code editing, configuration, building, flashing, and debugging features for STM32 microcontrollers. It has a robust Device Configuration Tool (Cube MX) that enables graphical configuration of clocks, middleware, peripherals, and pin mapping. Drivers, HAL, or LL APIs can be seamlessly integrated with automated code generation. The STM32CubeIDE is appropriate for both novice and experienced embedded developers since it offers multicore debugging, project management, and version control. From peripheral setup to real-time code monitoring, it offers a structured workflow for sensor-based applications, motor control, and traffic light systems.

STM32CubeMX is widely used in embedded systems development for:

1. Rapid prototyping
2. Peripheral initialization
3. Reducing development time and errors
4. Generating boilerplate code for STM32 projects

4.2 CMSIS-Based Coding Approach

the CMSIS (Cortex Microcontroller Software Interface Standard), a standardized API layer offered by ARM, is supported by STM32CubeIDE for development. This contains CMSIS-Driver for peripheral abstraction and CMSIS-Core for low-level processor access. To guarantee code portability and maintainability, developers can use CMSIS-based function calls in conjunction with ST’s HAL 14 (Hardware Abstraction Layer) or LL (Low Layer) drivers. CMSIS-based programming promotes clean code organization in RTOS and bare-metal programs by enabling structured access to registers, interrupts, and vector tables while also conforming to ARM Cortex-M standards

Chapter 5

System Design and Flowcharts

5.1 Theory of Working

The Vehicle Health monitoring system (VHMS) is designed to monitor vehicle health parameters engine temperature, cabin humidity, Fuel level in the tank, Detection of hurdle while parking and also detection of vibrations or tilts in the axle and chassis. The system is built around the STM32F405RGT6 microcontroller, which interfaces with multiple sensors and actuators. The system gets on as soon as the vehicle gets on or we can say microcontroller gpt power supply. The DHT11 sensor here is used for measuring the temperature and humidity. The soil moisture sensor usually used in the fields but here it is dipped in a liquid and as the level decreases its ADC value increases and it gives us the indication of low fuel. While parking HCSR04 also known as ultrasonic sensor comes in picture by emitting soundwaves which detects hurdles if some are present there. MPU- 6050 here used as tilt detector and using Coriolis effect for doing this. For displaying all this a 16x2 LCD is used which enables user to see what is happening in the systems. To show status to user this display is refreshing constantly. There are some actuators are also present like motors fans and buzzers and LED to show some normal and abnormal conditions. STM32 microcontroller here is commanded for continuous monitoring and capable of taking actions in abnormal conditions like overheating etc. If engine gets overheated actuators got on to cool down the engine and buzzer is her to provide us the information whether there is any hurdle or not. Some LEDs here are glowing to show the normal conditions. This whole make a system which is capable of monitoring the vehicle health constantly.

5.2 Flow Chart

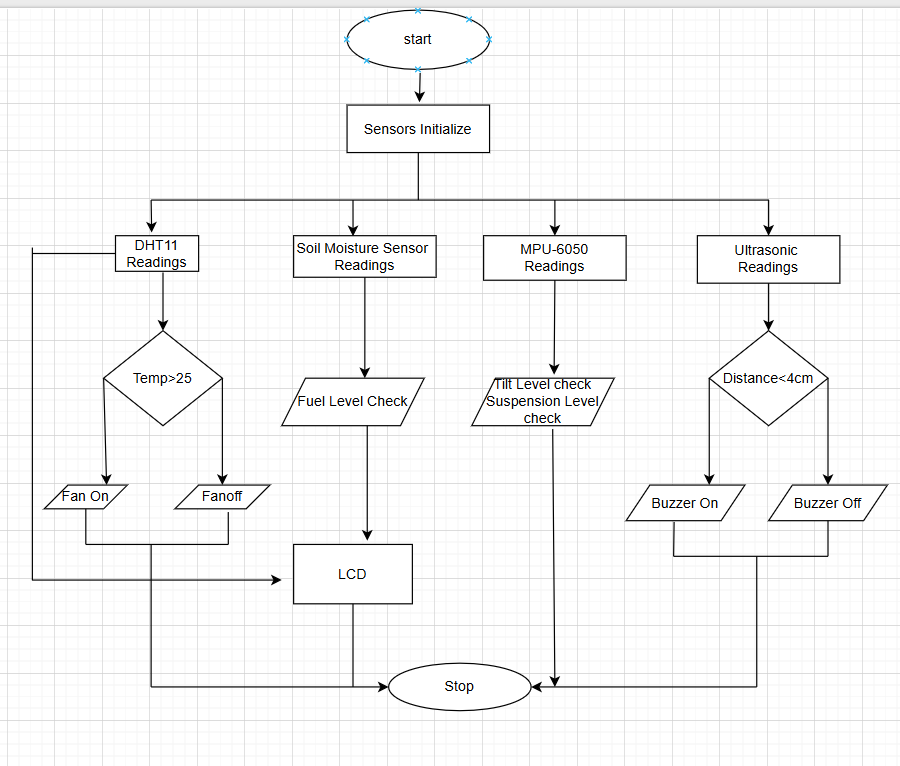


Figure 5.1 Flow chart

5.2.1 Use case Chart

The below diagram is a Use Case UML (Unified Modeling Language) diagram, which visually represents the interaction between external actors (users or systems) and the functionalities (use cases) of a system. In this diagram, the primary actors are the User and the system represents the Vehicle Health monitoring system. It includes Ultrasonic sensor, Soil moisture sensor, Mpu-6050 and DHT11 which gives the readings for engine temperature, humidity, Tilts in axle and hurdle detection. After successful access, it gives all the readings and gives command to actuators like fans and buzzer. Variation in axis gives tilts check for the vehicle.

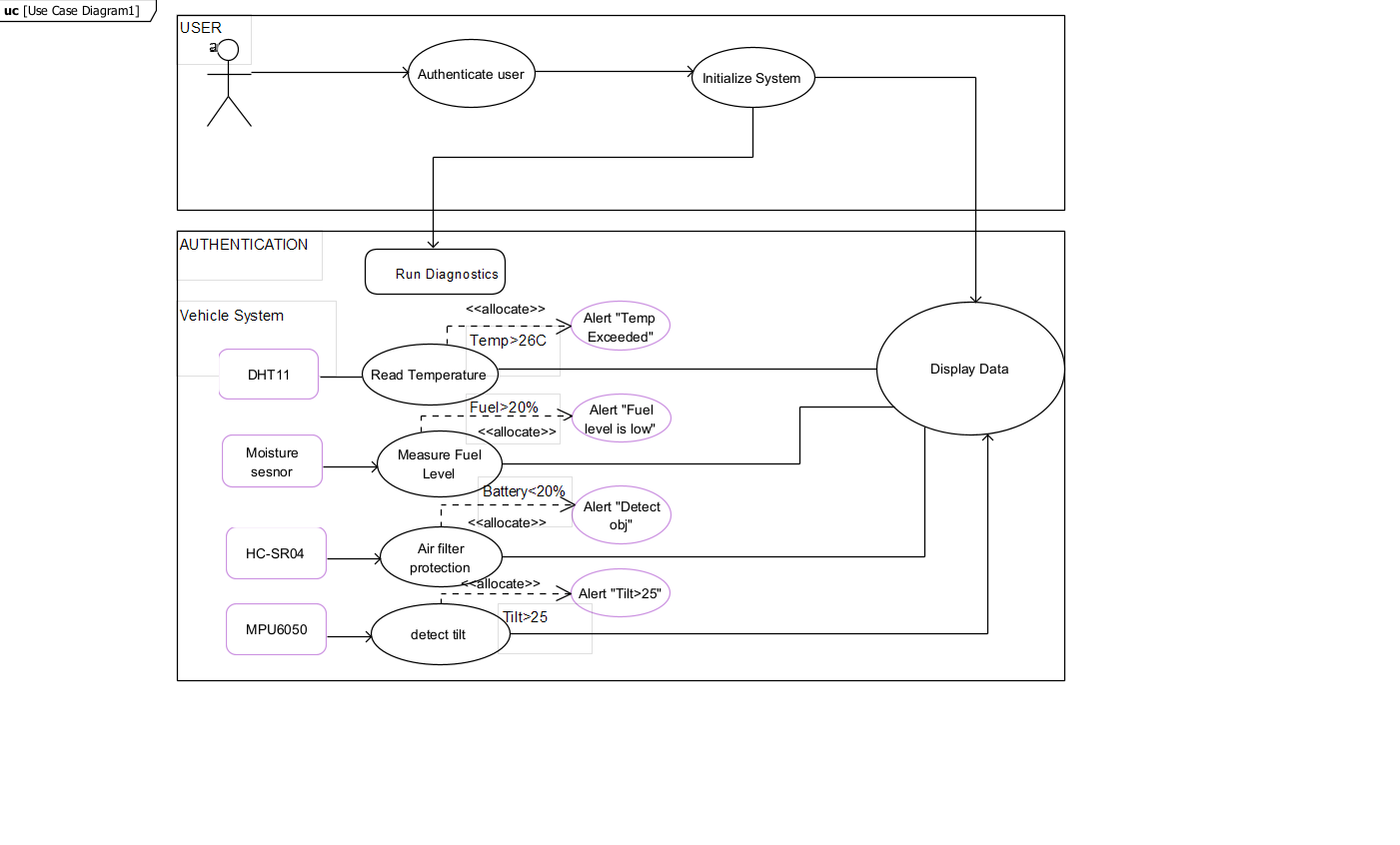


Figure 5.2 use case chart

5.3 Data Flow chart

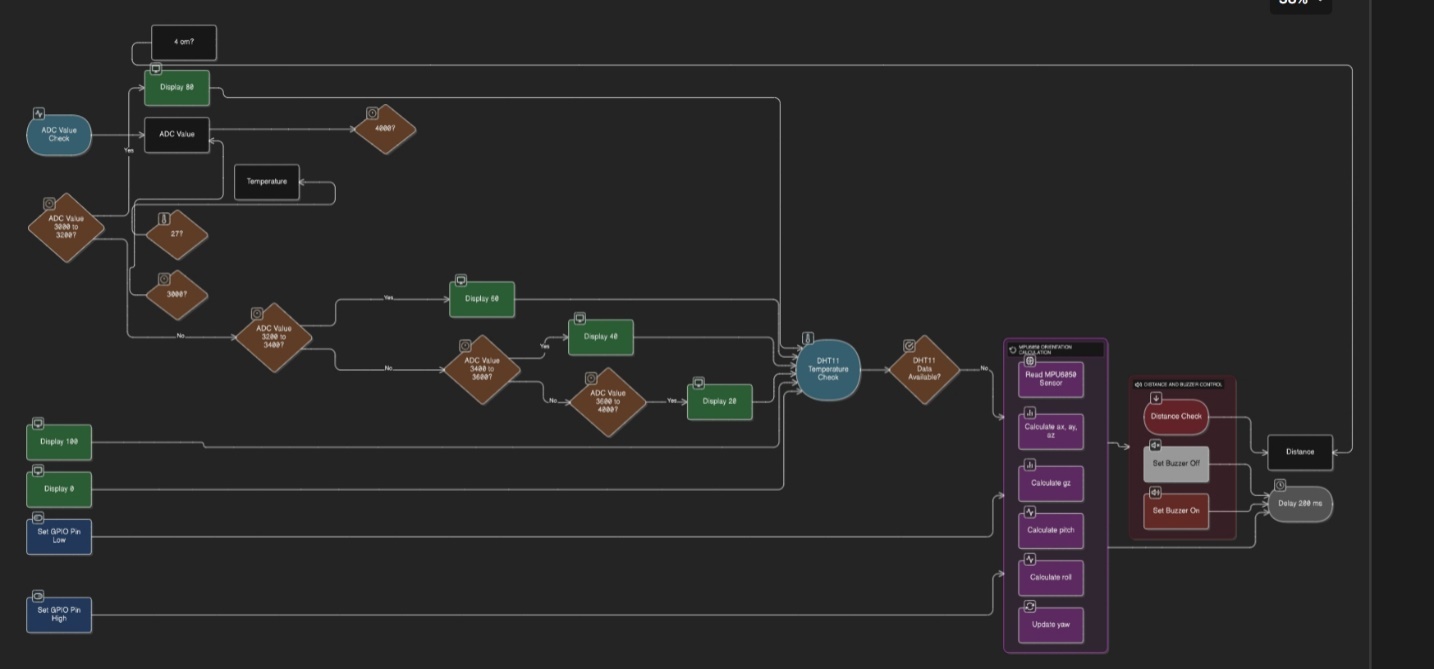
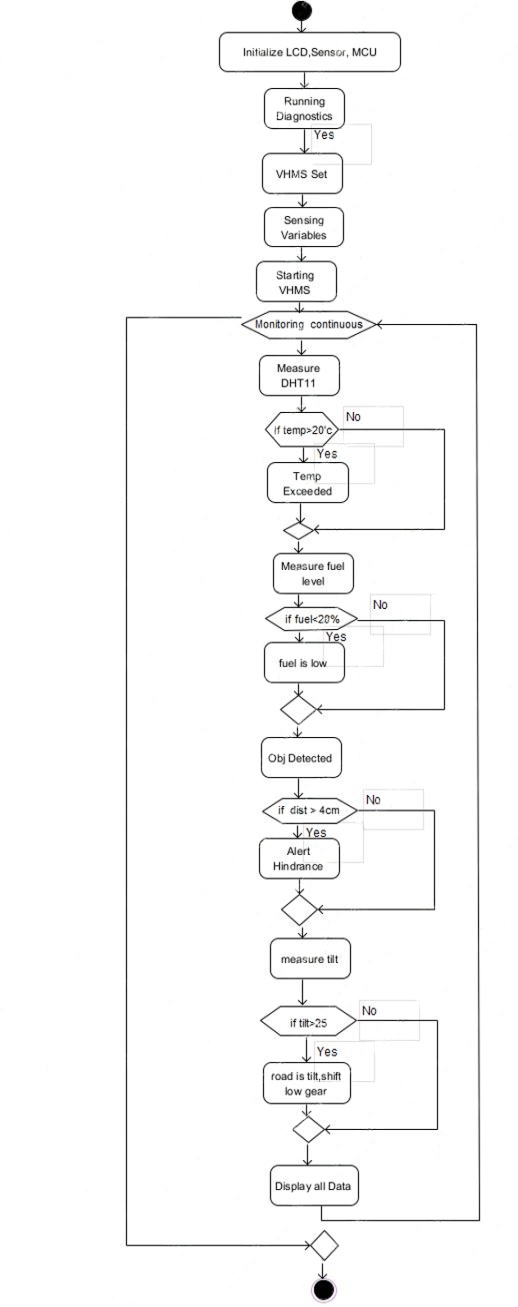


Figure 5.3 Data Flow chart

5.4 Sequence Flow Chart



5.5 Final sequence

The above UML Sequence Diagram illustrates the time-ordered interactions between the user, STM32F407 microcontroller, sensors, and output devices in the Vehicle Health monitoring system. It covers modules like humidity Monitoring, Temperature Sensing, Health Status Indication and Data Transmission. The process starts with the user powering on the device and then starting of each sensor. Upon authentication, the system initializes components, captures health metrics, displays them, triggers alerts if needed, and sends data continuously to a mobile device via I2C for MPU6050. This diagram highlights the sequential coordination of hardware and software for real-time monitoring.

Chapter 6

Software Implementation

6.1 GPIO pins configuration

DHT11 Configuration

|  |  |
| --- | --- |
| VCC | PIN13 |
| GND | PIN 12 |
| DATA (SIGNAL PIN) | PA2 (16) |

Table 6.1.1

Ultrasonic Sensor Configuration

|  |  |
| --- | --- |
| VCC | Debugger Vcc |
| GND | Debugger Ground |
| Trigger | Pin 22 |
| Echo | Pin 23 |

Table 6.1.2

MPU6050 Configuration

|  |  |
| --- | --- |
| VCC | ST-LINK DEGUGGER (+3V) |
| GND | ST-LINK DEBUGGER (GND) |
| SCL | PIN 28 |
| SDA | PIN 29 |

Table 6.1.3

Soil Moisture sensor

|  |  |
| --- | --- |
| VCC | PIN 19 |
| GND | PIN 18 |
| DATA | PIN 20 |

Table 6.1.4

RELAY AND MOTOR

|  |  |
| --- | --- |
| RELAY SIGNAL | PIN 21 |
| RELAY +VE | +VE 12V Adapter |
| RELAY -VE | -VE 12V Adapter |
| RELAY NO | Motor +ve |
| MOTOR -VE | -VE 12 V Adapter |

Table 6.1.5

6.2 Software Circuit Diagram

This schematic represents a health monitoring and control system built around the STM32F405RGT6 microcontroller. The STM32 serves as the brain of the project, coordinating data collection, user input, wireless communication, and output display. LCD display is used for showing numerical readings such as heart rate, SpO, or temperature in real time. The displays are driven using multiplexing techniques to minimize pin usage while ensuring clear and rapid updates. The circuit also integrates a MAX30102 pulse Software Circuit Diagram oximeter and heart rate sensor and MPU 6050 connected to the STM32 via the I²C interface (SCL and SDA lines). This sensor provides measurements for both pulse rate and blood oxygen levels. Temperature sensor is simply connected in a 1 wire configuration with a signal wire. Voltage supply and ground are attached with the help of breadboard.

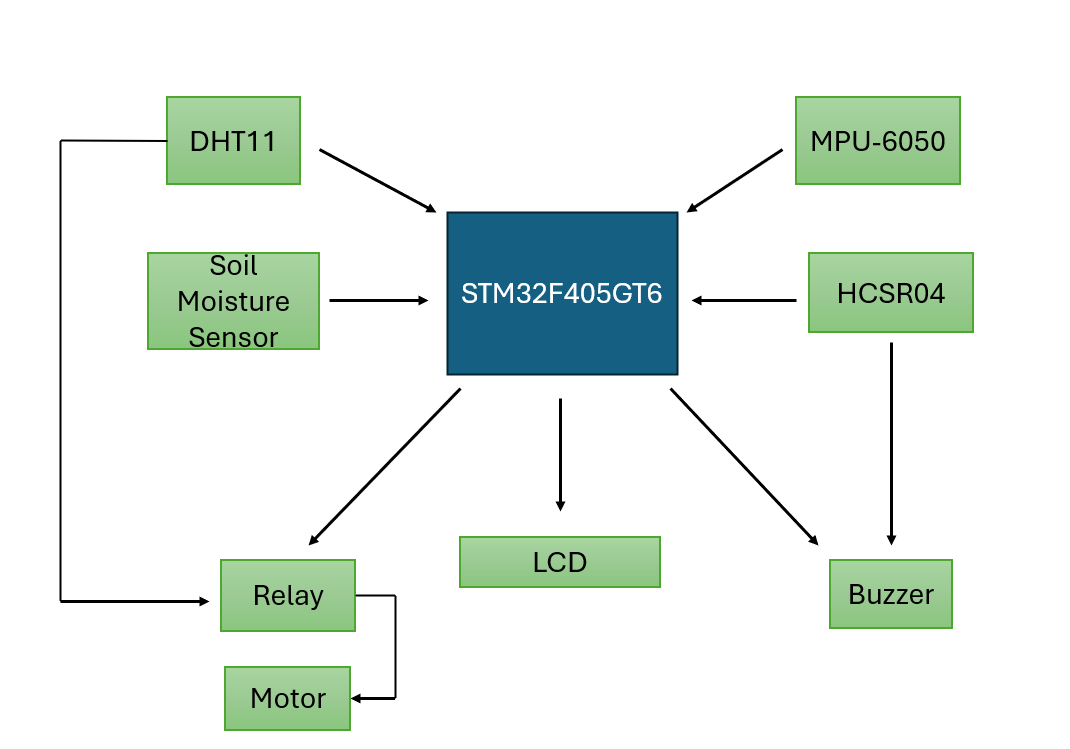


Figure 6.2 Circuit Diagram

Chapter 7

Hardware Implementation

7.1 Hardware setup

This circuit represents a health monitoring system built around the STM32F405RGT6 microcontroller. The STM32 acts as the central controller, managing communication with various sensors and output displays. A 16x2 LCD display allowing it to show numerical data such as Temperature, humidity and Fuel level. This system is using mainly four sensors for this work done. DHT11 monitors the heat rate and humidity of vehicle while soil moisture is used for checking the fuel level in the tank. Ultrasonic sensor is used for detection of hurdle which in results beeping of buzzer of any detected. MPU6050 here is used for the axle and suspension alignment. The actuators here are used with a connection of relay to avoid unnecessary current surges. If temperature goes above threshold it results in starting of fans. All this thing is displayed on a 16X2 LCD display for user’s ease.

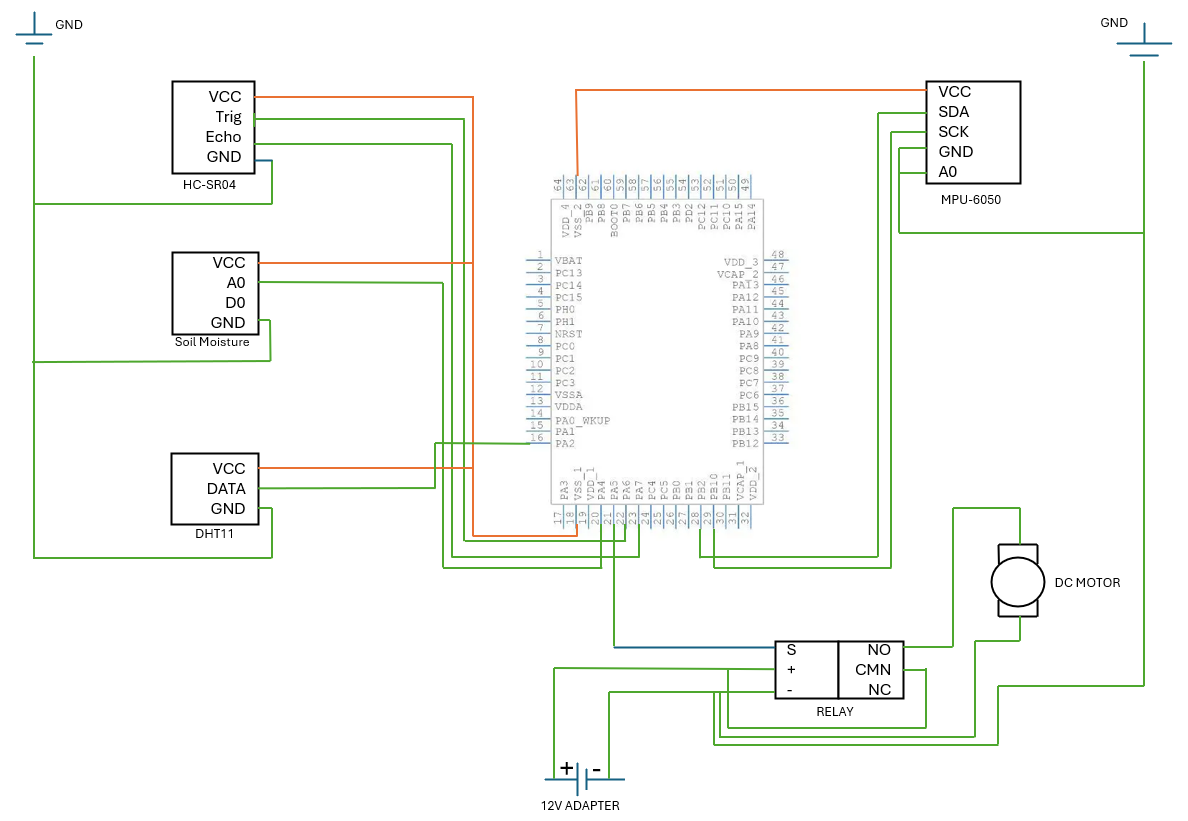


Figure 7.1 Hardware connections

Chapter 8

Source Code

8.1 main.c

#include "main.h"

#include "dht11.h"

#include <math.h>

#include "stm32f405xx.h"

#include "system\_stm32f4xx.h"

#include "stm32f4xx\_hal.h"

#include "lcd.h"

#include <stdio.h>

#define MPU6050\_ADDR       0x68

#define MPU6050\_REG\_PWR    0x6B

#define MPU6050\_REG\_ACCEL  0x3B

#define TRIG\_PIN    GPIO\_PIN\_6

#define ECHO\_PIN    GPIO\_PIN\_7

#define TRIG\_PORT   GPIOA

#define ECHO\_PORT   GPIOA

#define BUZZER\_PIN  GPIO\_PIN\_9

#define BUZZER\_PORT GPIOC

#define TIMEOUT\_US  60000  // 60 ms (~10 m max)

extern TIM\_HandleTypeDef htim3;

extern TIM\_HandleTypeDef htim4;

// --- TIM handles ---

TIM\_HandleTypeDef htim3;  // TIM3 for microsecond counter

TIM\_HandleTypeDef htim4;  // TIM4 for timeout counter

// Sensor data

int16\_t accel[3], gyro[3], temp;

// Alignment variables

volatile float pitch = 0.0f, roll = 0.0f, yaw = 0.0f;

float dt = 0.01f;

ADC\_HandleTypeDef hadc1;

TIM\_HandleTypeDef htim2;

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_ADC1\_Init(void);

static void MX\_TIM2\_Init(void);

void I2C2\_Init(void);

void I2C2\_Start(void);

void I2C2\_Stop(void);

void I2C2\_Write(uint8\_t data);

uint8\_t I2C2\_Read\_Ack(void);

uint8\_t I2C2\_Read\_Nack(void);

void I2C2\_WriteReg(uint8\_t devAddr, uint8\_t reg, uint8\_t data);

void I2C2\_ReadMulti(uint8\_t devAddr, uint8\_t reg, uint8\_t \*buf, uint8\_t len);

void MPU6050\_Init(void);

void MPU6050\_ReadRaw(int16\_t \*accel, int16\_t \*gyro, int16\_t \*temp);

uint32\_t adc\_value =0;

DHT11\_Data dht;

void delay\_us(uint32\_t us)

{

    // Reset the counter

    \_\_HAL\_TIM\_SET\_COUNTER(&htim3, 0);

    // Wait until the counter reaches the desired time

    while (\_\_HAL\_TIM\_GET\_COUNTER(&htim3) < us)

    {

        // Do nothing, just wait

    }

}

void TIM3\_us\_init(void)

{

    \_\_HAL\_RCC\_TIM3\_CLK\_ENABLE();

    htim3.Instance = TIM3;

    htim3.Init.Prescaler = (SystemCoreClock / 1000000) - 1; // 1 MHz → 1 µs tick

    htim3.Init.CounterMode = TIM\_COUNTERMODE\_UP;

    htim3.Init.Period = 0xFFFF;

    htim3.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

    HAL\_TIM\_Base\_Init(&htim3);

    HAL\_TIM\_Base\_Start(&htim3);

}

// Initialize TIM4 for timeout

void TIM4\_us\_init(void)

{

    \_\_HAL\_RCC\_TIM4\_CLK\_ENABLE();

    htim4.Instance = TIM4;

    htim4.Init.Prescaler = (SystemCoreClock / 1000000) - 1; // 1 MHz → 1 µs tick

    htim4.Init.CounterMode = TIM\_COUNTERMODE\_UP;

    htim4.Init.Period = 0xFFFF;

    htim4.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

    HAL\_TIM\_Base\_Init(&htim4);

    HAL\_TIM\_Base\_Start(&htim4);

}

// GPIO Init

void GPIO\_Init\_Ultrasonic(void)

{

    \_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

    GPIO\_InitTypeDef GPIO\_InitStruct = {0};

    // TRIG pin output

    GPIO\_InitStruct.Pin = TRIG\_PIN;

    GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

    GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_HIGH;

    HAL\_GPIO\_Init(TRIG\_PORT, &GPIO\_InitStruct);

    // ECHO pin input

    GPIO\_InitStruct.Pin = ECHO\_PIN;

    GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

    HAL\_GPIO\_Init(ECHO\_PORT, &GPIO\_InitStruct);

}

void GPIO\_Init\_Buzzer(void)

{

    \_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();

    GPIO\_InitTypeDef GPIO\_InitStruct = {0};

    GPIO\_InitStruct.Pin = BUZZER\_PIN;

    GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

    GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

    HAL\_GPIO\_Init(BUZZER\_PORT, &GPIO\_InitStruct);

    HAL\_GPIO\_WritePin(BUZZER\_PORT, BUZZER\_PIN, GPIO\_PIN\_RESET);

}

// Trigger pulse

void send\_trigger(void)

{

    HAL\_GPIO\_WritePin(TRIG\_PORT, TRIG\_PIN, GPIO\_PIN\_SET);

    delay\_us(10);

    HAL\_GPIO\_WritePin(TRIG\_PORT, TRIG\_PIN, GPIO\_PIN\_RESET);

}

// Read ECHO pulse width in us using TIM3 + TIM4

uint32\_t read\_echo\_time(void)

{

    \_\_HAL\_TIM\_SET\_COUNTER(&htim4, 0);

    // Wait for ECHO HIGH

    while (HAL\_GPIO\_ReadPin(ECHO\_PORT, ECHO\_PIN) == GPIO\_PIN\_RESET)

    {

        if (\_\_HAL\_TIM\_GET\_COUNTER(&htim4) > TIMEOUT\_US)

            return 0; // Timeout

    }

    // Measure HIGH duration

    \_\_HAL\_TIM\_SET\_COUNTER(&htim3, 0);

    \_\_HAL\_TIM\_SET\_COUNTER(&htim4, 0);

    while (HAL\_GPIO\_ReadPin(ECHO\_PORT, ECHO\_PIN) == GPIO\_PIN\_SET)

    {

        if (\_\_HAL\_TIM\_GET\_COUNTER(&htim4) > TIMEOUT\_US)

            return 0; // Timeout

    }

    return \_\_HAL\_TIM\_GET\_COUNTER(&htim3);

}

// Clock config (same as your version)

void SystemClock\_Config(void)

{

    RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

    RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

    \_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

    \_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE2);

    RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

    RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

    RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

    RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

    RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;

    RCC\_OscInitStruct.PLL.PLLM = 16;

    RCC\_OscInitStruct.PLL.PLLN = 336;

    RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV4; // SYSCLK = 84 MHz

    RCC\_OscInitStruct.PLL.PLLQ = 7;

    if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

    {

        while(1);

    }

    RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK | RCC\_CLOCKTYPE\_SYSCLK |

                                  RCC\_CLOCKTYPE\_PCLK1 | RCC\_CLOCKTYPE\_PCLK2;

    RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

    RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

    RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;  // 42 MHz

    RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;  // 84 MHz

    if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)

    {

        while(1);

    }

}

int main(void)

{

        HAL\_Init();

        SystemClock\_Config();

        LcdInit();

        TIM3\_us\_init();   // ✅ use TIM3 instead of TIM2

        TIM4\_us\_init();   // timeout timer

        GPIO\_Init\_Ultrasonic();

        GPIO\_Init\_Buzzer();

        char str[20];

        uint32\_t echo\_time;

        float distance\_cm;

        HAL\_Delay(100);

       I2C2\_Init();

       MPU6050\_Init();

       DWT\_Init();

      MX\_GPIO\_Init();

      MX\_ADC1\_Init();

       MX\_TIM2\_Init();

      lprint(0x80, "Distance:");

  while (1)

  {

              send\_trigger();

              echo\_time = read\_echo\_time();

              distance\_cm = echo\_time / 58.0f;  // Convert us → cm

              uint32\_t int\_part = (uint32\_t)distance\_cm;

              uint32\_t frac\_part = (uint32\_t)((distance\_cm - int\_part) \* 10);

              sprintf(str, "%lu.%lu cm", int\_part, frac\_part);

              lprint(0xC0, str);

              if (distance\_cm < 4.0f)

                  HAL\_GPIO\_WritePin(BUZZER\_PORT, BUZZER\_PIN, GPIO\_PIN\_SET);

              else

                  HAL\_GPIO\_WritePin(BUZZER\_PORT, BUZZER\_PIN, GPIO\_PIN\_RESET);

              HAL\_Delay(200);

      if (DHT11\_GetData(GPIOA, GPIO\_PIN\_2, &dht)) {

            }

            if(dht.Temperature < 27 )

            {

                GPIOA->ODR &=~(1<<5);

            }

            else

            {

                GPIOA->ODR |=(1<<5);

            }

      HAL\_ADC\_Start(&hadc1);

              HAL\_ADC\_PollForConversion(&hadc1, HAL\_MAX\_DELAY);

                        adc\_value = HAL\_ADC\_GetValue(&hadc1);

                        if (adc\_value > 3000)

                        {

                            GPIOB->ODR |= (1 << 13);

                            HAL\_Delay(100);

                            GPIOB->ODR &=~(1<< 13);

                            HAL\_Delay(100);

                        }

                        else

                        {

                            GPIOB->ODR &=~(1<<13);

                        }

                     {

                            MPU6050\_ReadRaw(accel, gyro, &temp);

                            float ax = accel[0] / 16384.0f;

                            float ay = accel[1] / 16384.0f;

                            float az = accel[2] / 16384.0f;

                            float gz = gyro[2] / 131.0f;

                            pitch = roundf(atan2f(ax, sqrtf(ay \* ay + az \* az)) \* 180.0f / M\_PI \* 100.0f) / 100.0f;

                            roll  = roundf(atan2f(ay, sqrtf(ax \* ax + az \* az)) \* 180.0f / M\_PI \* 100.0f) / 100.0f;

                            yaw  += gz \* dt;

                            yaw   = roundf(yaw \* 100.0f) / 100.0f;

                            for (volatile int i = 0; i < 100000; i++);

                        }

  }

  /\* USER CODE END 3 \*/

}

        void I2C2\_Init(void)

        {

            RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOBEN;

            RCC->APB1ENR |= RCC\_APB1ENR\_I2C2EN;

            GPIOB->MODER   &= ~((3 << (10\*2)) | (3 << (11\*2)));

            GPIOB->MODER   |=  (2 << (10\*2)) | (2 << (11\*2));

            GPIOB->OTYPER  |=  (1 << 10) | (1 << 11);

            GPIOB->PUPDR   |=  (1 << (10\*2)) | (1 << (11\*2));

            GPIOB->OSPEEDR |=  (3 << (10\*2)) | (3 << (11\*2));

            GPIOB->AFR[1]  |=  (4 << ((10-8)\*4)) | (4 << ((11-8)\*4));

            I2C2->CR1 = I2C\_CR1\_SWRST;

            I2C2->CR1 = 0;

            I2C2->CR2 = 42;

            I2C2->CCR = 210;

            I2C2->TRISE = 43;

            I2C2->CR1 |= I2C\_CR1\_PE;

        }

        void I2C2\_Start(void)

        {

            I2C2->CR1 |= I2C\_CR1\_START;

            while (!(I2C2->SR1 & I2C\_SR1\_SB));

        }

        void I2C2\_Stop(void)

        {

            I2C2->CR1 |= I2C\_CR1\_STOP;

        }

        void I2C2\_Write(uint8\_t data)

        {

            while (!(I2C2->SR1 & I2C\_SR1\_TXE));

            I2C2->DR = data;

            while (!(I2C2->SR1 & I2C\_SR1\_BTF));

        }

        uint8\_t I2C2\_Read\_Ack(void)

        {

            I2C2->CR1 |= I2C\_CR1\_ACK;

            while (!(I2C2->SR1 & I2C\_SR1\_RXNE));

            return I2C2->DR;

        }

        uint8\_t I2C2\_Read\_Nack(void)

        {

            I2C2->CR1 &= ~I2C\_CR1\_ACK;

            while (!(I2C2->SR1 & I2C\_SR1\_RXNE));

            return I2C2->DR;

        }

        void I2C2\_WriteReg(uint8\_t devAddr, uint8\_t reg, uint8\_t data)

        {

            I2C2\_Start();

            I2C2->DR = devAddr << 1;

            while (!(I2C2->SR1 & I2C\_SR1\_ADDR));

            (void)I2C2->SR2;

            I2C2\_Write(reg);

            I2C2\_Write(data);

            I2C2\_Stop();

        }

        void I2C2\_ReadMulti(uint8\_t devAddr, uint8\_t reg, uint8\_t \*buf, uint8\_t len)

        {

            I2C2\_Start();

            I2C2->DR = devAddr << 1;

            while (!(I2C2->SR1 & I2C\_SR1\_ADDR));

            (void)I2C2->SR2;

            I2C2\_Write(reg);

            I2C2\_Start();

            I2C2->DR = (devAddr << 1) | 1;

            while (!(I2C2->SR1 & I2C\_SR1\_ADDR));

            (void)I2C2->SR2;

            for (uint8\_t i = 0; i < len - 1; i++)

                buf[i] = I2C2\_Read\_Ack();

            buf[len - 1] = I2C2\_Read\_Nack();

            I2C2\_Stop();

        }

        void MPU6050\_Init(void)

        {

            I2C2\_WriteReg(MPU6050\_ADDR, MPU6050\_REG\_PWR, 0x00);

        }

        void MPU6050\_ReadRaw(int16\_t \*accel, int16\_t \*gyro, int16\_t \*temp)

        {

            uint8\_t buf[14];

            I2C2\_ReadMulti(MPU6050\_ADDR, MPU6050\_REG\_ACCEL, buf, 14);

            accel[0] = (buf[0] << 8) | buf[1];

            accel[1] = (buf[2] << 8) | buf[3];

            accel[2] = (buf[4] << 8) | buf[5];

            \*temp    = (buf[6] << 8) | buf[7];

            gyro[0]  = (buf[8] << 8) | buf[9];

            gyro[1]  = (buf[10] << 8) | buf[11];

            gyro[2]  = (buf[12] << 8) | buf[13];

        }

        static void MX\_ADC1\_Init(void)

        {

          ADC\_ChannelConfTypeDef sConfig = {0};

          hadc1.Instance = ADC1;

          hadc1.Init.ClockPrescaler = ADC\_CLOCK\_SYNC\_PCLK\_DIV2;

          hadc1.Init.Resolution = ADC\_RESOLUTION\_12B;

          hadc1.Init.ScanConvMode = DISABLE;

          hadc1.Init.ContinuousConvMode = DISABLE;

          hadc1.Init.DiscontinuousConvMode = DISABLE;

          hadc1.Init.ExternalTrigConvEdge = ADC\_EXTERNALTRIGCONVEDGE\_NONE;

          hadc1.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;

          hadc1.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

          hadc1.Init.NbrOfConversion = 1;

          hadc1.Init.DMAContinuousRequests = DISABLE;

          hadc1.Init.EOCSelection = ADC\_EOC\_SINGLE\_CONV;

          if (HAL\_ADC\_Init(&hadc1) != HAL\_OK)

          {

            Error\_Handler();

          }

          sConfig.Channel = ADC\_CHANNEL\_4;

          sConfig.Rank = 1;

          sConfig.SamplingTime = ADC\_SAMPLETIME\_3CYCLES;

          if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

          {

            Error\_Handler();

          }

        }

        static void MX\_TIM2\_Init(void)

        {

          TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

          TIM\_MasterConfigTypeDef sMasterConfig = {0};

          htim2.Instance = TIM2;

          htim2.Init.Prescaler = 15;

          htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP;

          htim2.Init.Period = 65535;

          htim2.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

          htim2.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

          if (HAL\_TIM\_Base\_Init(&htim2) != HAL\_OK)

          {

            Error\_Handler();

          }

          sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

          if (HAL\_TIM\_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL\_OK)

          {

            Error\_Handler();

          }

          sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

          sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

          if (HAL\_TIMEx\_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL\_OK)

          {

            Error\_Handler();

          }

        }

        static void MX\_GPIO\_Init(void)

        {

          \_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

          RCC->AHB1ENR |=(1<<1); // Port B

          RCC->AHB1ENR |=(1<<2); // Port C

          GPIOC->MODER |=(1<<18);  // PC9 as buzzer

          GPIOC->MODER &=~(1<<19);

          GPIOB->MODER |=(1<<26); // PB13 as led

          GPIOB->MODER &=~(1<<27);

          GPIOA->MODER |=(1<<4); // PA2 as opt for dht

          GPIOA->MODER &=~(1<<5);

          GPIOA->MODER |=(1<<10);

          GPIOA->MODER &=~(1<<11);// PA5 as op relay

          GPIOA->ODR |=(1<<5);

        }

        void Error\_Handler(void)

        {

          \_\_disable\_irq();

          while (1)

          {

          }

        }

        #ifdef  USE\_FULL\_ASSERT

        void assert\_failed(uint8\_t \*file, uint32\_t line)

        {

        }

        #endif /\* USE\_FULL\_ASSERT \*/

8.2 MPU 6050.c

**#include** <stdint.h>

**#include**"LCD.h"

**#include**"cmn.h"

**#if** !defined(\_\_SOFT\_FP\_\_) && defined(\_\_ARM\_FP)

// #warning "FPU is not initialized, but the project is compiling for an FPU. Please initialize the FPU before use."

**#endif**

**#include** "stm32f405xx.h"

**#include** "system\_stm32f4xx.h"

**#define** MPU6050\_ADDR 0x68 // AD0=GND => 0x68

**#define** MPU6050\_REG\_PWR 0x6B

**#define** MPU6050\_REG\_ACCEL 0x3B

**#define** MPU6050\_REG\_GYRO 0x43

**#define** MPU6050\_REG\_TEMP 0x41

**#define** I2C\_TIMEOUT 10000

**void** **I2C2\_Init**(**void**);

**void** **I2C2\_Start**(**void**);

**void** **I2C2\_Stop**(**void**);

**void** **steps**();

**void** **I2C2\_Write**(uint8\_t data);

uint8\_t **I2C2\_Read\_Ack**(**void**);

uint8\_t **I2C2\_Read\_Nack**(**void**);

**void** **I2C2\_WriteReg**(uint8\_t devAddr, uint8\_t reg, uint8\_t data);

**void** **I2C2\_ReadMulti**(uint8\_t devAddr, uint8\_t reg, uint8\_t \*buf, uint8\_t len);

**void** **MPU6050\_Init**(**void**);

**void** **MPU6050\_ReadRaw**(int16\_t \*accel, int16\_t \*gyro, int16\_t \*temp);

int16\_t accel[3], gyro[3], temp;

**int** step =0 ;

**int** i,j=0,s=0;

**void** **time**();

**void** **tim\_init**();

**void** **gpio\_init**();

**int** **main**(**void**)

{

LcdInit();

I2C2\_Init();

MPU6050\_Init();

tim\_init();

gpio\_init();

**while** (1)

{

**if**(!(GPIOB->IDR &(1<<3)))

{

**while**(1)

{time();}

}

// if(!(GPIOB->IDR & (1<<7)))

// {

lprint(0xc2,"steps ");

// MPU6050\_Init();

// lprint(0x82,"steps");

MPU6050\_ReadRaw(accel, gyro, &temp);

// Place a breakpoint here to see live values

// accel[0], accel[1], accel[2]

// gyro[0], gyro[1], gyro[2]

// temp

**for**(**volatile** **int** i=0; i<100000; i++); // Small delay

steps();

// }

}

}

/\* ------------------- I2C2 Initialization ------------------- \*/

**void** **I2C2\_Init**(**void**)

{

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOBEN; // Enable GPIOB clock

RCC->APB1ENR |= RCC\_APB1ENR\_I2C2EN; // Enable I2C2 clock

// PB10=SCL, PB11=SDA → AF4, OpenDrain, PullUp, HighSpeed

GPIOB->MODER &= ~((3 << (10\*2)) | (3 << (11\*2)));

GPIOB->MODER |= (2 << (10\*2)) | (2 << (11\*2)); // AF mode

GPIOB->OTYPER |= (1 << 10) | (1 << 11); // Open-drain

GPIOB->PUPDR |= (1 << (10\*2)) | (1 << (11\*2)); // Pull-up

GPIOB->OSPEEDR |= (3 << (10\*2)) | (3 << (11\*2)); // High speed

GPIOB->AFR[1] |= (4 << ((10-8)\*4)) | (4 << ((11-8)\*4)); // AF4

I2C2->CR1 = I2C\_CR1\_SWRST; // Reset I2C

I2C2->CR1 = 0;

I2C2->CR2 = 42; // APB1=42MHz

I2C2->CCR = 210; // 100kHz => CCR=42MHz/(2\*100k)=210

I2C2->TRISE = 43; // TRISE = Freq + 1

I2C2->CR1 |= I2C\_CR1\_PE; // Enable I2C2

}

/\* ------------------- I2C2 Low-Level Functions ------------------- \*/

**void** **I2C2\_Start**(**void**)

{

I2C2->CR1 |= I2C\_CR1\_START;

**while**(!(I2C2->SR1 & I2C\_SR1\_SB));

}

**void** **I2C2\_Stop**(**void**)

{

I2C2->CR1 |= I2C\_CR1\_STOP;

}

**void** **I2C2\_Write**(uint8\_t data)

{

**while**(!(I2C2->SR1 & I2C\_SR1\_TXE));

I2C2->DR = data;

**while**(!(I2C2->SR1 & I2C\_SR1\_BTF));

}

uint8\_t **I2C2\_Read\_Ack**(**void**)

{

I2C2->CR1 |= I2C\_CR1\_ACK;

**while**(!(I2C2->SR1 & I2C\_SR1\_RXNE));

**return** I2C2->DR;

}

uint8\_t **I2C2\_Read\_Nack**(**void**)

{

I2C2->CR1 &= ~I2C\_CR1\_ACK;

**while**(!(I2C2->SR1 & I2C\_SR1\_RXNE));

**return** I2C2->DR;

}

/\* ------------------- I2C2 Read & Write ------------------- \*/

**void** **I2C2\_WriteReg**(uint8\_t devAddr, uint8\_t reg, uint8\_t data)

{

I2C2\_Start();

I2C2->DR = devAddr << 1;

**while**(!(I2C2->SR1 & I2C\_SR1\_ADDR));

(**void**)I2C2->SR2;

I2C2\_Write(reg);

I2C2\_Write(data);

I2C2\_Stop();

}

**void** **I2C2\_ReadMulti**(uint8\_t devAddr, uint8\_t reg, uint8\_t \*buf, uint8\_t len)

{

I2C2\_Start();

I2C2->DR = devAddr << 1;

**while**(!(I2C2->SR1 & I2C\_SR1\_ADDR));

(**void**)I2C2->SR2;

I2C2\_Write(reg);

I2C2\_Start();

I2C2->DR = (devAddr << 1) | 1;

**while**(!(I2C2->SR1 & I2C\_SR1\_ADDR));

(**void**)I2C2->SR2;

**for**(uint8\_t i=0; i<len-1; i++)

buf[i] = I2C2\_Read\_Ack();

buf[len-1] = I2C2\_Read\_Nack();

I2C2\_Stop();

}

/\* ------------------- MPU6050 Functions ------------------- \*/

**void** **MPU6050\_Init**(**void**)

{

// Wake up the sensor

I2C2\_WriteReg(MPU6050\_ADDR, MPU6050\_REG\_PWR, 0x00);

}

**void** **steps**()

{

**if**(accel[0] > 200)

{

step++;

aprint(0xc10,step);

}

}

**void** **MPU6050\_ReadRaw**(int16\_t \*accel, int16\_t \*gyro, int16\_t \*temp)

{

uint8\_t buf[14];

I2C2\_ReadMulti(MPU6050\_ADDR, MPU6050\_REG\_ACCEL, buf, 14);

accel[0] = (buf[0] << 8) | buf[1];

accel[1] = (buf[2] << 8) | buf[3];

accel[2] = (buf[4] << 8) | buf[5];

\*temp = (buf[6] << 8) | buf[7];

gyro[0] = (buf[8] << 8) | buf[9];

gyro[1] = (buf[10] << 8) | buf[11];

gyro[2] = (buf[12] << 8) | buf[13];

}

**void** **time**()

{

**for**(i=0;i<60;i++)

{

aprint(0x86,i);

// HAL\_Delay(1000);

}

**if**(i>59)

{

j++;

i=0;

}

aprint(0x83,j);

**if**(j%10 == 0)

{

GPIOC->ODR |=(1<<9);//PC 9 AS BUZZER

HAL\_Delay(1000);

GPIOC->ODR &=~(1<<9);

HAL\_Delay(1000);

lprint(0xc2,"Drink water");

}

**else**

{

lprint(0xc2," ");

}

**if**(j>59)

{

s++;

aprint(0x81,s);

}

}

**void** **tim\_init**()

{

RCC->APB1ENR |=(1<<0);

TIM2->PSC = 1000-1;

TIM2->ARR = 10000-1;

TIM2->CNT =0;

TIM2->CR1 |=(1<<0);

}

**void** **gpio\_init**()

{

RCC->AHB1ENR |=(1<<2);

RCC->AHB1ENR |=(1<<1); // PORT B

GPIOC->MODER |=(1<<18);

GPIOC->MODER &=~(1<<19);

GPIOB->MODER &=~(3<<14); // SW1(PB3) AS INP

GPIOB->MODER &=~(3<<6);

}

8.3 Ultrasonic.c

#include "stm32f4xx.h"  
#include "lcd.h"  
#include <stdio.h>

#define TRIG\_PIN    6   // PA6  
#define ECHO\_PIN    7   // PA7  
#define TIMEOUT\_US 60000  // 60 ms timeout (~10 m max)

#define BUZZER\_PIN  9   // PC9

// --- Microsecond delay using TIM2 ---  
void TIM2\_us\_init(void)  
{  
    RCC->APB1ENR |= RCC\_APB1ENR\_TIM2EN;      // Enable TIM2 clock  
    TIM2->PSC = (SystemCoreClock / 1000000) - 1; // 1 MHz (1 us per tick)  
    TIM2->ARR = 0xFFFF;  
    TIM2->CR1 |= TIM\_CR1\_CEN;  
}

void delay\_us(uint32\_t us)  
{  
    TIM2->CNT = 0;  
    while (TIM2->CNT < us);  
}

// --- TIM4 for timeout ---  
void TIM4\_us\_init(void)  
{  
    RCC->APB1ENR |= RCC\_APB1ENR\_TIM4EN;  
    TIM4->PSC = (SystemCoreClock / 1000000) - 1; // 1 MHz  
    TIM4->ARR = 0xFFFF;  
    TIM4->CR1 |= TIM\_CR1\_CEN;  
}

// --- GPIO Init ---  
void GPIO\_Init\_Ultrasonic(void)  
{  
    RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOAEN;

    // TRIG PA6 output  
    GPIOA->MODER &= ~(3 << (2\*TRIG\_PIN));  
    GPIOA->MODER |=  (1 << (2\*TRIG\_PIN));  // Output  
    GPIOA->OTYPER &= ~(1 << TRIG\_PIN);  
    GPIOA->OSPEEDR |= (3 << (2\*TRIG\_PIN));

    // ECHO PA7 input  
    GPIOA->MODER &= ~(3 << (2\*ECHO\_PIN));  
}

void GPIO\_Init\_Buzzer(void)  
{  
    RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOCEN;  
    GPIOC->MODER &= ~(3 << (2\*BUZZER\_PIN));  
    GPIOC->MODER |=  (1 << (2\*BUZZER\_PIN)); // Output  
    GPIOC->ODR &= ~(1 << BUZZER\_PIN);       // OFF initially  
}

// --- Trigger pulse ---  
void send\_trigger(void)  
{  
    GPIOA->BSRR = (1 << TRIG\_PIN);          // HIGH  
    delay\_us(10);                           // 10 us pulse  
    GPIOA->BSRR = (1 << (TRIG\_PIN + 16));  // LOW  
}

// --- Read ECHO pulse width in us ---  
uint32\_t read\_echo\_time(void)  
{  
    TIM4->CNT = 0;

    // Wait for ECHO HIGH  
    while (!(GPIOA->IDR & (1 << ECHO\_PIN)))  
    {  
        if (TIM4->CNT > TIMEOUT\_US) return 0; // Timeout  
    }

    // Measure HIGH duration  
    TIM2->CNT = 0;  
    TIM4->CNT = 0;  
    while (GPIOA->IDR & (1 << ECHO\_PIN))  
    {  
        if (TIM4->CNT > TIMEOUT\_US) return 0; // Timeout  
    }

    return TIM2->CNT; // in microseconds  
}

// --- Main ---  
int main(void)  
{  
    char str[20];  
    uint32\_t echo\_time;  
    float distance\_cm;

    // Initialize  
    SystemInit();  
    LcdInit();  
    TIM2\_us\_init();  
    TIM4\_us\_init();  
    GPIO\_Init\_Ultrasonic();  
    GPIO\_Init\_Buzzer();

    lprint(0x80, "Distance:");

    while (1)  
    {  
        send\_trigger();  
        echo\_time = read\_echo\_time();

        if (echo\_time == 0)  
        {  
            lprint(0xC0, "D: FAR     ");  
            GPIOC->ODR &= ~(1 << BUZZER\_PIN); // Buzzer OFF  
        }  
        else  
        {  
            distance\_cm = echo\_time / 58.0f; // Convert to cm  
            sprintf(str, "D: %lu cm", (uint32\_t)distance\_cm);  
            lprint(0xC0, str);

            if (distance\_cm < 4.0f)  
                GPIOC->ODR |= (1 << BUZZER\_PIN); // ON  
            else  
                GPIOC->ODR &= ~(1 << BUZZER\_PIN); // OFF  
        }

        // ~100 ms delay  
        for (volatile uint32\_t i=0; i<1000000; i++);  
    }  
}

8.4 DHT11.c

**#include** "ds18b20.h"

**#include** "delay.h"

**#include**"lcd.h"

#include "dht11.h"

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

void DWT\_Init(void) {  
    CoreDebug->DEMCR |= CoreDebug\_DEMCR\_TRCENA\_Msk;  
    DWT->CTRL |= DWT\_CTRL\_CYCCNTENA\_Msk;  
}

void delay\_us(uint32\_t us) {  
    uint32\_t startTick = DWT->CYCCNT;  
    uint32\_t delayTicks = us \* (HAL\_RCC\_GetHCLKFreq() / 1000000);  
    while ((DWT->CYCCNT - startTick) < delayTicks);  
}

static void DHT11\_Set\_Pin\_Output(GPIO\_TypeDef \*GPIOx, uint16\_t GPIO\_Pin) {  
    GPIO\_InitStruct.Pin = GPIO\_Pin;  
    GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;  
    GPIO\_InitStruct.Pull = GPIO\_NOPULL;  
    GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;  
    HAL\_GPIO\_Init(GPIOx, &GPIO\_InitStruct);  
}

static void DHT11\_Set\_Pin\_Input(GPIO\_TypeDef \*GPIOx, uint16\_t GPIO\_Pin) {  
    GPIO\_InitStruct.Pin = GPIO\_Pin;  
    GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;  
    GPIO\_InitStruct.Pull = GPIO\_NOPULL;  
    HAL\_GPIO\_Init(GPIOx, &GPIO\_InitStruct);  
}

static void DHT11\_Start(GPIO\_TypeDef \*GPIOx, uint16\_t GPIO\_Pin) {  
    DHT11\_Set\_Pin\_Output(GPIOx, GPIO\_Pin);  
    HAL\_GPIO\_WritePin(GPIOx, GPIO\_Pin, GPIO\_PIN\_RESET);  
    HAL\_Delay(20);  
    HAL\_GPIO\_WritePin(GPIOx, GPIO\_Pin, GPIO\_PIN\_SET);  
    delay\_us(30);  
    DHT11\_Set\_Pin\_Input(GPIOx, GPIO\_Pin);  
}

static uint8\_t DHT11\_Check\_Response(GPIO\_TypeDef \*GPIOx, uint16\_t GPIO\_Pin) {  
    uint8\_t Response = 0;  
    delay\_us(40);  
    if (!(HAL\_GPIO\_ReadPin(GPIOx, GPIO\_Pin))) {  
        delay\_us(80);  
        if ((HAL\_GPIO\_ReadPin(GPIOx, GPIO\_Pin))) Response = 1;  
        delay\_us(80);  
    }  
    return Response;  
}

static uint8\_t DHT11\_Read(GPIO\_TypeDef \*GPIOx, uint16\_t GPIO\_Pin) {  
    uint8\_t i, data = 0;  
    for (i = 0; i < 8; i++) {  
        while (!(HAL\_GPIO\_ReadPin(GPIOx, GPIO\_Pin)));  
        delay\_us(40);  
        if (HAL\_GPIO\_ReadPin(GPIOx, GPIO\_Pin))  
            data |= (1 << (7 - i));  
        while (HAL\_GPIO\_ReadPin(GPIOx, GPIO\_Pin));  
    }  
    return data;  
}

uint8\_t DHT11\_GetData(GPIO\_TypeDef \*GPIOx, uint16\_t GPIO\_Pin, DHT11\_Data \*data) {  
    uint8\_t Rh\_byte1, Rh\_byte2, Temp\_byte1, Temp\_byte2, checksum;

    DHT11\_Start(GPIOx, GPIO\_Pin);  
    if (DHT11\_Check\_Response(GPIOx, GPIO\_Pin)) {  
        Rh\_byte1 = DHT11\_Read(GPIOx, GPIO\_Pin);  
        Rh\_byte2 = DHT11\_Read(GPIOx, GPIO\_Pin);  
        Temp\_byte1 = DHT11\_Read(GPIOx, GPIO\_Pin);  
        Temp\_byte2 = DHT11\_Read(GPIOx, GPIO\_Pin);  
        checksum = DHT11\_Read(GPIOx, GPIO\_Pin);

        if (checksum == (Rh\_byte1 + Rh\_byte2 + Temp\_byte1 + Temp\_byte2)) {  
            data->Humidity = Rh\_byte1;  
            data->Temperature = Temp\_byte1;  
            return 1;  
        }  
    }  
    return 0;  
}

8.5 Water Level Sensor

**#include** "main.h"

**void** **SystemClock\_Config**(**void**);

**static** **void** **MX\_GPIO\_Init**(**void**);

**static** **void** **MX\_ADC1\_Init**(**void**);

\*/

uint32\_t adc\_value =0;

**int** **main**(**void**)

{

HAL\_Init();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

SystemClock\_Config();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_ADC1\_Init();

/\* USER CODE BEGIN 2 \*/

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1)

{

HAL\_ADC\_Start(&hadc1);

HAL\_ADC\_PollForConversion(&hadc1, HAL\_MAX\_DELAY);

adc\_value = HAL\_ADC\_GetValue(&hadc1);

**if** (adc\_value > 3000)

{

GPIOC->ODR |= (1 << 9);

HAL\_Delay(100);

GPIOC->ODR &=~(1<<9);

HAL\_Delay(100);

}

**else**

{

GPIOC->ODR &=~(1<<9);

}

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

**void** **SystemClock\_Config**(**void**)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

**if** (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != *HAL\_OK*)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_HSI;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

**if** (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_0) != *HAL\_OK*)

{

Error\_Handler();

}

}

/\*\*

\* @brief ADC1 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_ADC1\_Init**(**void**)

{

/\* USER CODE BEGIN ADC1\_Init 0 \*/

/\* USER CODE END ADC1\_Init 0 \*/

ADC\_ChannelConfTypeDef sConfig = {0};

/\* USER CODE BEGIN ADC1\_Init 1 \*/

/\* USER CODE END ADC1\_Init 1 \*/

/\*\* Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)

\*/

hadc1.Instance = ADC1;

hadc1.Init.ClockPrescaler = ADC\_CLOCK\_SYNC\_PCLK\_DIV2;

hadc1.Init.Resolution = ADC\_RESOLUTION\_12B;

hadc1.Init.ScanConvMode = *DISABLE*;

hadc1.Init.ContinuousConvMode = *DISABLE*;

hadc1.Init.DiscontinuousConvMode = *DISABLE*;

hadc1.Init.ExternalTrigConvEdge = ADC\_EXTERNALTRIGCONVEDGE\_NONE;

hadc1.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;

hadc1.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

hadc1.Init.NbrOfConversion = 1;

hadc1.Init.DMAContinuousRequests = *DISABLE*;

hadc1.Init.EOCSelection = ADC\_EOC\_SINGLE\_CONV;

**if** (HAL\_ADC\_Init(&hadc1) != *HAL\_OK*)

{

Error\_Handler();

}

/\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

\*/

sConfig.Channel = ADC\_CHANNEL\_4;

sConfig.Rank = 1;

sConfig.SamplingTime = ADC\_SAMPLETIME\_3CYCLES;

**if** (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != *HAL\_OK*)

{

Error\_Handler();

}

/\* USER CODE BEGIN ADC1\_Init 2 \*/

/\* USER CODE END ADC1\_Init 2 \*/

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_GPIO\_Init**(**void**)

{

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

RCC->AHB1ENR |=(1<<2);

GPIOC->MODER |=(1<<18);

GPIOC->MODER &=~(1<<19);

}

**#endif**

8.6 LCD.c

**#include** "lcd.h"

**#include** "cmn.h"

**void** **DelayLcd**(**void**)

{

uint32\_t i=0;

**for**(i=0;i<16800;i++);

}

**void** **LcdInit**(**void**)

{

RCC->AHB1ENR |=(RCC\_AHB1ENR\_GPIOAEN);

RCC->AHB1ENR |=(RCC\_AHB1ENR\_GPIOBEN);

SetOutput(PORT\_RS,PIN\_RS);

SetOutput(PORT\_EN,PIN\_EN);

SetOutput(PORT\_D4,PIN\_D4);

SetOutput(PORT\_D5,PIN\_D5);

SetOutput(PORT\_D6,PIN\_D6);

SetOutput(PORT\_D7,PIN\_D7);

LcdFxn(0,0x33);

LcdFxn(0,0x32);

LcdFxn(0,0x28);

LcdFxn(0,0x0c);

LcdFxn(0,0x01);

}

**void** **LcdFxn**(uint8\_t cmd,uint8\_t val)

{

**if**(val&(1<<7))

SetBit(PORT\_D7,PIN\_D7);

**else**

ClrBit(PORT\_D7,PIN\_D7);

**if**(val&(1<<6))

SetBit(PORT\_D6,PIN\_D6);

**else**

ClrBit(PORT\_D6,PIN\_D6);

**if**(val&(1<<5))

SetBit(PORT\_D5,PIN\_D5);

**else**

ClrBit(PORT\_D5,PIN\_D5);

**if**(val&(1<<4))

SetBit(PORT\_D4,PIN\_D4);

**else**

ClrBit(PORT\_D4,PIN\_D4);

**if**(cmd) SetBit(PORT\_RS,PIN\_RS);

**else** ClrBit(PORT\_RS,PIN\_RS);

SetBit(PORT\_EN,PIN\_EN);

DelayLcd();

ClrBit(PORT\_EN,PIN\_EN);

DelayLcd();

**if**(val&(1<<3))

SetBit(PORT\_D7,PIN\_D7);

**else**

ClrBit(PORT\_D7,PIN\_D7);

**if**(val&(1<<2))

SetBit(PORT\_D6,PIN\_D6);

**else**

ClrBit(PORT\_D6,PIN\_D6);

**if**(val&(1<<1))

SetBit(PORT\_D5,PIN\_D5);

**else**

ClrBit(PORT\_D5,PIN\_D5);

**if**(val&(1<<0))

SetBit(PORT\_D4,PIN\_D4);

**else**

ClrBit(PORT\_D4,PIN\_D4);

**if**(cmd) SetBit(PORT\_RS,PIN\_RS);

**else** ClrBit(PORT\_RS,PIN\_RS);

SetBit(PORT\_EN,PIN\_EN);

DelayLcd();

ClrBit(PORT\_EN,PIN\_EN);

DelayLcd();

}

**void** **lprint**(uint8\_t add, **char** \*str)

{

uint8\_t i=0;

LcdFxn(0,add);

**while**(str[i]!=0)

{

LcdFxn(1,str[i]);

str++;

**if**(i>16)

**break**;

}

}

**void** **aprint**(uint8\_t addr,uint32\_t dval)

{

**long** **int** var=0;

**unsigned** **char** d1,d2,d3,d4=0;

var=dval;

d1=var%10;

var=var/10;

d2=var%10;

var=var/10;

d3=var%10;

d4=var/10;

LcdFxn(0,addr);

LcdFxn(1,d4|0x30);

LcdFxn(1,d3|0x30);

LcdFxn(1,d2|0x30);

LcdFxn(1,d1|0x30);

}

// Function to clear LCD

**void** **LcdClear**(**void**) {

LcdFxn(0, 0x01); // Send "clear display" command

DelayLcd(); // Small delay

}

Chapter 9

Conclusion

The Vehicle Health Management System successfully monitors key vehicle parameters in real-time, including fuel level, engine temperature, motion, and obstacles. By providing timely alerts and controlling components like the cooling fan, it enhances safety, efficiency, and reliability. This system serves as a practical and scalable solution for safer and smarter driving, with potential for future improvements like IoT integration and predictive maintenance.​

​

Key Points:​

* Provides real-time monitoring and alerts for critical vehicle conditions.​
* Enhances driver safety, convenience, and vehicle reliability.​
* Offers a scalable and future-ready system for advanced vehicle management.​

​

9.1Future Scopes

1.Adding GPS for location monitoring

2.Predictive maintenance using sensor data and AI analytics.​

3.Addition of GPS tracking and route optimization for fleet vehicles.​

4.Expansion to hybrid and electric vehicles with battery and energy monitoring.​

5.Incorporation of advanced safety alerts, like automatic braking or lane departure warnings.​